Package ‘quickmatch’

July 22, 2025

Type Package

Title Quick Generalized Full Matching

Version 0.2.3

Date 2024-07-31

Description Provides functions for constructing near-optimal generalized full matching. Generalized full matching is an extension of the original full matching method to situations with more intricate study designs. The package is made with large data sets in mind and derives matches more than an order of magnitude quicker than other methods.

Depends R (>= 3.4.0), distances

Imports sandwich, scclust (>= 0.2.0), stats

Suggests testthat

NeedsCompilation yes

License GPL (>= 3)

URL `https://github.com/fsavje/quickmatch`

BugReports `https://github.com/fsavje/quickmatch/issues`

Encoding UTF-8

RoxygenNote 7.2.3

Author Fredrik Savje [aut, cre],
Jasjeet Sekhon [aut],
Michael Higgins [aut]

Maintainer Fredrik Savje `<rpackages@fredriksavje.com>`

Repository CRAN

Date/Publication 2024-07-31 22:30:18 UTC

Contents

`quickmatch-package` *quickmatch: Quick Generalized Full Matching*

Description

Provides functions for constructing near-optimal generalized full matchings. Generalized full matching is an extension of the original full matching method to situations with more intricate study designs. The package is made with large data sets in mind and derives matchings more than an order of magnitude quicker than other methods.

Details

See `quickmatch` for the main matching function.

See the package's website for more information: `https://github.com/fsavje/quickmatch`.

Bug reports and suggestions are greatly appreciated. They are best reported here: `https://github.com/fsavje/quickmatch/issues`.

References

Sävje, Fredrik and Michael J. Higgins and Jasjeet S. Sekhon (2017), 'Generalized Full Matching', arXiv 1703.03882. `https://arxiv.org/abs/1703.03882`

`covariate_averages` *Covariate averages in matched sample*

Description

`covariate_averages` derives covariate averages for treatment groups in matched samples.

Usage

```
covariate_averages(treatments, covariates, matching = NULL, target = NULL)
```

Arguments

`treatments`	factor specifying the units' treatment assignments.
`covariates`	vector, matrix or data frame with covariates to calculate averages for.
`matching`	`qm_matching` or `scclust` object with the matched groups. If `NULL`, averages are derived for the unmatched sample.
`target`	units to target the averages for. If `NULL`, the averages will be the raw average over all units in the sample (i.e., ATE). A non-null value specifies a subset of units to derive averages for (e.g., ATT or ATC). If `target` is a logical vector with the same length as the sample size, units indicated with `TRUE` will be targeted. If `target` is an integer vector, the units with indices in `target` are targeted. If `target` is a character vector, it should contain treatment labels, and the corresponding units (as given by `treatments`) will be targeted. If `matching` is `NULL`, `target` is ignored.

Details

`covariate_averages` calculates covariate averages by first deriving the average for each covariate for each treatment conditions in each matched group. It then aggregates the group averages by a weighted average, where the `target` parameter decides the weights. If a matched group contains many units not targeted (e.g., control units when ATT is of interest), those units will contribute less to the covariate average for the corresponding treatment condition than units in matched groups with many targeted units. This means that the covariate average is calculated in the same way as the potential outcomes are estimated. In fact, `covariate_averages` can be used as an estimator for potential outcomes by calling it with the outcome variable as a covariate.

When the average treatment effect (ATE) is of interest (i.e., `target` == `NULL`), the matched groups will be weighted by their sizes. When `target` indicates that some subset of units is of interest, the number of such units in each matched group will decide its weight. For example, if we are interested in the average treatment effect of the treated (ATT), the weight of a group will be proportional to the number of treated units in that group.

In practice, the function first derives the unit-level weights implied by the matching. In detail, let $S(g)$ be the number of units indicated by `target` in group g. Let T be the total number of units indicated by `target` in the sample. Let $A(t, g)$ be the number of units assigned to treatment t in group g. The weight for a unit in group g that is assigned to treatment t is given by:

$$\frac{S(g)}{T \times A(t, g)}.$$

See `matching_weights` for more details.

`covariate_averages` focuses on means, but higher moments and interactions can be investigated by adding corresponding columns to the covariate matrix (see examples below).

Value

Returns a matrix with the average of each covariate for each treatment group. The rows in the matrix correspond to the covariates in order and the columns correspond to the treatment groups. For example, a possible output with three treatment groups ("C", "T1" and "T2") and four covariates is:

C	T1	T2
3.0	3.3	3.5
-0.3	0.0	-0.2
0.1	0.2	0.0
5.0	5.1	4.9

which indicates that the average of the first covariate in the matched sample is 3.0 for units assigned to condition "C", and that the average of the third covariate is 0.2 for units assigned to condition "T1".

Examples

```
# Construct example data
my_data <- data.frame(y = rnorm(100),
                      x1 = runif(100),
                      x2 = runif(100),
                     treatment = factor(sample(rep(c("T1", "T2", "C"), c(25, 25, 50)))))

# Make distances
my_distances <- distances(my_data, dist_variables = c("x1", "x2"))

# Treatment group averages in unmatched sample
covariate_averages(my_data$treatment, my_data[c("x1", "x2")])

# Make matching
my_matching <- quickmatch(my_distances, my_data$treatment)

# Treatment group averages in matched sample
covariate_averages(my_data$treatment, my_data[c("x1", "x2")], my_matching)

# Averages in matched sample for ATT
covariate_averages(my_data$treatment,
                   my_data[c("x1", "x2")],
                   my_matching,
                   target = c("T1", "T2"))

# Second-order moments and interactions
mod_covs <- data.frame(x1 = my_data$x1,
                       x2 = my_data$x2,
                       x1sq = my_data$x1^2,
                       x2sq = my_data$x2^2,
                       x1x2 = my_data$x1 * my_data$x2)
covariate_averages(my_data$treatment, mod_covs, my_matching)
```

`covariate_balance` *Covariate balance in matched sample*

Description

`covariate_balance` derives measures of covariate balance between treatment groups in matched samples. The function calculates normalized mean differences between all pairs of treatment conditions for each covariate.

Usage

```
covariate_balance(
  treatments,
  covariates,
  matching = NULL,
  target = NULL,
  normalize = TRUE,
  all_differences = FALSE
)
```

Arguments

`treatments`	factor specifying the units' treatment assignments.
`covariates`	vector, matrix or data frame with covariates to derive balance for.
`matching`	`qm_matching` or `scclust` object with the matched groups. If `NULL`, balance is derived for the unmatched sample.
`target`	units to target the balance measures for. If `NULL`, the measures will be the raw average over all units in the sample (i.e., ATE). A non-null value specifies a subset of units to derive balance measures for (e.g., ATT or ATC). If `target` is a logical vector with the same length as the sample size, units indicated with `TRUE` will be targeted. If `target` is an integer vector, the units with indices in `target` are targeted. If `target` is a character vector, it should contain treatment labels, and the corresponding units (as given by `treatments`) will be targeted. If `matching` is `NULL`, `target` is ignored.
`normalize`	logical scalar indicating whether differences should be normalized by the sample standard deviation of the corresponding covariates.
`all_differences`	logical scalar indicating whether full matrices of differences should be reported. If `FALSE`, only the maximum difference for each covariate is returned.

Details

`covariate_balance` calculates covariate balance by first deriving the (normalized) mean difference between all treatment conditions for each covariate in each matched group. It then aggregates the differences by a weighted average, where the `target` parameter decides the weights. When the average treatment effect (ATE) is of interest (i.e., `target == NULL`), the matched groups will be weighted by their sizes. When `target` indicates that some subset of units is of interest, the number of such units in each matched group will decide its weight. For example, if we are interested in the average treatment effect of the treated (ATT), the weight of a group will be proportional to the number of treated units in that group. The reweighting of the groups captures that we are prepared to accept greater imbalances in groups with few units of interest.

By default, the differences are normalized by the sample standard deviation of the corresponding covariate (see the `normalize` parameter). In more detail, the sample variance of the covariate is derived separately for each treatment group. The square root of the mean of these variances is then used for the normalization. The matching is ignored when deriving the normalization factor so that balance can be compared across different matchings or with the unmatched sample.

`covariate_balance` focuses on mean differences, but higher moments and interactions can be investigated by adding corresponding columns to the covariate matrix (see examples below).

Value

Returns the mean difference between treatment groups in the matched sample for each covariate.

When `all_differences = TRUE`, the function returns a matrix for each covariate with the mean difference for each possible pair of treatment conditions. Rows in the matrices indicate minuends in the differences and columns indicate subtrahends. For example, when differences are normalized, the matrix:

	a	b	c
a	0.0	0.3	0.5
b	-0.3	0.0	0.2
c	-0.5	-0.2	0.0

reports that the mean difference for the corresponding covariate between treatments "a" and "b" is 30% of a sample standard deviation of the covariate. The maximum difference (in absolute value) is also reported in a separate vector. For example, the maximum difference for the covariate in the example above is 0.5.

When `all_differences = FALSE`, only the maximum differences are reported.

Examples

```
# Construct example data
my_data <- data.frame(y = rnorm(100),
                      x1 = runif(100),
                      x2 = runif(100),
                     treatment = factor(sample(rep(c("T1", "T2", "C"), c(25, 25, 50)))))

# Make distances
my_distances <- distances(my_data, dist_variables = c("x1", "x2"))

# Balance in unmatched sample (maximum for each covariate)
covariate_balance(my_data$treatment, my_data[c("x1", "x2")])

# Make matching
my_matching <- quickmatch(my_distances, my_data$treatment)

# Balance in matched sample (maximum for each covariate)
covariate_balance(my_data$treatment, my_data[c("x1", "x2")], my_matching)

# Balance in matched sample for ATT
```

```
covariate_balance(my_data$treatment,
                  my_data[c("x1", "x2")],
                  my_matching,
                  target = c("T1", "T2"))

# Balance on second-order moments and interactions
balance_cov <- data.frame(x1 = my_data$x1,
                          x2 = my_data$x2,
                          x1sq = my_data$x1^2,
                          x2sq = my_data$x2^2,
                          x1x2 = my_data$x1 * my_data$x2)
covariate_balance(my_data$treatment, balance_cov, my_matching)

# Report all differences (not only maximum for each covariate)
covariate_balance(my_data$treatment,
                  my_data[c("x1", "x2")],
                  my_matching,
                  all_differences = TRUE)
```

is.qm_matching *Check qm_matching object*

Description

is.qm_matching checks whether the provided object is a valid instance of the qm_matching class.

Usage

```
is.qm_matching(x)
```

Arguments

x	object to check.

Details

is.qm_matching does not check whether the matching itself is sensible or whether it satisfies some set of constraints. See check_clustering for that functionality.

Value

Returns TRUE if x is a valid qm_matching object, otherwise FALSE.

lm_match	*Regression-based matching estimator of treatment effects*

Description

lm_match estimates treatment effects in matched samples. The function expects the user to provide the outcomes, treatment indicators, and a matching object. It returns point estimates of the average treatment effects and variance estimates. It is possible to estimate treatment effects for subsets of the observations, such as estimates of the average treatment effect for the treated (ATT).

Usage

```
lm_match(outcomes, treatments, matching, covariates = NULL, target = NULL)
```

Arguments

outcomes	numeric vector with observed outcomes.
treatments	factor specifying the units' treatment assignments.
matching	qm_matching or scclust object with the matched groups.
covariates	vector, matrix or data frame with covariates to include in the estimation. If NULL, no covariates are included.
target	units to target the estimation for. If NULL, the effect is estimated for all units in the sample (i.e., ATE). A non-null value specifies a subset of units for which the effect should be estimated (e.g., ATT or ATC). If target is a logical vector with the same length as the sample size, units indicated with TRUE will be targeted. If target is an integer vector, the units with indices in target are targeted. If target is a character vector, it should contain treatment labels, and the effect for the corresponding units (as given by treatments) will be estimated.

Details

lm_match estimates treatment effects using weighted regression. The function first derives the unit-level weights implied by the matching. In detail, let $S(g)$ be the number of units indicated by target in group g. Let T be the total number of units indicated by target in the sample. Let $A(t, g)$ be the number of units assigned to treatment t in group g. The weight for a unit in group g that is assigned to treatment t is given by:

$$\frac{S(g)}{T \times A(t, g)}.$$

See matching_weights for more details.

The function uses the derived weights in a weighted least squares regression (using the lm function) with indicator variables for the treatment conditions. Optionally, covariates can be added to the regression (e.g., a common recommendation is to include the covariates used to construct the matching). Standard errors are estimated with the heteroskedasticity-robust "HC1" estimator in the vcovHC function. Units not assigned to matched groups and units assigned weights of zero are excluded from the estimation.

Value

A list with two numeric matrices with all estimated treatment effects and their estimated variances is returned. The first matrix (`effects`) contains estimated treatment effects. Rows in this matrix indicate minuends in the treatment effect contrast and columns indicate subtrahends. For example, in the matrix:

	a	b	c
a	0.0	4.5	5.5
b	-4.5	0.0	1.0
c	-5.5	-1.0	0.0

the estimated treatment effect between conditions a and b is 4.5, and the estimated treatment effect between conditions c and b is -1.0. In symbols, $E[Y(a) - Y(b)|S] = 4.5$ and $E[Y(c) - Y(b)|S] = -1.0$ where S is the condition set indicated by the `target` parameter.

The second matrix (`effect_variances`) contains estimates of variances of the corresponding effect estimators.

References

Stuart, Elizabeth A. (2010), 'Matching Methods for Causal Inference: A Review and a Look Forward'. Statistical Science, 25(1), 1–21. https://doi.org/10.1214/09-STS313

Examples

```
# Construct example data
my_data <- data.frame(y = rnorm(100),
                      x1 = runif(100),
                      x2 = runif(100),
                     treatment = factor(sample(rep(c("T1", "T2", "C"), c(25, 25, 50)))))

# Make distances
my_distances <- distances(my_data, dist_variables = c("x1", "x2"))

# Make matching
my_matching <- quickmatch(my_distances, my_data$treatment)

# ATE without covariates
lm_match(my_data$y,
         my_data$treatment,
         my_matching)

# ATE with covariates
lm_match(my_data$y,
         my_data$treatment,
         my_matching,
         my_data[c("x1", "x2")])

# ATT for T1
lm_match(my_data$y,
         my_data$treatment,
         my_matching,
         my_data[c("x1", "x2")],
         target = "T1")
```

`matching_weights` *Unit weights implied by matching*

Description

`matching_weights` derives the weights implied by a matching for units assigned to matched groups. If the matching is exact, reweighting the units with this function will produce a sample where all treatment groups are identical on the matching covariates. If the matching is approximate but of good quality, the reweighted treatment groups will be close to identical.

Usage

```
matching_weights(treatments, matching, target = NULL)
```

Arguments

`treatments`	factor specifying the units' treatment assignments.
`matching`	`qm_matching` or `scclust` object with the matched groups.
`target`	units to target the weights for. If `NULL`, the weights will target all units, so that the reweighted treatment groups are as similar as possible to the complete sample (i.e., corresponding to ATE). A non-null value specifies a subset of units that the weights should be targeted for (e.g., ATT or ATC). If `target` is a logical vector with the same length as the sample size, units indicated with `TRUE` will be targeted. If `target` is an integer vector, the units with indices in `target` are targeted. If `target` is a character vector, it should contain treatment labels, and the weights target the corresponding units (as given by `treatments`).

Details

Let $S(g)$ be the number of units indicated by `target` in group g (or the total number of units in the group if `target` is `NULL`). Let T be the total number of units indicated by `target` in the sample (or the sample size if `target` is `NULL`). Let $A(t, g)$ be the number of units assigned to treatment t in group g. The weight for a unit in group g that is assigned to treatment t is given by:

$$\frac{S(g)}{T \times A(t, g)}.$$

Consider, for example, a matched group with one treated unit and two control units when we are interested in the average effect of the treated (ATT) and we have 50 treated units in total ($T = 50$). For all three units in the group, we have $S(g) = 1$. For the treated unit we have $A(t, g) = 1$, so its weight becomes $1/50$. The two control units have $A(t, g) = 2$, so their weights are both $1/100$.

These weights are such that the difference between the weighted averages of the outcomes in two treatment conditions is the same as the average within-group difference-in-means between the two conditions.

If a matched group g with $S(g) > 0$ lacks some treatment condition t, no weights exist for the units assigned to t (in other groups) so to replicate group g; the matching does not contain enough information to impute the missing treatment in group g. Subsequently, all units assigned to t will be given the weight `NA`. There is two ways to solve this problem. First, one can change the target estimand by setting $S(g) = 0$ for all groups that are missing units assigned to t. This is done with the `target` parameter. Second, one can change the matching so that all groups contain at least one unit assigned to t (e.g., by merging groups).

Units not assigned to matched groups are given zero weights.

Value

Returns a numeric vector with the weights of the units in the matching.

Examples

```
# Construct example data
my_data <- data.frame(y = rnorm(100),
                      x1 = runif(100),
                      x2 = runif(100),
                     treatment = factor(sample(rep(c("T1", "T2", "C"), c(25, 25, 50)))))

# Make distances
my_distances <- distances(my_data, dist_variables = c("x1", "x2"))

# Make matching
my_matching <- quickmatch(my_distances, my_data$treatment)

# Weights for ATE
weights_ate <- matching_weights(my_data$treatment, my_matching)

# Weights for ATT for T1
weights_att <- matching_weights(my_data$treatment, my_matching, target = "T1")

# Estimate treatment effects with WLS estimator (see `lm_match`)
effects <- lm(y ~ treatment + x1 + x2, data = my_data, weights = weights_att)
```

`qm_matching` *Constructor for qm_matching objects*

Description

The `qm_matching` function constructs a `qm_matching` object from existing matched group labels. The function does not derive matchings from sets of data points; see `quickmatch` for that functionality.

Usage

```
qm_matching(group_labels, unassigned_labels = NULL, ids = NULL)
```

Arguments

`group_labels`	a vector containing each unit's group label.
`unassigned_labels`	labels that denote unassigned units. If `NULL`, `NA` values in `group_labels` are used to denote unassigned points.
`ids`	IDs of the units. Should be a vector of the same length as `group_labels` or `NULL`. If `NULL`, the IDs are set to `1:length(group_labels)`.

Details

`qm_matching` objects are based on integer vectors, and it indexes matched groups starting with zero. The `qm_matching` class inherits from the `scclust` class.

Value

Returns a `qm_matching` object with the matching described by the provided labels.

Examples

```
# 10 units in 3 matched groups
matches1 <- qm_matching(c("A", "A", "B", "C", "B",
                          "C", "C", "A", "B", "B"))

# 8 units in 3 matched groups, 2 units unassigned
matches2 <- qm_matching(c(1, 1, 2, 3, 2,
                          NA, 3, 1, NA, 2))

# Custom labels indicating unassiged units
matches3 <- qm_matching(c("A", "A", "B", "C", "NONE",
                          "C", "C", "NONE", "B", "B"),
                        unassigned_labels = "NONE")

# Two different labels indicating unassiged units
matches4 <- qm_matching(c("A", "A", "B", "C", "NONE",
                          "C", "C", "0", "B", "B"),
                        unassigned_labels = c("NONE", "0"))

# Custom unit IDs
matches5 <- qm_matching(c("A", "A", "B", "C", "B",
                          "C", "C", "A", "B", "B"),
                        ids = letters[1:10])
```

quickmatch *Derive generalized full matchings*

Description

quickmatch constructs near-optimal generalized full matchings. The function expects the user to provide distances measuring the similarity of units and a set of matching constraints. It then constructs a matching so that units assigned to the same group are as similar as possible while satisfying the matching constraints.

Usage

```
quickmatch(
  distances,
  treatments,
  treatment_constraints = NULL,
  size_constraint = NULL,
  target = NULL,
  caliper = NULL,
  ...
)
```

Arguments

distances	distances object or a numeric vector, matrix or data frame. The parameter describes the similarity of the units to be matched. It can either be preprocessed distance information using a distances object, or raw covariate data. When called with covariate data, Euclidean distances are calculated unless otherwise specified.
treatments	factor specifying the units' treatment assignments.
treatment_constraints	named integer vector with the treatment constraints. If NULL, the function ensures that each matched group contains one unit from each treatment condition.
size_constraint	integer with the required total number of units in each group. Must be greater or equal to the sum of treatment_constraints. If NULL, no constraints other than the treatment constraints are imposed.
target	units to target the matching for. All units indicated by target are ensured to be assigned to a matched group (disregarding eventual caliper setting). Units not indicated by target could be left unassigned if they are not necessary to satisfy the matching constraints. If NULL, quickmatch targets the complete sample and ensures that all units are assigned to a group. If target is a logical vector with the same length as the sample size, units indicated with TRUE will be targeted. If target is an integer vector, the units with indices in target are targeted. Indices starts at 1 and target must be sorted. If target is a character vector, it should contain treatment labels, and the corresponding units (as given by treatments) will be targeted.

caliper	restrict the maximum within-group distance.
...	additional parameters to be sent either to the distances function when the distances parameter contains covariate data, or to the underlying sc_clustering function.

Details

The treatment_constraints parameter should be a named vector with treatment-specific constraints. For example, in a sample with treatment conditions "A", "B" and "C", the vector c("A" = 1, "B" = 2, "C" = 0) specifies that each matched group should contain at least one unit with treatment "A", at least two units with treatment "B" and any number of units with treatment "C". Treatments not specified in the vector defaults to zero. For example, the vector c("A" = 1, "B" = 2) is identical to the previous one. When treatment_constraints is NULL, the function requires at least one unit for each treatment in each group. In our current example, NULL would be shorthand for c("A" = 1, "B" = 1, "C" = 1).

The size_constraint parameter can be used to constrain the matched groups to contain at least a certain number of units in total (independently of treatment assignment). For example, if treatment_constraints = c("A" = 1, "B" = 2) and total_size_constraint = 4, each matched group will contain at least one unit assigned to "A", at least two units assigned to "B" and at least four units in total, where the fourth unit can be from any treatment condition.

The target parameter can be used to control which units are included in the matching. When target is NULL (the default), all units will be assigned to a matched group. When not NULL, the parameter indicates that some units must be assigned to matched group and that the remaining units can safely be ignored. This can be useful, for example, when one is interested in estimating treatment effects only for a certain type of units (e.g., the average treatment effect for the treated, ATT). It is particularly useful when units of interested are not represented in the whole covariate space (i.e., an one-sided overlap problem). Without the target parameter, the function would in such cases try to assign every unit to a group, including units in sparse regions that we are not interested in. This could lead to unnecessarily large and diverse matched groups. By specifying that some units are of interest only insofar as they help us satisfy the matching constraints (i.e., setting the target parameter to the appropriate value), we can avoid such situations.

Consider, as an example, a study with two treatment conditions, "A" and "B". Units assigned to "B" are more numerous and tend to have more extreme covariate values. We are, however, only interested in estimating the treatment effect for units assigned to "A". By specifying target = "A", the function ensures that all "A" units are assigned to matched groups. Some units assigned to treatment "B" – in particular the units with extreme covariate values – will be left unassigned. However, as those units are not of interest, they can safely be ignored, and we avoid groups of poor quality.

Even if some of the units that can be ignored are not needed to satisfy the matching constraints, it is rarely beneficial to discard them blindly; they can occasionally provide useful information. The default behavior when target is non-NULL is to assign as many of the ignorable units as possible given that the within-group distances do not increase too much (using secondary_unassigned_method = "estimated_radius"). This behavior might, however, reduce covariate balance in some instances. If called with secondary_unassigned_method = "ignore", units not specified in target will be discarded unless they are absolutely needed to satisfying the matching constraints. This tends to reduce bias since the within-group distances are minimized, but it could increase variance since we ignore potentially useful information in the sample. An intermediate alternative is to specify an aggressive caliper for the ignorable units, which is done with the `secondary_radius` parameter. (These parameters are part of the `sc_clustering` function that `quickmatch` calls. The `target` parameter corresponds to the `primary_data_points` parameter in that function.)

The `caliper` parameter constrains the maximum distance between units assigned to the same matched group. This is implemented by restricting the edge weight in the graph used to construct the matched groups (see `sc_clustering` for details). As a result, the caliper will affect all groups in the matching and, in general, make it harder for the function to find good matches even for groups where the caliper is not binding. In particular, a too tight `caliper` can lead to discarded units that otherwise would be assigned to a group satisfying both the matching constraints and the caliper. For this reason, it is recommended to set the `caliper` value quite high and only use it to avoid particularly poor matches. It strongly recommended to use the `caliper` parameter only when `primary_unassigned_method = "closest_seed"` in the underlying `sc_clustering` function (which is the default behavior).

`quickmatch` calls `sc_clustering` with `seed_method = "inwards_updating"`. The `seed_method` parameter governs how the seeds are selected in the nearest neighborhood graph that is used to construct the matched groups (see `sc_clustering` for details). The `"inwards_updating"` option generally works well and is safe with most datasets. Using `seed_method = "exclusion_updating"` often leads to better performance (in the sense of matched groups with more similar units), but it may increase run time. Discrete data (or more generally when units tend to be at equal distance to many other units) will lead to particularly poor run time with this option. If the data set has at least one continuous covariate, `"exclusion_updating"` is typically reasonably quick. A third option is `seed_method = "lexical"`, which decreases the run time relative to `"inwards_updating"` (sometimes considerably) at the cost of performance. `quickmatch` passes parameters on to `sc_clustering`, so to change `seed_method`, call `quickmatch` with the parameter specified as usual: `quickmatch(..., seed_method = "exclusion_updating")`.

Value

Returns a `qm_matching` object with the matched groups.

References

Sävje, Fredrik, Michael J. Higgins and Jasjeet S. Sekhon (2017), ‘Generalized Full Matching’, arXiv 1703.03882. https://arxiv.org/abs/1703.03882

See Also

See `sc_clustering` for the underlying function used to construct the matched groups.

Examples

```
# Construct example data
my_data <- data.frame(y = rnorm(100),
                      x1 = runif(100),
                      x2 = runif(100),
                     treatment = factor(sample(rep(c("T1", "T2", "C"), c(25, 25, 50)))))

# Make distances
my_distances <- distances(my_data, dist_variables = c("x1", "x2"))
```

```
# Make matching with one unit from "T1", "T2" and "C" in each matched group
quickmatch(my_distances, my_data$treatment)

# Require at least two "C" in each group
quickmatch(my_distances,
           my_data$treatment,
           treatment_constraints = c("T1" = 1, "T2" = 1, "C" = 2))

# Require groups with at least six units in total
quickmatch(my_distances,
           my_data$treatment,
           treatment_constraints = c("T1" = 1, "T2" = 1, "C" = 2),
           size_constraint = 6)

# Focus the matching to units assigned to "T1" and "T2" (i.e., all
# units assigned to "T1" or T2 will be assigned to a matched group).
# Units assigned to treatment "C" will be assigned to groups so to
# ensure that each group contains at least one unit of each treatment
# condition. Remaining "C" units could be left unassigned.
quickmatch(my_distances,
           my_data$treatment,
           target = c("T1", "T2"))

# Impose caliper
quickmatch(my_distances,
           my_data$treatment,
           caliper = 0.25)

# Call `quickmatch` directly with covariate data (ie., not pre-calculating distances)
quickmatch(my_data[c("x1", "x2")], my_data$treatment)

# Call `quickmatch` directly with covariate data using Mahalanobis distances
quickmatch(my_data[c("x1", "x2")],
           my_data$treatment,
           normalize = "mahalanobize")
```

Index