Package ‘designmatch’

February 20, 2026

Type Package

Title Matched Samples that are Balanced and Representative by Design

Version 0.5.5

Description Includes functions for the construction of matched samples that are balanced and representative by design. Among others, these functions can be used for matching in observational studies with treated and control units, with cases and controls, in related settings with instrumental variables, and in discontinuity designs. Also, they can be used for the design of randomized experiments, for example, for matching before randomization. By default, 'designmatch' uses the 'highs' optimization solver, but its performance is greatly enhanced by the 'Gurobi' optimization solver and its associated R interface. For their installation, please follow the instructions at <https://www.gurobi.com/getting-started/> and <https://docs.gurobi.com/projects/optimizer/en/current/reference/r/setup.html>. We have also included directions in the gurobi_installation file in the inst folder.

Depends R (>= 4.4.0)

Imports MASS, slam, highs (>= 1.12.0), stats, graphics

Suggests gurobi, Rcplex, Rglpk, Rmosek, Rsymphony

License GPL-2 | GPL-3

Encoding UTF-8

URL https://github.com/jrzubizarreta/designmatch

BugReports https://github.com/jrzubizarreta/designmatch/issues

NeedsCompilation no

Author Jose R. Zubizarreta [aut, cre] (ORCID:
<https://orcid.org/0000-0002-0322-147X>),
Cinar Kilcioglu [aut],
Juan P. Vielma [aut],
Eric R. Cohn [aut],
Noah Greifer [ctb] (ORCID: <https://orcid.org/0000-0003-3067-7154>)

Maintainer Jose R. Zubizarreta <zubizarreta@hcp.med.harvard.edu>

Repository CRAN

Date/Publication 2026-02-20 08:50:02 UTC

Contents

`designmatch-package` *Optimal Matched Design of Randomized Experiments and Observational Studies*

Description

designmatch includes two functions for the construction of matched samples that are balanced and representative by design. These two functions are `bmatch` and `nmatch` for bipartite and nonbipartite matching, respectively. Both functions include options for directly balancing means, higher order moments, and distributions of the observed covariates. In both `bmatch` and `nmatch`, an integer programming (IP) problem is solved. This IP problem either minimizes the total sum of covariate distances between matched units, maximizes the total number of matched units, or optimizes a combination of the two, subject to matching and covariate balancing constraints. In order to solve these problems, four different optimization solvers can be used: CPLEX, GLPK, Gurobi, HiGHS, and Symphony. By default, both `bmatch` and `nmatch` solve a relaxation of these integer programs using HiGHS, which runs quickly but may violate to some extent some of the balancing constraints. If the user wants to solve for an exact solution of the program, we strongly recommend using either CPLEX or Gurobi, which are much faster but require a license (free for academic users) and special installation (see the installation instructions). Between the two, Gurobi is considerably easier to install. Among others, **designmatch** can be used for matching in treatment-control as well as case-control observational studies; observational studies with instrumental variables and discontinuity designs; and for the design of randomized experiments, for example for matching before randomization. The package also includes functions for assessing covariate balance in the matched samples.

Details

Package:	designmatch
Type:	Package
Version:	0.5.5
License:	GPL-2 | GPL-3

Author(s)

Jose R. Zubizarreta, Cinar Kilcioglu.

References

Greevy, R., Lu, B., Silber, J. H., and Rosenbaum, P. R. (2004), "Optimal Multivariate Matching Before Randomization," *Biostatistics*, 5, 263-275.

Hsu. J., Zubizarreta, J. R., Small, D. S., and Rosenbaum, P. R. (2015), "Strong Control of the Family-Wise Error Rate in Observational Studies that Discover Effect Modification by Exploratory Methods," *Biometrika*, 102, 767-782.

Keele, L., Titiunik, R., and Zubizarreta, J. R., (2015), "Enhancing a Geographic Regression Discontinuity Design Through Matching to Estimate the Effect of Ballot Initiatives on Voter Turnout," *Journal of the Royal Statistical Society: Series A*, 178, 223-239.

Kilcioglu, C., & Zubizarreta, J. R. (2016). "Maximizing the information content of a balanced matched sample in a study of the economic performance of green buildings". *The Annals of Applied Statistics*, 10(4), 1997-2020. doi:10.1214/16AOAS962

Lu, B., Greevy, R., Xu, X., and Beck C. (2011), "Optimal Nonbipartite Matching and its Statistical Applications," *The American Statistician*, 65, 21-30.

Rosenbaum, P. R. (2010), *Design of Observational Studies*, Springer.

Rosenbaum, P. R. (2012), "Optimal Matching of an Optimally Chosen Subset in Observational studies," *Journal of Computational and Graphical Statistics*, 21, 57-71.

Yang, D., Small, D., Silber, J. H., and Rosenbaum, P. R. (2012), "Optimal Matching With Minimal Deviation From Fine Balance in a Study of Obesity and Surgical Outcomes," *Biometrics*, 68, 628-636.

Yang. F., Zubizarreta, J. R., Small, D. S., Lorch, S. A., and Rosenbaum, P. R. (2014), "Dissonant Conclusions When Testing the Validity of an Instrumental Variable," *The American Statistician*, 68, 253-263.

Zou, J., and Zubizarreta, J. R., (2015) "Covariate Balanced Restricted Randomization: Optimal Designs, Exact Tests, and Asymptotic Results," working paper.

Zubizarreta, J. R., Reinke, C. E., Kelz, R. R., Silber, J. H., and Rosenbaum, P. R. (2011), "Matching for Several Sparse Nominal Variables in a Case-Control Study of Readmission Following Surgery," *The American Statistician*, 65, 229-238.

Zubizarreta, J. R. (2012), "Using Mixed Integer Programming for Matching in an Observational Study of Kidney Failure after Surgery," *Journal of the American Statistical Association*, 107, 1360-1371.

Zubizarreta, J. R., Paredes, R. D., and Rosenbaum, P. R. (2014), "Matching for Balance, Pairing for Heterogeneity in an Observational Study of the Effectiveness of For-profit and Not-for-profit High Schools in Chile," *Annals of Applied Statistics*, 8, 204-231.

`absstddif` *Absolute standardized differences in means.*

Description

Function for calculating absolute differences in means between the covariates in the treatment and control groups in terms of the original units of the covariates. Here, the absolute differences in means are normalized by the simple average of the treated and control standard deviations before matching (see Rosenbaum and Rubin 1985 for details).

Usage

```
absstddif(X_mat, t_ind, std_dif)
```

Arguments

`X_mat`	matrix of covariates: a matrix of covariates used to build the rank-based Mahalanobis distance matrix.
`t_ind`	treatment indicator: a vector of zeros and ones indicating treatment (1 = treated; 0 = control).
`std_dif`	standardized differences: a scalar determining the number of absolute standardized differences.

Value

A vector that can be used with the `mom`, `near` and `far` options of `bmatch` and `nmatch`.

Author(s)

Jose R. Zubizarreta, Cinar Kilcioglu.

References

Rosenbaum, P. R., and Rubin, D. B. (1985), "Constructing a Control Group by Multivariate Matched Sampling Methods that Incorporate the Propensity Score," *The American Statistician*, 39, 33-38.

Examples

```
# Load and attach data
data(lalonde)
attach(lalonde)

# Treatment indicator
t_ind = treatment
```

```
# Constrain differences in means to be at most .05 standard deviations apart
mom_covs = cbind(age, education, black, hispanic, married, nodegree, re74, re75)
mom_tols = absstddif(mom_covs, t_ind, .05)
```

`bmatch` *Optimal bipartite matching in observational studies*

Description

Function for optimal bipartite matching in observational studies that directly balances the observed covariates. `bmatch` allows the user to enforce different forms of covariate balance in the matched samples such as moment balance (e.g., of means, variances and correlations), distributional balance (e.g., fine balance, near-fine balance, strength-*k* balancing) and exact matching. In observational studies where an instrumental variable is available, `bmatch` can be used to handle weak instruments and strengthen them by means of the `far` options (see Yang et al. 2014 for an example). `bmatch` can also be used in discontinuity designs by matching units in a neighborhood of the discontinuity (see Keele et al. 2015 for details). In any of these settings, `bmatch` either minimizes the total sum of covariate distances between matched units, maximizes the total number of matched units, or optimizes a combination of the two, subject to matching, covariate balancing, and representativeness constraints (see the examples below).

Usage

```
bmatch(t_ind, dist_mat = NULL, subset_weight = NULL, n_controls = 1,
       total_groups = NULL, mom = NULL, ks = NULL, exact = NULL,
       near_exact = NULL, fine = NULL, near_fine = NULL, near = NULL,
       far = NULL, solver = NULL)
```

Arguments

`t_ind`	treatment indicator: a vector of zeros and ones indicating treatment (1 = treated; 0 = control). Please note that the data needs to be sorted in decreasing order according to this treatment indicator.
`dist_mat`	distance matrix: a matrix of positive distances between treated units (rows) and controls (columns). If `dist_mat = NULL` and `subset_weight = 1`, then bmatch will solve the cardinality matching problem in Zubizarreta et al. (2014).
`subset_weight`	subset matching weight: a scalar that regulates the trade-off between the total sum of distances between matched pairs and the total number of matched pairs. The larger `subset_weight`, the more importance will be given to the the total number of matched pairs relative to the total sum of distances between matched pairs. See Rosenbaum (2012) and Zubizarreta et al. (2013) for a discussion of this parameter. If `subset_weight = NULL`, then `bmatch` will match all the treated observations, provided there exists a feasible solution. If `subset_weight = 1` and `dist_mat = NULL`, then `bmatch` will solve the cardinality matching problem in Zubizarreta et al. (2014).

`n_controls`	number of controls: a scalar defining the number of controls to be matched with a fixed rate to each treated unit. The default is `n_controls = 1`, i.e. pair matching.
`total_groups`	total number of matched pairs: a scalar specifying the number of matched pairs to be obtained. If `total_groups = NULL`, then no specific number of matched pairs is required before matching.
`mom`	moment balance parameters: a list with three arguments, `mom = list(covs = mom_covs, tols = mom_tols, targets = mom_targets)`. `mom_covs` is a matrix where each column is a covariate whose mean is to be balanced. `mom_tols` is a vector of tolerances for the maximum difference in means for the covariates in `mom_covs`. `mom_targets` is a vector of target moments (e.g., means) of a distribution to be approximated by matched sampling. `mom_targets` is optional, but if `mom_covs` is specified then `mom_tols` needs to be specified too. If `mom_targets` is `NULL`, then `bmatch` will match treated and control units so that covariates in `mom_covs` differ at most by `mom_tols`. If `mom_targets` is specified, then `bmatch` will match treated and control units so that each matched group differs at most by `mom_tols` units from the respective moments in `mom_targets`. As a result, the matched groups will differ at most `mom_tols * 2` from each other. Under certain assumptions, `mom_targets` can be used for constructing a representative matched sample. The lengths of `mom_tols` and `mom_target` have to be equal to the number of columns of `mom_covs`. Note that the columns of `mom_covs` can be transformations of the original covariates to balance higher order single-dimensional moments like variances and skewness, and multidimensional moments such as correlations (Zubizarreta 2012).
`ks`	Kolmogorov-Smirnov balance parameters: a list with three objects, `ks = list(covs = ks_covs, n_grid = ks_n_grid, tols = ks_tols)`. `ks_covs` is a matrix where each column is a covariate whose difference in a coarsened version of the Kolmogorov-Smirnov statistic between treated units and matched controls is to be constrained. `ks_n_grid` is scalar defining the number of equispaced grid points for calculating the Kolmogorov-Smirnov statistic between the distributions of the treated units and matched controls for the covariates in `ks_covs`. The default is `ks_n_grid = 10` for the deciles of the empirical cumulative distribution functions of the treated units for the covariate in question. `ks_tols` is a vector of tolerances for the maximum differences in means for the covariates defined in `ks_covs`. Note that the length of `ks_tols` has to be equal to the number of columns of `ks_covs`.
`exact`	Exact matching parameters: a list with one argument, `exact = list(covs = exact_covs)`, where `exact_covs` is a matrix where each column is a nominal covariate for exact matching.
`near_exact`	Near-exact matching parameters: a list with two objects, `near_exact = list(covs = near_exact_covs, devs = near_exact_devs)`. `near_exact_covs` are the near-exact matching covariates; specifically, a matrix where each column is a nominal covariate for near-exact matching. `near_exact_devs` are the maximum deviations from near-exact matching: a vector of scalars defining the maximum deviation allowed from exact matching for the covariates defined in `near_exact_covs`. Note that the length of `near_exact_devs` has to be

	equal to the number of columns of `near_exact_covs`. For detailed expositions of near-exact matching, see section 9.2 of Rosenbaum (2010) and Zubizarreta et al. (2011).
`fine`	Fine balance parameters: a list with one argument, `fine = list(covs = fine_covs)`, where `fine_covs` is a matrix where each column is a nominal covariate for fine balance. Fine balance enforces exact distributional balance on nominal covariates, but without constraining treated and control units to be matched within each category of each nominal covariate as in exact matching. See chapter 10 of Rosenbaum (2010) for details.
`near_fine`	Near-fine balance parameters: a list with two objects, `near_fine = list(covs = near_fine_covs, devs = near_fine_devs)`. `near_fine_covs` is a matrix where each column is a nominal covariate for near-fine matching. `near_fine_devs` is a vector of scalars defining the maximum deviation allowed from fine balance for the covariates in `near_fine_covs`. Note that the length of `near_fine_devs` has to be equal to the number of columns of `near_fine_covs`. See Yang et al. (2012) for a description of near-fine balance.
`near`	Near matching parameters: a list with three objects, `near = list(covs = near_covs, pairs = near_pairs, groups = near_groups)`. `near_covs` is a matrix where each column is a variable for near matching. `near_pairs` is a vector determining the maximum distance between individual matched pairs for each variable in `near_covs`. `near_groups` is a vector determining the maximum average distance between matched groups (in aggregate) for each covariate in `near_covs`. If `near_covs` is specified, then either `near_pairs`, `near_covs` or both must be specified as well, and the length of `near_pairs` and/or `near_groups` has to be equal to the number of columns of `near_covs`. The `near` options can be useful for matching within a window of a discontinuity (see Keele et al. 2015 for details).
`far`	Far matching parameters: a list with three objects, `far = list(covs = far_covs, pairs = far_pairs, groups = far_groups)`. `far_covs` is a matrix where each column is a variable (a covariate or an instrumental variable) for far matching. `far_pairs` is a vector determining the minimum distance between units in a matched pair for each variable in `far_covs`, and `far_groups` is a vector defining the minimum average (aggregate) distance between matched groups for each variable in `far_covs`. If `far_covs` is specified, then either `far_pairs`, `far_covs`, or both, must be specified, and the length of `far_pairs` and/or `far_groups` has to be equal to the number of columns of `far_covs`. See Yang et al. (2014) for strengthening an instrumental variable via far matching in a bipartite setting.
`solver`	Optimization solver parameters: a list with four objects, `solver = list(name = name, t_max = t_max, approximate = 1, round_cplex = 0,` `trace_cplex = 0)`. `solver` is a string that determines the optimization solver to be used. The options are: `cplex`, `glpk`, `gurobi`, `highs`, and `symphony`. The default solver is `highs` with `approximate = 1`, so that by default an approximate solution is

found (see approximate below). For an exact solution, we strongly recommend using cplex or gurobi as they are much faster than the other solvers, but they do require a license (free for academics, but not for people outside universities). Between cplex and gurobi, note that installing the R interface for gurobi is much simpler.

t_max is a scalar with the maximum time limit for finding the matches. This option is specific to cplex and gurobi. If the optimal matches are not found within this time limit, a partial, suboptimal solution is given.

approximate is a scalar that determines the method of solution. If approximate = 1 (the default), an approximate solution is found via a relaxation of the original integer program. This method of solution is faster than approximate = 0, but some balancing constraints may be violated to some extent. This option works only with n_controls = 1, i.e. pair matching.

round_cplex is binary specific to cplex. round_cplex = 1 ensures that the solution found is integral by rounding and all the constraints are exactly satisfied; round_cplex = 0 (the default) encodes there is no rounding which may return slightly infeasible integer solutions.

trace is a binary specific to cplex and gurobi. trace = 1 turns the optimizer output on. The default is trace = 0.

Value

A list containing the optimal solution, with the following objects:

obj_total	value of the objective function at the optimum;
obj_dist_mat	value of the total sum of distances term of the objective function at the optimum;
t_id	indexes of the matched treated units at the optimum;
c_id	indexes of the matched controls at the optimum;
group_id	matched pairs or groups at the optimum;
time	time elapsed to find the optimal solution.

Author(s)

Jose R. Zubizarreta, Cinar Kilcioglu.

References

Keele, L., Titiunik, R., and Zubizarreta, J. R., (2015), "Enhancing a Geographic Regression Discontinuity Design Through Matching to Estimate the Effect of Ballot Initiatives on Voter Turnout," *Journal of the Royal Statistical Society: Series A*, 178, 223-239.

Rosenbaum, P. R. (2010), *Design of Observational Studies*, Springer.

Rosenbaum, P. R. (2012), "Optimal Matching of an Optimally Chosen Subset in Observational studies," *Journal of Computational and Graphical Statistics*, 21, 57-71.

Yang, D., Small, D., Silber, J. H., and Rosenbaum, P. R. (2012), "Optimal Matching With Minimal Deviation From Fine Balance in a Study of Obesity and Surgical Outcomes," *Biometrics*, 68, 628-36.

Yang. F., Zubizarreta, J. R., Small, D. S., Lorch, S. A., and Rosenbaum, P. R. (2014), "Dissonant Conclusions When Testing the Validity of an Instrumental Variable," *The American Statistician*, 68, 253-263.

Zubizarreta, J. R., Reinke, C. E., Kelz, R. R., Silber, J. H., and Rosenbaum, P. R. (2011), "Matching for Several Sparse Nominal Variables in a Case-Control Study of Readmission Following Surgery," *The American Statistician*, 65, 229-238.

Zubizarreta, J. R. (2012), "Using Mixed Integer Programming for Matching in an Observational Study of Kidney Failure after Surgery," *Journal of the American Statistical Association*, 107, 1360-1371.

Zubizarreta, J. R., Paredes, R. D., and Rosenbaum, P. R. (2014), "Matching for Balance, Pairing for Heterogeneity in an Observational Study of the Effectiveness of For-profit and Not-for-profit High Schools in Chile," *Annals of Applied Statistics*, 8, 204-231.

See Also

sensitivitymv, **sensitivitymw**.

Examples

```
## Uncomment the following examples
## Load, sort, and attach data
#data(lalonde)
#lalonde = lalonde[order(lalonde$treatment, decreasing = TRUE), ]
#attach(lalonde)

#################################
## Example 1: cardinality matching
#################################

## Cardinality matching finds the largest matched sample of pairs that meets balance
## requirements.  Here the balance requirements are mean balance, fine balance and
## exact matching for different covariates. The solver used is glpk with the
## approximate option.

## Treatment indicator; note that the data needs to be sorted in decreasing order
## according to this treatment indicator
#t_ind = treatment
#t_ind

## Distance matrix
#dist_mat = NULL

## Subset matching weight
#subset_weight = 1

## Moment balance: constrain differences in means to be at most .05 standard deviations apart
#mom_covs = cbind(age, education, black, hispanic, married, nodegree, re74, re75)
#mom_tols = round(absstddif(mom_covs, t_ind, .05), 2)
#mom = list(covs = mom_covs, tols = mom_tols)

## Fine balance
```

```
#fine_covs = cbind(black, hispanic, married, nodegree)
#fine = list(covs = fine_covs)

## Exact matching
#exact_covs = cbind(black)
#exact = list(covs = exact_covs)

## Solver options
#t_max = 60*5
#solver = "glpk"
#approximate = 1
#solver = list(name = solver, t_max = t_max, approximate = approximate,
#round_cplex = 0, trace = 0)

## Match
#out = bmatch(t_ind = t_ind, dist_mat = dist_mat, subset_weight = subset_weight,
#mom = mom, fine = fine, exact = exact, solver = solver)

## Indices of the treated units and matched controls
#t_id = out$t_id
#c_id = out$c_id

## Time
#out$time/60

## Matched group identifier (who is matched to whom)
#out$group_id

## Assess mean balance
#meantab(mom_covs, t_ind, t_id, c_id)

## Assess fine balance (note here we are getting an approximate solution)
#for (i in 1:ncol(fine_covs)) {
#  print(finetab(fine_covs[, i], t_id, c_id))
#}

## Assess exact matching balance
#table(exact_covs[t_id]==exact_covs[c_id])

##################################
## Example 2: minimum distance matching
##################################

## The goal here is to minimize the total of distances between matched pairs.  In
## this example there are no covariate balance requirements.  Again, the solver
## used is glpk with the approximate option

## Treatment indicator
#t_ind = treatment

## Matrix of covariates
#X_mat = cbind(age, education, black, hispanic, married, nodegree, re74, re75)
```

```
## Distance matrix
#dist_mat = distmat(t_ind, X_mat)

## Subset matching weight
#subset_weight = NULL

## Total pairs to be matched
#total_groups = sum(t_ind)

## Solver options
#t_max = 60*5
#solver = "glpk"
#approximate = 1
#solver = list(name = solver, t_max = t_max, approximate = approximate,
#round_cplex = 0, trace_cplex = 0)

## Match
#out = bmatch(t_ind = t_ind, dist_mat = dist_mat, total_groups = total_groups,
#solver = solver)

## Indices of the treated units and matched controls
#t_id = out$t_id
#c_id = out$c_id

## Total of distances between matched pairs
#out$obj_total

## Assess mean balance
#meantab(X_mat, t_ind, t_id, c_id)

##################################
## Example 3: optimal subset matching
##################################

## Optimal subset matching pursues two competing goals at
## the same time: to minimize the total sum of covariate distances
## while matching as many observations as possible.  The trade-off
## between these two goals is regulated by the parameter subset_weight
## (see Rosenbaum 2012 and Zubizarreta et al. 2013 for a discussion).
## Here the balance requirements are mean balance, near-fine balance
## and near-exact matching for different covariates.
## Again, the solver used is glpk with the approximate option.

## Treatment indicator
#t_ind = treatment

## Matrix of covariates
#X_mat = cbind(age, education, black, hispanic, married, nodegree, re74, re75)

## Distance matrix
#dist_mat = distmat(t_ind, X_mat)

## Subset matching weight
```

```
#subset_weight = median(dist_mat)

## Moment balance: constrain differences in means to be at most .05 standard deviations apart
#mom_covs = cbind(age, education, black, hispanic, married, nodegree, re74, re75)
#mom_tols = round(absstddif(mom_covs, t_ind, .05), 2)
#mom = list(covs = mom_covs, tols = mom_tols)

## Near-fine balance
#near_fine_covs = cbind(married, nodegree)
#near_fine_devs = rep(5, 2)
#near_fine = list(covs = near_fine_covs, devs = near_fine_devs)

## Near-exact matching
#near_exact_covs = cbind(black, hispanic)
#near_exact_devs = rep(5, 2)
#near_exact = list(covs = near_exact_covs, devs = near_exact_devs)

## Solver options
#t_max = 60*5
#solver = "glpk"
#approximate = 1
#solver = list(name = solver, t_max = t_max, approximate = approximate,
#round_cplex = 0, trace_cplex = 0)

## Match
#out = bmatch(t_ind = t_ind, dist_mat = dist_mat, subset_weight = subset_weight,
#mom = mom, near_fine = near_fine, near_exact = near_exact, solver = solver)

## Indices of the treated units and matched controls
#t_id = out$t_id
#c_id = out$c_id

## Time
#out$time/60

## Matched group identifier (who is matched to whom)
#out$group_id

## Assess mean balance (note here we are getting an approximate solution)
#meantab(X_mat, t_ind, t_id, c_id)

## Assess fine balance
#for (i in 1:ncol(near_fine_covs)) {
# print(finetab(near_fine_covs[, i], t_id, c_id))
#}

## Assess exact matching balance
#for (i in 1:ncol(near_exact_covs)) {
# print(table(near_exact_covs[t_id, i]==near_exact_covs[c_id, i]))
#}
```

`cardmatch` *Optimal cardinality matching in observational studies*

Description

Function for optimal cardinality matching in observational studies. `cardmatch` finds the largest matched sample that is balanced and representative by design. The formulation of `cardmatch` has been simplified to handle larger data than `bmatch` or `nmatch`. Similar to `bmatch` or `nmatch`, the performance of `cardmatch` is greatly enhanced by using the `solver` options `cplex` or `gurobi`.

Usage

```
cardmatch(t_ind, mom = NULL, fine = NULL, solver = NULL)
```

Arguments

`t_ind` treatment indicator: a vector of zeros and ones indicating treatment (1 = treated; 0 = control). Please note that the data needs to be sorted in decreasing order according to this treatment indicator.

`mom` moment balance parameters: a list with three arguments,

`mom = list(covs = mom_covs, tols = mom_tols, targets = mom_targets)`.

`mom_covs` is a matrix where each column is a covariate whose mean is to be balanced. `mom_tols` is a vector of tolerances for the maximum difference in means for the covariates in `mom_covs`. `mom_targets` is a vector of target moments (e.g., means) of a distribution to be approximated by matched sampling. If `mom_targets` is specified, then `cardmatch` will select units from each treatment group so that each matched group differs at most by `mom_tols` units from the respective moments in `mom_targets`. As a result, the matched groups will differ at most `mom_tols * 2` from each other. Under certain assumptions, `mom_targets` can be used for constructing a representative matched sample. The lengths of `mom_tols` and `mom_target` have to be equal to the number of columns of `mom_covs`. Note that the columns of `mom_covs` can be transformations of the original covariates to balance higher order single-dimensional moments like variances and skewness, and multidimensional moments such as correlations (Zubizarreta 2012).

`fine` Fine balance parameters: a list with one argument,

`fine = list(covs = fine_covs)`,

where `fine_covs` is a matrix where each column is a nominal covariate for fine balance. Fine balance enforces exact distributional balance on nominal covariates, but without constraining treated and control units to be matched within each category of each nominal covariate as in exact matching. See chapter 10 of Rosenbaum (2010) for details.

`solver` Optimization solver parameters: a list with four objects,

`solver = list(name = name, t_max = t_max, approximate = 1, round_cplex = 0,`
`trace_cplex = 0)`.

`solver` is a string that determines the optimization solver to be used. The options are: `cplex`, `glpk`, `gurobi`, `highs`, and `symphony`. The default solver is `highs` with `approximate = 1`. For an exact solution, we strongly recommend using `cplex` or `gurobi` as they are much faster than the other solvers, but they do require a license (free for academics, but not for people outside universities). Between `cplex` and `gurobi`, note that installing the R interface for `gurobi` is much simpler.

`t_max` is a scalar with the maximum time limit for finding the matches. This option is specific to `cplex`, `gurobi`, and `highs`. If the optimal matches are not found within this time limit, a partial, suboptimal solution is given.

`approximate` is a scalar that determines the method of solution. If `approximate = 1` (the default), an approximate solution is found via a relaxation of the original integer program. This method of solution is faster than `approximate = 0`, but some balancing constraints may be violated to some extent. This option works only with `n_controls = 1`, i.e. pair matching.

`round_cplex` is binary specific to `cplex`. `round_cplex = 1` ensures that the solution found is integral by rounding and all the constraints are exactly satisfied; `round_cplex = 0` (the default) encodes there is no rounding which may return slightly infeasible integer solutions.

`trace` is a binary specific to `cplex` and `gurobi`. `trace = 1` turns the optimizer output on. The default is `trace = 0`.

Value

A list containing the optimal solution, with the following objects:

`obj_total`	value of the objective function at the optimum;
`obj_dist_mat`	value of the total sum of distances term of the objective function at the optimum;
`t_id`	indexes of the matched treated units at the optimum;
`c_id`	indexes of the matched controls at the optimum;
`group_id`	matched pairs or groups at the optimum;
`time`	time elapsed to find the optimal solution.

Author(s)

Jose R. Zubizarreta, Cinar Kilcioglu, Juan P. Vielma.

References

Bennett, M., Vielma, J. P., & Zubizarreta, J. R. (2020). "Building Representative Matched Samples With Multi-Valued Treatments in Large Observational Studies". *Journal of Computational and Graphical Statistics*, 1-29. doi:10.1080/10618600.2020.1753532

Visconti, G., & Zubizarreta, J. R. (2018). "Handling Limited Overlap in Observational Studies with Cardinality Matching". *Observational Studies*, 4(1), 217-249. doi:10.1353/obs.2018.0012

Zubizarreta, J. R. (2012), "Using Mixed Integer Programming for Matching in an Observational Study of Kidney Failure after Surgery," *Journal of the American Statistical Association*, 107, 1360-1371.

Zubizarreta, J. R., Paredes, R. D., and Rosenbaum, P. R. (2014), "Matching for Balance, Pairing for Heterogeneity in an Observational Study of the Effectiveness of For-profit and Not-for-profit High Schools in Chile," *Annals of Applied Statistics*, 8, 204-231.

See Also

sensitivitymv, **sensitivitymw**.

Examples

```
# Load, sort, and attach data
data(lalonde)
lalonde = lalonde[order(lalonde$treatment, decreasing = TRUE), ]
attach(lalonde)

#################################
# Step 1: use cardinality matching to find the largest sample of matched pairs for which
# all the covariates are finely balanced.
#################################

# Discretize covariates
quantiles = function(covar, n_q) {
p_q = seq(0, 1, 1/n_q)
val_q = quantile(covar, probs = p_q, na.rm = TRUE)
covar_out = rep(NA, length(covar))
for (i in 1:n_q) {
if (i == 1) {covar_out[covar < val_q[i+1]] = i}
else if (i < n_q) {covar_out[covar >= val_q[i] & covar < val_q[i+1]] = i}
else {covar_out[covar >= val_q[i] & covar <= val_q[i+1]] = i}
}
covar_out
}
age_5 = quantiles(age, 5)
education_5 = quantiles(education, 5)
re74_5 = quantiles(re74, 5)
re75_5 = quantiles(re75, 5)

# Treatment indicator; note that the data needs to be sorted in decreasing order
# according to this treatment indicator
t_ind = treatment
t_ind

# Fine balance
fine_covs = cbind(black, hispanic, married, nodegree, age_5, education_5, re74_5, re75_5)
fine = list(covs = fine_covs)

# Solver options
t_max = 60*5
solver = "highs"
approximate = 0
solver = list(name = solver, t_max = t_max, approximate = approximate,
round_cplex = 0, trace = 0)
```

```
# Match
out_1 = cardmatch(t_ind, fine = fine, solver = solver)

# Indices of the treated units and matched controls
t_id_1 = out_1$t_id
c_id_1 = out_1$c_id

# Mean balance
covs = cbind(age, education, black, hispanic, married, nodegree, re74, re75)
meantab(covs, t_ind, t_id_1, c_id_1)

# Fine balance (note here we are getting an approximate solution)
for (i in 1:ncol(fine_covs)) {
print(finetab(fine_covs[, i], t_id_1, c_id_1))
}

#################################
# Step 2: use optimal matching (minimum distance matching) to find the (re)pairing of
# treated and control that minimizes the total sum of covariate distances between matched
# pairs.  For this, use the function 'distmatch' which is a wrapper for 'bmatch'.
#################################

# New treatment indicator
t_ind_2 = t_ind[c(t_id_1, c_id_1)]
table(t_ind_2)

# To build the distance matrix, the idea is to use strong predictors of the outcome
dist_mat_2 = abs(outer(re74[t_id_1], re74[c_id_1], "-"))
dim(dist_mat_2)

# Match
out_2 = distmatch(t_ind_2, dist_mat_2, solver)

# Indices of the treated units and matched controls
t_id_2 = t_id_1[out_2$t_id]
c_id_2 = c_id_1[out_2$c_id-length(out_2$c_id)]

# Covariate balance is preserved...
meantab(covs, t_ind, t_id_2, c_id_2)
for (i in 1:ncol(fine_covs)) {
print(finetab(fine_covs[, i], t_id_2, c_id_2))
}

# ... but covariate distances are reduced
distances_step_1 = sum(diag(dist_mat_2))
distances_step_2 = sum(diag(dist_mat_2[out_2$t_id, out_2$c_id-length(out_2$c_id)]))
distances_step_1
distances_step_2

# The mean difference in outcomes is the same...
mean(re78[t_id_1]-re78[c_id_1])
mean(re78[t_id_2]-re78[c_id_2])
```

```
# ... but their standard deviation is reduced
sd(re78[t_id_1]-re78[c_id_1])
sd(re78[t_id_2]-re78[c_id_2])
```

`distmat` *Build a rank-based Mahalanobis distance matrix*

Description

Function for building a normalized rank-based Mahalanobis distance matrix with a penalty for caliper violation.

Usage

```
distmat(t_ind, X_mat, calip_cov = NULL, calip_size = NULL, calip_penalty = NULL,
        near_exact_covs = NULL, near_exact_penalties = NULL, digits = 1)
```

Arguments

`t_ind`	treatment indicator: a vector of zeros and ones indicating treatment (1 = treated; 0 = control).
`X_mat`	matrix of covariates: a matrix of covariates used to build the based Mahalanobis distance matrix.
`calip_cov`	caliper covariate: a covariate vector used to define the caliper. In most applications this is the propensity score, but a covariate can be used as well.
`calip_size`	caliper size: a scalar that determines the size of the caliper for which there will be no penalty. Most applications use `0.2*sd(calip_cov)`.
`calip_penalty`	a scalar used to multiply the magnitude of the violation of the caliper.
`near_exact_covs`	a matrix of covariates used for near-exact matching.
`near_exact_penalties`	a vector of scalars used for near-exact matching. The length of `near_exact_penalties` has to be equal to the number of columns of `near_exact_covs`.
`digits`	a scalar indicating the number of digits used to produce each entry of the distance matrix. The default is 1 digit.

Details

`distmat` is a function for building a normalized rank-based Mahalanobis distance matrix with a penalty for caliper violations on a covariate (say, the propensity score) and penalties for near-exact matching.

As explained in Rosenbaum (2010), the use of a rank-based Mahalanobis distance prevents an outlier from inflating the variance for a variable, and it thus decreases its importance in the matching. In the calculation of the matrix the variances are constrained to not decrease as ties become more common, so that it is not more important to match on a rare binary variable than on a common binary one. The penalty for caliper violations ensures good balance on the propensity score or the covariate used. In this way the rank-based Mahalanobis distance with a penalty for caliper violations in the propensity score constitutes a robust distance for matching.

As explained in Zubizarreta et al. (2011), the distance matrix can also be modified for near-exact matching. Penalties are added to the distance matrix every time that a treated and a control unit have a different value for the corresponding near-exact matching covariate.

Value

A matrix that can be used for optimal matching with the `bmatch` functions in the `designmatch` package.

References

Rosenbaum, P. R. (2010), *Design of Observational Studies*, Springer.

Zubizarreta, J. R., Reinke, C. E., Kelz, R. R., Silber, J. H., and Rosenbaum, P. R. (2011), "Matching for Several Sparse Nominal Variables in a Case-Control Study of Readmission Following Surgery," *The American Statistician*, 65, 229-238.

Examples

```
# Load data
data(germancities)
attach(germancities)

# Treatment indicator
t_ind = treat

# Matrix of covariates
X_mat = cbind(log2pop, popgrowth1939, popgrowth3339, emprate, indrate, rubble,
rubblemiss, flats, flatsmiss, refugees)

# Distance matrix
dist_mat = distmat(t_ind, X_mat)
```

`distmatch` *Optimal distance matching in observational studies*

Description

Function for optimal distance matching in observational studies. `distmatch` minimizes the total sum of covariate distances between matches. `distmatch` is a wrapper to `bmatch`.

Usage

```
distmatch(t_ind, dist_mat = NULL, solver = NULL)
```

Arguments

`t_ind`	treatment indicator: a vector of zeros and ones indicating treatment (1 = treated; 0 = control). Please note that the data needs to be sorted in decreasing order according to this treatment indicator.
`dist_mat`	distance matrix: a matrix of positive distances between treated units (rows) and controls (columns). If `dist_mat = NULL` and `subset_weight = 1`, then bmatch will solve the cardinality matching problem in Zubizarreta et al. (2014).
`solver`	Optimization solver parameters: a list with four objects, `solver = list(name = name, t_max = t_max, approximate = 1, round_cplex = 0,` `trace_cplex = 0)`. `solver` is a string that determines the optimization solver to be used. The options are: `cplex`, `glpk`, `gurobi`, `highs`, and `symphony`. The default solver is `highs` with `approximate = 1`, so that by default an approximate solution is found (see `approximate` below). For an exact solution, we strongly recommend using `cplex` or `gurobi` as they are much faster than the other solvers, but they do require a license (free for academics, but not for people outside universities). Between `cplex` and `gurobi`, note that installing the R interface for `gurobi` is much simpler. `t_max` is a scalar with the maximum time limit for finding the matches. This option is specific to `cplex` and `gurobi`. If the optimal matches are not found within this time limit, a partial, suboptimal solution is given. `approximate` is a scalar that determines the method of solution. If `approximate = 1` (the default), an approximate solution is found via a relaxation of the original integer program. This method of solution is faster than `approximate = 0`, but some balancing constraints may be violated to some extent. This option works only with `n_controls = 1`, i.e. pair matching. `round_cplex` is binary specific to `cplex`. `round_cplex = 1` ensures that the solution found is integral by rounding and all the constraints are exactly satisfied; `round_cplex = 0` (the default) encodes there is no rounding which may return slightly infeasible integer solutions. `trace` is a binary specific to `cplex` and `gurobi`. `trace = 1` turns the optimizer output on. The default is `trace = 0`.

Value

A list containing the optimal solution, with the following objects:

`obj_total`	value of the objective function at the optimum;
`obj_dist_mat`	value of the total sum of distances term of the objective function at the optimum;
`t_id`	indexes of the matched treated units at the optimum;
`c_id`	indexes of the matched controls at the optimum;
`group_id`	matched pairs or groups at the optimum;
`time`	time elapsed to find the optimal solution.

Author(s)

Jose R. Zubizarreta, Cinar Kilcioglu.

References

Rosenbaum, P. R. (2010), *Design of Observational Studies*, Springer.

See Also

sensitivitymv, **sensitivitymw**.

Examples

```
# Load, sort, and attach data
data(lalonde)
lalonde = lalonde[order(lalonde$treatment, decreasing = TRUE), ]
attach(lalonde)

#################################
# Step 1: use cardinality matching to find the largest sample of matched pairs for which
# all the covariates are finely balanced.
#################################

# Discretize covariates
quantiles = function(covar, n_q) {
p_q = seq(0, 1, 1/n_q)
val_q = quantile(covar, probs = p_q, na.rm = TRUE)
covar_out = rep(NA, length(covar))
for (i in 1:n_q) {
if (i == 1) {covar_out[covar < val_q[i+1]] = i}
else if (i < n_q) {covar_out[covar >= val_q[i] & covar < val_q[i+1]] = i}
else {covar_out[covar >= val_q[i] & covar <= val_q[i+1]] = i}
}
covar_out
}
age_5 = quantiles(age, 5)
education_5 = quantiles(education, 5)
re74_5 = quantiles(re74, 5)
re75_5 = quantiles(re75, 5)

# Treatment indicator; note that the data needs to be sorted in decreasing order
# according to this treatment indicator
t_ind = treatment
t_ind

# Fine balance
fine_covs = cbind(black, hispanic, married, nodegree, age_5, education_5, re74_5, re75_5)
fine = list(covs = fine_covs)

# Solver options
t_max = 60*5
solver = "highs"
```

```
approximate = 0
solver = list(name = solver, t_max = t_max, approximate = approximate,
round_cplex = 0, trace = 0)

# Match
out_1 = cardmatch(t_ind, fine = fine, solver = solver)

# Indices of the treated units and matched controls
t_id_1 = out_1$t_id
c_id_1 = out_1$c_id

# Mean balance
covs = cbind(age, education, black, hispanic, married, nodegree, re74, re75)
meantab(covs, t_ind, t_id_1, c_id_1)

# Fine balance (note here we are getting an approximate solution)
for (i in 1:ncol(fine_covs)) {
print(finetab(fine_covs[, i], t_id_1, c_id_1))
}

#################################
# Step 2: use optimal matching (minimum distance matching) to find the (re)pairing of
# treated and control that minimizes the total sum of covariate distances between matched
# pairs.  For this, use the function 'distmatch' which is a wrapper for 'bmatch'.
#################################

# New treatment indicator
t_ind_2 = t_ind[c(t_id_1, c_id_1)]
table(t_ind_2)

# To build the distance matrix, the idea is to use strong predictors of the outcome
dist_mat_2 = abs(outer(re74[t_id_1], re74[c_id_1], "-"))
dim(dist_mat_2)

# Match
out_2 = distmatch(t_ind_2, dist_mat_2, solver)

# Indices of the treated units and matched controls
t_id_2 = t_id_1[out_2$t_id]
c_id_2 = c_id_1[out_2$c_id-length(out_2$c_id)]

# Covariate balance is preserved...
meantab(covs, t_ind, t_id_2, c_id_2)
for (i in 1:ncol(fine_covs)) {
print(finetab(fine_covs[, i], t_id_2, c_id_2))
}

# ... but covariate distances are reduced
distances_step_1 = sum(diag(dist_mat_2))
distances_step_2 = sum(diag(dist_mat_2[out_2$t_id, out_2$c_id-length(out_2$c_id)]))
distances_step_1
distances_step_2
```

```
# The mean difference in outcomes is the same...
mean(re78[t_id_1]-re78[c_id_1])
mean(re78[t_id_2]-re78[c_id_2])

# ... but their standard deviation is reduced
sd(re78[t_id_1]-re78[c_id_1])
sd(re78[t_id_2]-re78[c_id_2])
```

---

ecdfplot *Empirical cumulative distribution function plot for assessing covariate balance*

Description

Function that plots the empirical cumulative distribution function of a given covariate for treated units and matched controls. `ecdfplot` can be used to visually inspect the balance of the entire empirical distribution function of the covariate in question.

Usage

```
ecdfplot(x, t_id, c_id, main_title = "", legend_position = "right")
```

Arguments

`x`	a covariate vector to be used to assess balance.
`t_id`	a vector of indexes of the treated units.
`c_id`	a vector of indexes of the matched controls.
`main_title`	a string defining the main title of the plot.
`legend_position`	a string specifying the position of the legend. The default is `right`. Other options are: `topright`, `bottomright`, `bottom`, `bottomleft`, `left`, `topleft`, `top` and `center`

Details

Function that plots the empirical cumulative distribution function of a given covariate for treated units and matched controls. `ecdfplot` can be used to visually inspect the balance of the entire empirical distribution function of the covariate in question.

Author(s)

Jose R. Zubizarreta, Cinar Kilcioglu.

Examples

```
# Load data
data(germancities)

# Sort and attach data
germancities = germancities[order(germancities$treat, decreasing = TRUE), ]
attach(germancities)

# Treatment indicator
t_ind = treat

# Indexes of the treated units
t_id = which(t_ind == 1)

# Indexes of the controls before matching
c_id_before = which(t_ind == 0)

# Indixes of the matched controls (obtained using bmatch in designmatch)
c_id_after = c(80, 82, 35, 59, 69, 68, 34, 62, 104, 61, 106, 120, 56, 119, 28,
113, 76, 118, 75, 71)

# ecdfplot
par(mfrow = c(2, 1))
ecdfplot(rubble, t_id, c_id_before, "Before matching")
ecdfplot(rubble, t_id, c_id_after, "After matching")
```

---

finetab *Tabulate the marginal distribution of a nominal covariate after matching*

Description

Function for tabulating the marginal distributions of a nominal covariate for the treated units and matched controls.

Usage

```
finetab(nom_cov, t_id, c_id)
```

Arguments

nom_cov	a nominal covariate vector used to assess balance.
t_id	a vector of indexes of the treated units.
c_id	a vector of indexes of the matched controls.

Details

finetab is a function for tabulating the marginal distributions of a nominal covariate for the treated units and matched controls. finetab is useful for assessing covariate balance after matching with after exact, near-exact matching, fine and near-balance with the bmatch or nmatch functions in the designmatch package.

Value

A table with the counts for the treated units and matched controls for each category of a nominal covariate.

Author(s)

Jose R. Zubizarreta, Cinar Kilcioglu.

Examples

```
# Load data
data(germancities)

# Sort and attach data
germancities = germancities[order(germancities$treat, decreasing = TRUE), ]
attach(germancities)

# Treatment indicator
t_ind = treat

# Indexes of the treated units
t_id = which(t_ind == 1)

# Indixes of the matched controls (obtained using bmatch in designmatch)
c_id = c(80, 82, 35, 59, 69, 68, 34, 62, 104, 61, 106, 120, 56, 119, 28,
113, 76, 118, 75, 71)

  # finetab
finetab(publicat, t_id, c_id)
finetab(busiservcat, t_id, c_id)
```

---

germancities *Data from German cities before and after the Second World War*

Description

This is part of the data used by Redding and Sturm (2008) to study the impact of market access on economic development in West German cities after the division of Germany after the Second World War. There are 119 rows corresponding to different cities and 21 columns that stand for different variables. These variables are: one treatment indicator, 15 baseline covariates, and five outcomes. Treated cities are those West German cities within 75 kilometers of the border between East and West Germany after the Second World War (see Redding and Sturm (2008) for details). The complete dataset is available at doi:10.1257/aer.98.5.1766.

Usage

```
data(germancities)
```

Format

A data frame with 122 observations corresponding to 20 treated and 102 control cities. The treatment assignment indicator is the first column of the data frame: treat (1 = treated; 0 = control). The next 15 columns are the covariates:

- `log2pop`, logarithm base 2 of the population in each city in 1939;
- `popgrowth1939`, population growth in each city from 1919 to 1939;
- `popgrowth3339`, population growth in each city from 1919 to 1939;
- `emprate`, employment rates in each city in 1939;
- `indrate`, industry rates in each city in 1939;
- `rubble`, amount of rubble in cubic meters per capita in each city in 1939;
- `rubblemiss`, missing data indicator for rubble; the missing values were imputed with the mean;
- `flats`, number of destroyed dwellings in each city in 1939 as a percentage of the stock of dwelling;
- `flatsmiss`, missing data indicator for flats; the missing values were imputed with the mean;
- `refugees`, proportion of each city's population that identified themselves as refugees in 1939;
- `educat`, categories for the employment rates in the educational sector in each city in 1939;
- `publicat`, categories for the employment rates in the public administration sector in each city in 1939;
- `busiservcat`, categories for the employment rates in the business services sector in each city in 1939;
- `mineralcat`, categories for the employment rates in the minerals sector in each city in 1939;
- `transcat`, categories for the employment rates in the transport sector in each city in 1939.

The last five columns of the data frame are outcomes: `pop50`, `pop60`, `pop70`, `pop80` and `pop88`, the populations in each city in 1950, 1960, 1970, 1980, and 1988, respectively.

Source

doi:10.1257/aer.98.5.1766

References

Redding, S. J., and Daniel M. S. (2008), "The Costs of Remoteness: Evidence from German Division and Reunification," *American Economic Review*, 98, 1766-1797. doi:10.1257/aer.98.5.1766

lalonde *Lalonde data set*

Description

This is one of the data sets from the National Supported Work Demonstration used by Dehejia and Wahba (1999) to evaluate propensity score matching methods. This and other related data sets are available at `https://users.nber.org/~rdehejia/nswdata2.html`.

Usage

```
data(lalonde)
```

Format

A data frame with 445 observations, corresponding to 185 treated and 260 control subjects, and 10 variables. The treatment assignment indicator is the first variable of the data frame: `treatment` (1 = treated; 0 = control). The next 7 columns are the covariates:

- `age`, measured in years;
- `education`, measured in years;
- `black`, indicating race (1 if black, 0 otherwise);
- `hispanic`, indicating race (1 if Hispanic, 0 otherwise);
- `married`, indicating marital status (1 if married, 0 otherwise);
- `nodegree`, indicating high school diploma (1 if no degree, 0 otherwise);
- `re74`, real earnings in 1974;
- `re75`, real earnings in 1975.

The last variable of the data frame is `re78`, the real the earnings in 1978.

Source

`https://users.nber.org/~rdehejia/nswdata2.html`

References

Dehejia, R., and Wahba, S. (1999), "Causal Effects in Nonexperimental Studies: Reevaluating the Evaluation of Training Programs," *Journal of the American Statistical Association*, 94, 1053-1062.

Lalonde, R. (1986), "Evaluating the Econometric Evaluations of Training Programs," *American Economic Review*, 76, 604-620.

loveplot — *Love plot for assessing covariate balance*

Description

Function that creates a Love plot for assessing covariate balance after matching.

Usage

```
loveplot(X_mat, t_id, c_id, v_line, legend_position = "topright")
```

Arguments

`X_mat`	matrix of covariates: a matrix of covariates used to assess balance.
`t_id`	a vector of indexes of the treated units.
`c_id`	a vector of indexes of the matched controls.
`v_line`	a scalar defining the location of the vertical line that denotes a satisfactory balance.
`legend_position`	a string specifying the position of the legend. The default is `topright`. Other options are: `bottomright`, `bottom`, `bottomleft`, `left`, `topleft`, `top`, `topright`, `right` and `center`

Details

In the spirit of Love (2004), `loveplot` draws a love plot for assessing covariate balance after matching. Specifically, `loveplot` plots the absolute standardized differences in means before and after matching for all the covariates specified in `X_mat`.

Author(s)

Jose R. Zubizarreta, Cinar Kilcioglu.

References

Love, T. (2004), "Displaying Covariate Balance After Adjustment for Selection Bias," https://chrp.org/love/JSM_Aug11_TLove.pdf.

Examples

```
# Load data
data(germancities)

# Sort and attach data
germancities = germancities[order(germancities$treat, decreasing = TRUE), ]
attach(germancities)
```

```
# Treatment indicator
t_ind = treat

# Indexes of the treated units
t_id = which(t_ind == 1)

# Matrix of covariates
X_mat = cbind(log2pop, popgrowth1939, popgrowth3339, emprate, indrate,
rubble, rubblemiss, flats, flatsmiss, refugees)

# Indices of the matched controls (obtained using bmatch in designmatch)
c_id = c(67, 75, 39, 104, 38, 93, 79, 59, 64, 55, 106, 99, 97, 61, 82, 57, 76, 47, 46, 49)

# Vertical line for satisfactory balance
vline = 0.15

  # loveplot
loveplot(X_mat, t_id, c_id, vline)
```

---

meantab *Tabulate means of covariates after matching*

Description

Function for tabulating the means and other basic statistics useful to assess covariate balance after matching.

Usage

```
meantab(X_mat, t_ind, t_id, c_id, exact = NULL, digits = 2)
```

Arguments

X_mat	matrix of covariates: a matrix of covariates used to assess balance.
t_ind	treatment indicator: a vector of zeros and ones indicating treatment (1 = treated; 0 = control).
t_id	a vector of indexes of the treated units.
c_id	a vector of indexes of the matched controls.
exact	a vector of characters equal to "f" or "w" indicating whether Fisher's exact test or Wilcoxon rank-sum test should be used for binary (or categorical) and continuous covariates, respectively. Otherwise, if exact `exact = NULL`, simple t-tests are used. The default is `exact = NULL`. If `exact` is not `NULL`, the length of `exact` has to be equal to the number of columns of `X_mat`.
digits	a scalar indicating the number of digits to display in the columns of the table.

Details

`meantab` is a function for tabulating the means and other basic statistics useful to assess covariate balance after matching.

Value

A table with the following columns:

`Mis`	proportion of missing values for each covariate;
`Min`	minimum value for each covariate;
`Max`	maximum value for each covariate;
`Mean T`	mean of the treated units for each covariate;
`Mean C`	mean of the matched controls for each covariate;
`Std Dif`	standardized differences in means after matching for each covariate;
`P-val`	P-values for t-tests for differences in means between treated units and matched controls for each covariate.

See Also

cobalt, which supports **designmatch** objects

Examples

```
# Load data
data(germancities)

# Sort and attach data
germancities = germancities[order(germancities$treat, decreasing = TRUE), ]
attach(germancities)

# Treatment indicator
t_ind = treat

# Indexes of the treated units
t_id = which(t_ind == 1)

# Matrix of covariates
X_mat = cbind(log2pop, popgrowth1939, popgrowth3339, emprate, indrate, rubble,
rubblemiss, flats, flatsmiss, refugees)

# Indices of the matched controls (obtained using bmatch in designmatch)
c_id = c(67, 75, 39, 104, 38, 93, 79, 59, 64, 55, 106, 99, 97, 61, 82, 57,
76, 47, 46, 49)

  # meantab
meantab(X_mat, t_ind, t_id, c_id)

# meantab
meantab(X_mat, t_ind, t_id, c_id, exact = c(rep("w", 6), "f", "w", "f", "w"), digits = 3)
```

nmatch *Optimal nonbipartite matching in randomized experiments and observational studies*

Description

Function for optimal nonbipartite matching in randomized experiments and observational studies that directly balances the observed covariates. nmatch allows the user to enforce different forms of covariate balance in the matched samples, such as moment balance (e.g., of means, variances, and correlations), distributional balance (e.g., fine balance, near-fine balance, strength-*k* balancing), and exact matching. Among others, nmatch can be used in the design of randomized experiments for matching before randomization (Greevy et al. 2004, Zou and Zubizarreta 2016), and in observational studies for matching with doses and strengthening an instrumental variable (Baiocchi et al. 2010, Lu et al. 2011).

Usage

```
nmatch(dist_mat, subset_weight = NULL, total_pairs = NULL, mom = NULL,
       exact = NULL, near_exact = NULL, fine = NULL, near_fine = NULL,
       near = NULL, far = NULL, solver = NULL)
```

Arguments

dist_mat	distance matrix: a matrix of positive distances between units.
subset_weight	subset matching weight: a scalar that regulates the trade-off between the total sum of distances between matched pairs and the total number of matched pairs. The larger subset_weight, the more importance will be given to the the total number of matched pairs relative to the total sum of distances between matched pairs. See Rosenbaum (2012) and Zubizarreta et al. (2013) for a discussion of this parameter. If subset_weight = NULL, then nmatch will match all the available units, provided it exists a feasible solution exists.
total_pairs	total number of matched pairs: a scalar specifying the number of matched pairs to be obtained. If total_pairs = NULL then no specific number of matched pairs is required before matching.
mom	moment balance parameters: a list with three arguments, mom = list(covs = mom_covs, tols = mom_tols, targets = mom_targets). mom_covs is a matrix where each column is a covariate whose mean is to be balanced. mom_tols is a vector of tolerances for the maximum difference in means for the covariates in mom_covs. mom_targets is a vector of target moments (e.g., means) of a distribution to be approximated by matching. mom_targets is optional, but if mom_covs is specified then mom_tols needs to be specified too. If mom_targets is NULL, then nmatch will match treated and control units so that covariates in mom_covs differ at most by mom_tols. If mom_targets is specified, then nmatch will match treated and control units so that each matched group differs at most by mom_tols units from the respective moments in mom_targets. As a result, the matched groups will differ at most mom_tols * 2 from each

	other. Under certain assumptions, `mom_targets` can be used for constructing a representative matched sample. The lengths of `mom_tols` and `mom_target` have to be equal to the number of columns of `mom_covs`. Note that the columns of `mom_covs` can be transformations of the original covariates to balance higher order single-dimensional moments like variances and skewness, and multidimensional moments such as correlations (Zubizarreta 2012).
`exact`	Exact matching parameters: a list with one argument, `exact = list(covs = exact_covs)`, where `exact_covs` is a matrix where each column is a nominal covariate for exact matching.
`near_exact`	Near-exact matching parameters: a list with two arguments, `near_exact = list(covs = near_exact_covs, devs = near_exact_devs)`. `near_exact_covs` are the near-exact matching covariates; specifically, a matrix where each column is a nominal covariate for near-exact matching. `near_exact_devs` are the maximum deviations from near-exact matching: a vector of scalars defining the maximum deviation allowed from exact matching for the covariates defined in `near_exact_covs`. Note that the length of `near_exact_devs` has to be equal to the number of columns of `near_exact_covs`. For detailed expositions of near-exact matching in the context of bipartite matching, see section 9.2 of Rosenbaum (2010) and Zubizarreta et al. (2011).
`fine`	Fine balance parameters: a list with one argument, `fine = list(covs = fine_covs)`, where `fine_covs` is a matrix where each column is a nominal covariate for fine balance. Fine balance enforces exact distributional balance on nominal covariates, but without constraining treated and control units to be matched within each category of each nominal covariate as in exact matching. See chapter 10 of Rosenbaum (2010) for details.
`near_fine`	Near-fine balance parameters: a list with two arguments, `near_fine = list(covs = near_fine_covs, devs = near_fine_devs)`. `near_fine_covs` is a matrix where each column is a nominal covariate for near-fine matching. `near_fine_devs` is a vector of scalars defining the maximum deviation allowed from fine balance for the covariates in `near_fine_covs`. Note that the length of `near_fine_devs` has to be equal to the number of columns of `near_fine_covs`. See Yang et al. (2012) for a description of near-fine balance.
`near`	Near matching parameters: a list with three arguments, `near = list(covs = near_covs, pairs = near_pairs, groups = near_groups)`. `near_covs` is a matrix where each column is a variable for near matching. `near_pairs` is a vector determining the maximum distance between individual matched pairs for each variable in `near_covs`. `near_groups` is a vector defining the maximum average distance (in aggregate) between matched groups for each covariate in `near_covs`. If `near_covs` is specified, then either `near_pairs`, `near_covs`, or both must be specified as well, and the length of `near_pairs` and/or `near_groups` has to be equal to the number of columns of `near_covs`.
`far`	Far matching parameters: a list with three arguments, `far = list(covs = far_covs, pairs = far_pairs, groups = far_groups)`.

	`far_covs` is a matrix where each column is a variable (a covariate or an instrumental variable) for far matching. `far_pairs` is a vector determining the minimum distance between units in a matched pair for each variable in `far_covs`, and `far_groups` is a vector defining the minimum average (aggregate) distance between matched groups for each variable in `far_covs`. If `far_covs` is specified, then either `far_pairs`, `far_covs`, or both, must be specified, and the length of `far_pairs` and/or `far_groups` has to be equal to the number of columns of `far_covs`. See Zubizarreta et al. (2013) for strengthening an instrumental variable with integer programming.
`solver`	Optimization solver parameters: a list with four objects, `solver = list(name = name, t_max = t_max, approximate = 1, round_cplex = 0,` `trace_cplex = 0)`. `solver` is a string that determines the optimization solver to be used. The options are: `cplex`, `glpk`, `gurobi`, `highs`, and `symphony`. The default solver is `highs` with `approximate = 1`, so that by default an approximate solution is found (see `approximate` below). For an exact solution, we strongly recommend using `cplex` or `gurobi` as they are much faster than the other solvers, but they do require a license (free for academics, but not for people outside universities). Between `cplex` and `gurobi`, note that the installation of the `gurobi` interface for R is much simpler. `t_max` is a scalar with the maximum time limit for finding the matches. This option is specific to `cplex` and `gurobi`. If the optimal matches are not found within this time limit, a partial, suboptimal solution is given. `approximate` is a scalar that determines the method of solution. If `approximate = 1` (the default), an approximate solution is found via a relaxation of the original integer program. This method of solution is faster than `approximate = 0`, but some balancing constraints may be violated to some extent. `round_cplex` is binary specific to `cplex`. `round_cplex = 1` ensures that the solution found is integral by rounding and all the constraints are exactly satisfied; `round_cplex = 0` (the default) encodes there is no rounding which may return slightly infeasible integer solutions. `trace` is a binary specific to `cplex` and `gurobi`. `trace = 1` turns the optimizer output on. The default is `trace = 0`.

Value

A list containing the optimal solution, with the following objects:

`obj_total`	value of the objective function at the optimum;
`obj_dist_mat`	value of the total sum of distances term of the objective function at the optimum;
`id_1`	indexes of the matched units in group 1 at the optimum;
`id_2`	indexes of the matched units in group 2 at the optimum;
`group_id`	matched pairs at the optimum;
`time`	time elapsed to find the optimal solution.

Author(s)

Jose R. Zubizarreta, Cinar Kilcioglu.

References

Baiocchi, M., Small, D., Lorch, S. and Rosenbaum, P. R. (2010), "Building a Stronger Instrument in an Observational Study of Perinatal Care for Premature Infants," *Journal of the American Statistical Association*, 105, 1285-1296.

Greevy, R., Lu, B., Silber, J. H., and Rosenbaum, P. R. (2004), "Optimal Multivariate Matching Before Randomization," *Biostatistics*, 5, 263-275.

Lu, B., Greevy, R., Xu, X., and Beck C. (2011), "Optimal Nonbipartite Matching and its Statistical Applications," *The American Statistician*, 65, 21-30.

Rosenbaum, P. R. (2010), *Design of Observational Studies*, Springer.

Rosenbaum, P. R. (2012), "Optimal Matching of an Optimally Chosen Subset in Observational studies," *Journal of Computational and Graphical Statistics*, 21, 57-71.

Yang. F., Zubizarreta, J. R., Small, D. S., Lorch, S. A., and Rosenbaum, P. R. (2014), "Dissonant Conclusions When Testing the Validity of an Instrumental Variable," *The American Statistician*, 68, 253-263.

Zou, J., and Zubizarreta, J. R. (2016), "Covariate Balanced Restricted Randomization: Optimal Designs, Exact Tests, and Asymptotic Results," working paper.

Zubizarreta, J. R., Reinke, C. E., Kelz, R. R., Silber, J. H., and Rosenbaum, P. R. (2011), "Matching for Several Sparse Nominal Variables in a Case-Control Study of Readmission Following Surgery," *The American Statistician*, 65, 229-238.

Zubizarreta, J. R. (2012), "Using Mixed Integer Programming for Matching in an Observational Study of Kidney Failure after Surgery," *Journal of the American Statistical Association*, 107, 1360-1371.

Examples

```
## Uncomment the following example
## Load and attach data
#data(lalonde)
#attach(lalonde)

#################################
## Example: optimal subset matching
#################################

## Optimal subset matching pursues two competing goals at
## the same time: to minimize the total of distances while
## matching as many observations as possible.  The trade-off
## between these two is regulated by the parameter subset_weight
## (see Rosenbaum 2012 and Zubizarreta et al. 2013 for a discussion).
## Here the balance requirements are mean and fine balance for
## different covariates.  We require 50 pairs to be matched.
## Again, the solver used is HiGHS with the approximate option.
## Matrix of covariates
#X_mat = cbind(age, education, black, hispanic, married, nodegree, re74, re75)

## Distance matrix
#dist_mat_covs = round(dist(X_mat, diag = TRUE, upper = TRUE), 1)
#dist_mat = as.matrix(dist_mat_covs)

## Subset matching weight
#subset_weight = 1

## Total pairs to be matched
#total_pairs = 50

## Moment balance: constrain differences in means to be at most .1 standard deviations apart
#mom_covs = cbind(age, education)
#mom_tols = apply(mom_covs, 2, sd)*.1
#mom = list(covs = mom_covs, tols = mom_tols)

## Solver options
#t_max = 60*5
#solver = "highs"
#approximate = 1
#solver = list(name = solver, t_max = t_max, approximate = approximate, round_cplex = 0,
#trace_cplex = 0)

## Match
#out = nmatch(dist_mat = dist_mat, subset_weight = subset_weight, total_pairs = total_pairs,
#mom = mom, solver = solver)

## Indices of the treated units and matched controls
#id_1 = out$id_1
#id_2 = out$id_2

## Assess mean balance
#a = apply(mom_covs[id_1, ], 2, mean)
#b = apply(mom_covs[id_2, ], 2, mean)
#tab = round(cbind(a, b, a-b, mom_tols), 2)
#colnames(tab) = c("Mean 1", "Mean 2", "Diffs", "Tols")
#tab

## Assess fine balance (note here we are getting an approximate solution)
#for (i in 1:ncol(fine_covs)) {
# print(finetab(fine_covs[, i], id_1, id_2))
#}
```

`pairsplot` *Pairs plot for visualizing matched pairs*

Description

Function for visualizing matched pairs in two dimensions.

Usage

```
pairsplot(cov1, cov2, t_id, c_id, xlab, ylab, main)
```

Arguments

`cov1`	a vector for the covariate to be plotted on the x axis.
`cov2`	a vector for the covariate to be plotted on the y axis.
`t_id`	a vector of indexes of the treated units.
`c_id`	a vector of indexes of the matched controls.
`xlab`	a string specifying the label of the x axis.
`ylab`	a string specifying the label of the y axis.
`main`	a string specifying the main title of the plot.

Details

`pairsplot` is a function for visualizing matched pairs in two dimensions, usually defined by two of the matching covariates. Matched pairs are connected by line segments. Horizontal and vertical lines show the means of treated units and matched controls for each of the covariates. Among others, `pairsplot` can be useful for visualizing near/far matches, e.g. when building a stronger instrumental variable (Baiocchi et al., 2010).

Author(s)

Jose R. Zubizarreta, Cinar Kilcioglu.

References

Baiocchi, M., Small, D., Lorch, S. and Rosenbaum, P. R. (2010), "Building a Stronger Instrument in an Observational Study of Perinatal Care for Premature Infants," *Journal of the American Statistical Association*, 105, 1285-1296.

Examples

```
# Load data
data(germancities)

# Sort and attach data
germancities = germancities[order(germancities$treat, decreasing = TRUE), ]
attach(germancities)

# Treatment indicator
t_ind = treat

# Indexes of the treated units
t_id = which(t_ind == 1)

# Indices of the matched controls (obtained using bmatch in designmatch)
c_id = c(67, 75, 39, 104, 38, 93, 79, 59, 64, 55, 106, 99, 97, 61, 82,
57, 76, 47, 46, 49)

# pairsplot
pairsplot(rubble, flats, t_id, c_id, "Rubble", "Flats", "")
```

profmatch *Optimal profile matching*

Description

Function for optimal profile matching to construct matched samples that are balanced toward a user-specified covariate profile. This covariate profile can represent a specific population or a target individual, facilitating the generalization and personalization of causal inferences (Cohn and Zubizarreta 2022). For each treatment group reservoir, `profmatch` finds the largest sample that is balanced relative to the profile. The formulation of `profmatch` has been simplified to handle larger data than `bmatch` or `nmatch`. Similar to `bmatch` or `nmatch`, the performance of `profmatch` is greatly enhanced by using the `solver` options `cplex` or `gurobi`.

Usage

```
profmatch(t_ind, mom, n_units = NULL, solver = NULL)
```

Arguments

`t_ind`	treatment indicator: a vector indicating treatment group of each observation.
`mom`	moment balance parameters: a list with three arguments, `mom = list(targets = mom_targets, covs = mom_covs, tols = mom_tols)`. `mom_targets` is the profile, i.e. a vector of target moments (e.g., means) of a distribution or a vector of characteristics of an individual toward which to balance the treatment groups. `mom_covs` is a matrix where each column is a covariate whose mean is to be balanced toward `mom_targets`. `mom_tols` is a vector of tolerances for the maximum difference in means between the covariates in `mom_covs` and the elements of `mom_targets`. `profmatch` will select units from each treatment group so that each matched group differs at most by `mom_tols` from the respective elements of `mom_targets`. As a result, the matched groups will differ at most `mom_tols * 2` from each other. Under certain assumptions, `mom_targets` can be used for constructing a representative matched sample. The lengths of `mom_tols` and `mom_target` have to be equal to the number of columns of `mom_covs`. Note that the columns of `mom_covs` can be transformations of the original covariates to balance higher order single-dimensional moments like variances and skewness, and multidimensional moments such as correlations (Zubizarreta 2012).
`n_units`	an optional vector with length equal to the number of treatment groups containing the desired size of each matched sample.

`solver`	Optimization solver parameters: a list with four objects, `solver = list(name = name, t_max = t_max, approximate = 1, round_cplex = 0,` `trace_cplex = 0)`. `solver` is a string that determines the optimization solver to be used. The options are: `cplex`, `glpk`, `gurobi`, `highs`, and `symphony`. The default solver is `highs` with `approximate = 1`. For an exact solution, we strongly recommend using `cplex` or `gurobi` as they are much faster than the other solvers, but they do require a license (free for academics, but not for people outside universities). Between `cplex` and `gurobi`, note that installing the R interface for `gurobi` is much simpler. `t_max` is a scalar with the maximum time limit for finding the matches. This option is specific to `cplex`, `gurobi`, and `highs`. If the optimal matches are not found within this time limit, a partial, suboptimal solution is given. `approximate` is a scalar that determines the method of solution. If `approximate = 1` (the default), an approximate solution is found via a relaxation of the original integer program. This method of solution is faster than `approximate = 0`, but some balancing constraints may be violated to some extent. This option works only with `n_controls = 1`, i.e. pair matching. `round_cplex` is binary specific to `cplex`. `round_cplex = 1` ensures that the solution found is integral by rounding and all the constraints are exactly satisfied; `round_cplex = 0` (the default) encodes there is no rounding which may return slightly infeasible integer solutions. `trace` is a binary specific to `cplex` and `gurobi`. `trace = 1` turns the optimizer output on. The default is `trace = 0`.

Value

A list containing the optimal solution, with the following objects:

`obj_totals`	values of the objective functions at the optima (one value for each treatment group matching problem);
`ids`	indices of the matched units at the optima;
`times`	time elapsed to find the optimal solutions (one value for each treatment group matching problem).

Author(s)

Eric R. Cohn, Jose R. Zubizarreta, Cinar Kilcioglu, Juan P. Vielma.

References

Zubizarreta, J. R. (2012), "Using Mixed Integer Programming for Matching in an Observational Study of Kidney Failure after Surgery," *Journal of the American Statistical Association*, 107, 1360-1371.

Cohn, E. R., & Zubizarreta, J. R. (2022). "Profile Matching for the Generalization and Personalization of Causal Inferences". *Epidemiology*, 33(5), 678. doi:10.1097/EDE.0000000000001517

See Also

sensitivitymv, **sensitivitymw**.

Examples

```
### Load, sort, and attach data
#data(lalonde)
#lalonde = lalonde[order(lalonde$treatment, decreasing = TRUE), ]
#attach(lalonde)

### Specify covariates
#covs = c("age", "education", "black", "hispanic", "married", "nodegree", "re74", "re75")

### Vector of treatment group indicators
#t_ind = lalonde$treatment

### Covariate matrix
#mom_covs = as.matrix(lalonde[, covs])

### Tolerances will be 0.05 * each covariate's standard deviation
#mom_sds = apply(lalonde[, covs], 2, sd)
#mom_tols = 0.05 * mom_sds

### Target moments will be the overall means in the sample
#mom_targets = colMeans(lalonde[, covs])

### Solver options
#t_max = 60*30
#solver = "gurobi"
#approximate = 0
#solver = list(name = solver, t_max = t_max, approximate = approximate, round_cplex = 0, trace = 0)

#mom = list(covs = mom_covs, tols = mom_tols, targets = mom_targets)
#pmatch_out = profmatch(t_ind, mom, solver)

### Selecting the matched units
#lalonde.matched = lalonde[pmatch_out$id,]

### Comparing TASMDs before and after matching
#TASMD.0.2 = abs(colMeans(lalonde.matched[which(lalonde.matched$treatment == 0), covs])
#                    - mom_targets) / mom_sds
#TASMD.1.2 = abs(colMeans(lalonde.matched[which(lalonde.matched$treatment == 1), covs])
#                    - mom_targets) / mom_sds

#TASMD.0.1 = abs(colMeans(lalonde[which(lalonde$treatment == 0), covs]) - mom_targets) / mom_sds
#TASMD.1.1 = abs(colMeans(lalonde[which(lalonde$treatment == 1), covs]) - mom_targets) / mom_sds

### For each treatment group, ASAMDs are reduced after matching (i.e., balance is achieved)
#cbind(TASMD.0.1, TASMD.0.2)
#cbind(TASMD.1.1, TASMD.1.2)
```

Index