Package ‘MatchIt’

November 4, 2020

Version 4.0.0

Title Nonparametric Preprocessing for Parametric Causal Inference

Description Selects matched samples of the original treated and control groups with similar covariate distributions -- can be used to match exactly on covariates, to match on propensity scores, or perform a variety of other matching procedures. The package also implements a series of recommendations offered in Ho, Imai, King, and Stuart (2007) ¡DOI:10.1093/pan/mpl013¿.

Depends R (¿= 3.0.0)

Imports backports (¿= 1.1.8),
Rcpp (¿= 1.0.5)

Suggests optmatch,
Matching,
rgenoud,
nnet,
rpart,
mgcv,
CBPS,
dbarts,
randomForest,
cobalt (¿= 4.2.3),
boot,
lmtest,
sandwich (¿= 2.5-1),
survival,
RcppProgress (¿= 0.4.2),
knitr,
rmarkdown

LinkingTo Rcpp, RcppProgress

Encoding UTF-8

LazyData true

License GPL (¿=2)

BugReports https://github.com/kosukeimai/MatchIt/issues

VignetteBuilder knitr

R topics documented:

add_s.weights *Add sampling weights to a matchit object*

Description

Adds sampling weights to a `matchit` object so that they are incorporated into balance assessment and creation of the weights. This would typically only be used when an argument to `s.weights` was not supplied to `matchit` (i.e., because they were not to be included in the estimation of the propensity score) but sampling weights are required for generalizing an effect to the correct population. Without adding sampling weights to the `matchit` object, balance assessment tools (i.e., `summary.matchit` and `plot.matchit`) will not calculate balance statistics correctly, and the weights produced by `match.data` and `get_matches` will not incorporate the sampling weights.

Usage

```
add_s.weights(m, s.weights = NULL, data = NULL)
```

Arguments

`m` a `matchit` object; the output of a call to `matchit`, typically with the `s.weights` argument unspecified.

`s.weights` an numeric vector of sampling weights to be added to the `matchit` object. Can also be specified as a string containing the name of variable in `data` to be used or a one-sided formula with the variable on the right-hand side (e.g., `~ SW`).

`data` a data frame containing the sampling weights if given as a string or formula. If unspecified, `add_s.weights` will attempt to find the dataset using the environment of the `matchit` object.

Value

a `matchit` object with an `s.weights` component containing the supplied sampling weights. The `nn` component containing the sample sizes before and after matching will be adjusted to incorporate the sampling weights. If `s.weights = NULL`, the original `matchit` object is returned.

Author(s)

Noah Greifer

See Also

matchit; match.data

Examples

```
data("lalonde")

# Generate random sampling weights, just
# for this example
sw <- rchisq(nrow(lalonde), 2)

# NN PS match using logistic regression PS that doesn't
# include sampling weights
m.out <- matchit(treat ~ age + educ + race + nodegree +
                   married  + re74 + re75, data = lalonde)

m.out

# Add s.weights to the matchit object
m.out <- add_s.weights(m.out, sw)

m.out #note additional output

# Check balance; note that sample sizes incorporate
# s.weights
summary(m.out, improvement = FALSE)
```

distance *Propensity scores and other distance measures*

Description

Several matching methods require or can involve the distance between treated and control units. Options include the Mahalanobis distance, propensity score distance, or distance between user-supplied values. Propensity scores are also used for common support via the `discard` options and for defined calipers. This page documents the options that can be supplied to the `distance` argument to `matchit`.

There are two ways to specify the `distance` argument: 1) as the string `"mahalanobis"`, 2) as a string containing the name of a method for estimating propensity scores, or 3) as a vector of values whose pairwise differences define the distance between units.

When `distance` is specified as one of the allowed strings (described below) other than `"mahalanobis"`, a propensity score is estimated using the variables in `formula` and the method corresponding to the given argument. This propensity score can be used to compute the distance between units as the absolute difference between the propensity scores of pairs of units. In this respect, the propensity score is more like a ”position” measure than a distance measure, since it is the pairwise difference that form the distance rather than the propensity scores themselves. Still, this naming convention is used to reflect their primary purpose without committing to the status of the estimated values as propensity scores, since transformations of the scores are allowed and user-supplied values that are not propensity scores can also be supplied (detailed below). Propensity scores can also be used to create calipers and common support restrictions, whether or not they are used in the actual distance measure used in the matching, if any.

In addition to the `distance` argument, two other arguments can be specified that relate to the estimation and manipulation of the propensity scores. The `link` argument allows for different links to be used in models that require them such as generalized linear models, for which the logit and probit links are allowed, among others. In addition to specifying the link, the `link` argument can be used to specify whether the propensity score or the linearized version of the propensity score should be used; by specifying `link = "linear.{link}"`, the linearized version will be used.

The `distance.options` argument can also be specified, which should be a list of values passed to the propensity score-estimating function, for example, to choose specific options or tuning parameters for the estimation method. If `formula`, `data`, or `verbose` are not supplied to `distance.options`, the corresponding arguments from `matchit` will be automatically supplied. See the Examples for demonstrations of the uses of `link` and `distance.options`. When `s.weights` is supplied in the call to `matchit`, it will automatically be passed to the propensity score-estimating function as the `weights` argument unless otherwise described below.

Allowable options

Below are the allowed options for `distance`:

`"glm"` The propensity scores are estimated using a generalized linear model (e.g., logistic regression). The `formula` supplied to `matchit` is passed directly to `glm`, and `predict.glm` is used to compute the propensity scores. The `link` argument can be specified as a link function supplied to `binomial`, e.g., `"logit"`, which is the default. When `link` is prepended by `"linear."`, the linear predictor is used instead of the predicted probabilities. `distance = "glm"` with `link = "logit"` (logistic regression) is the default in `matchit`.

`"gam"` The propensity scores are estimated using a generalized additive model. The `formula` supplied to `matchit` is passed directly to `gam` in the **mgcv** package, and `predict.gam` is used to compute the propensity scores. The `link` argument can be specified as a link function supplied to `binomial`, e.g., `"logit"`, which is the default. When `link` is prepended by `"linear."`, the linear predictor is used instead of the predicted probabilities. Note that unless the smoothing functions `s`, `te`, `ti`, or `t2` are used in `formula`, a generalized additive model is identical to a generalized linear model and will estimate the same propensity scores as `glm`. See the documentation for `gam`, `formula.gam`, and \pkgfunmgcvgam.models for more information on how to specify these models. Also note that the formula returned in the `matchit` output object will be a simplified version of the supplied formula with smoothing terms removed (but all named variables present).

`"rpart"` The propensity scores are estimated using a classification tree. The `formula` supplied to `matchit` is passed directly to `rpart` in the **rpart** package, and `predict.rpart` is used to compute the propensity scores. The `link` argument is ignored, and predicted probabilities are always returned as the distance measure.

`"randomforest"` The propensity scores are estimated using a random forest. The `formula` supplied to `matchit` is passed directly to `randomForest` in the **randomForest** package, and `predict.randomForest` is used to compute the propensity scores. The `link` argument is ignored, and predicted probabilities are always returned as the distance measure. When `s.weights` is supplied to `matchit`, it will not be passed to `randomForest` because `randomForest` does not accept weights.

`"nnet"` The propensity scores are estimated using a single-hidden-layer neural network. The `formula` supplied to `matchit` is passed directly to `nnet` in the **nnet** package, and `fitted` is used to compute the propensity scores. The `link` argument is ignored, and predicted probabilities are always returned as the distance measure. An argument to `size` must be supplied to `distance.options` when using `method = "nnet"`.

`"cbps"` The propensity scores are estimated using the covariate balancing propensity score (CBPS) algorithm, which is a form of logistic regression where balance constraints are incorporated to a generalized method of moments estimation of of the model coefficients. The `formula` supplied to `matchit` is passed directly to `CBPS` in the **CBPS** package, and `fitted` is used to compute the propensity scores. The `link` argument can be specified as `"linear"` to use the linear predictor instead of the predicted probabilities. No other links are allowed. The `estimand` argument supplied to `matchit` will be used to select the appropriate estimand for use in defining the balance constraints, so no argument needs to be supplied to `ATT` in `CBPS`.

`"bart"` The propensity scores are estimated using Bayesian additive regression trees (BART). The `formula` supplied to `matchit` is passed directly to `bart2` in the **dbarts** package, and `fitted` is used to compute the propensity scores. The `link` argument can be specified as `"linear"` to use the linear predictor instead of the predicted probabilities. When `s.weights` is supplied to `matchit`, it will not be passed to `bart2` because the `weights` argument in `bart2` does not correspond to sampling weights.

`"mahalanobis"` No propensity scores are estimated. Rather than using the propensity score difference as the distance between units, the Mahalanobis distance is used instead. See `mahalanobis` for details on how it is computed. The Mahalanobis distance is always computed using all the variables in `formula`. With this specification, calipers and common support restrictions cannot be used and the `distance` component of the output object will be empty because no propensity scores are estimated. The `link` and `distance.options` arguments are ignored. See individual methods pages for whether the Mahalanobis distance is allowed and how it is used. Sometimes this setting is just a placeholder to indicate that no propensity score is to be estimated (e.g., with `method = "genetic"`). To perform Mahalanobis distance matching *and* estimate propensity scores to be used for a purpose other than matching, the `mahvars` argument should be used along with a different specification to `distance`. See the individual matching method pages for details on how to use `mahvars`.

`distance` can also be supplied as a numeric vector whose values will be taken to function like propensity scores; their pairwise difference will define the distance between units. This might be beneficial to supply propensity scores computed outside `matchit` or to resupply `matchit` with propensity scores estimated before without having to recompute them. When `distance` is a supplied as a numeric vector, `link` and `distance.options` are ignored.

Outputs

When specifying an argument to `distance` that estimates a propensity score, the output of the function called to estimate the propensity score (e.g., the `glm` object when `distance = "glm"`) will be included in the `matchit` output object in the `model` component. When `distance` is anything other than `"mahalanobis"`, the estimated or supplied distance measures will be included in the `matchit` output object in the `distance` component.

Note

In versions of **MatchIt** prior to 4.0.0, `distance` was specified in a slightly different way. When specifying arguments using the old syntax, they will automatically be converted to the corresponding method in the new syntax but a warning will be thrown. `distance = "logit"`, the old default, will still work in the new syntax, though `distance = "glm",link = "logit"` is preferred (note that these are the default settings and don't need to be made explicit).

Examples

```
data("lalonde")
# Linearized probit regression PS:
m.out1 <- matchit(treat ~ age + educ + race + married +
                    nodegree + re74 + re75, data = lalonde,
                  distance = "glm", link = "linear.probit")

# GAM logistic PS with smoothing splines (s()):
m.out2 <- matchit(treat ~ s(age) + s(educ) + race + married +
                    nodegree + re74 + re75, data = lalonde,
                  distance = "gam")
summary(m.out2$model)

# CBPS for ATC matching w/replacement, using the just-
# identified version of CBPS (setting method = "exact"):
m.out3 <- matchit(treat ~ age + educ + race + married +
                    nodegree + re74 + re75, data = lalonde,
                  distance = "cbps", estimand = "ATC",
                  distance.options = list(method = "exact"),
                  replace = TRUE)

# Mahalanobis distance matching - no PS estimated
m.out4 <- matchit(treat ~ age + educ + race + married +
                    nodegree + re74 + re75, data = lalonde,
                  distance = "mahalanobis")

m.out4$distance #NULL

# Mahalanobis distance matching with PS estimated
# for use in a caliper; matching done on mahvars
m.out5 <- matchit(treat ~ age + educ + race + married +
                    nodegree + re74 + re75, data = lalonde,
                  distance = "glm", caliper = .1,
                  mahvars = ~ age + educ + race + married +
                                nodegree + re74 + re75)

summary(m.out5)

# User-supplied propensity scores
p.score <- fitted(glm(treat ~ age + educ + race + married +
                        nodegree + re74 + re75, data = lalonde,
                      family = binomial))

m.out6 <- matchit(treat ~ age + educ + race + married +
                    nodegree + re74 + re75, data = lalonde,
                  distance = p.score)
```

`lalonde` *Data from National Supported Work Demonstration and PSID, as analyzed by Dehejia and Wahba (1999).*

Description

This is a subsample of the data from the treated group in the National Supported Work Demonstration (NSW) and the comparison sample from the Population Survey of Income Dynamics (PSID). This data was previously analyzed extensively by Lalonde (1986) and Dehejia and Wahba (1999).

Usage

```
data(lalonde)
```

Format

A data frame with 614 observations (185 treated, 429 control). There are 9 variables measured for each individual.

- "treat" is the treatment assignment (1=treated, 0=control).
- "age" is age in years.
- "educ" is education in number of years of schooling.
- "race" is the individual's race/ethnicity, (Black, Hispanic, or White).
- "married" is an indicator for married (1=married, 0=not married).
- "nodegree" is an indicator for whether the individual has a high school degree (1=no degree, 0=degree).
- "re74" is income in 1974, in U.S. dollars.
- "re75" is income in 1975, in U.S. dollars.
- "re78" is income in 1978, in U.S. dollars.

"treat" is the treatment variable, "re78" is the outcome, and the others are pre-treatment covariates.

References

Lalonde, R. (1986). Evaluating the econometric evaluations of training programs with experimental data. American Economic Review 76: 604-620.

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

match.data — *Construct a matched dataset from a* `matchit` *object*

Description

`match.data` and `get_matches` create a data frame with additional variables for the distance measure, matching weights, and subclasses after matching. This dataset can be used to estimate treatment effects after matching or subclassification. `get_matches` is most useful after matching with replacement; otherwise, `match.data` is more flexible. See Details below for the difference between them.

Usage

```
match.data(object, group = "all", distance = "distance",
           weights = "weights", subclass = "subclass",
           data = NULL, include.s.weights = TRUE,
           drop.unmatched = TRUE)

get_matches(object, distance = "distance",
            weights = "weights", subclass = "subclass",
            id = "id", data = NULL, include.s.weights = TRUE)
```

Arguments

`object` a `matchit` object; the output of a call to `matchit`.

`group` which group should comprise the matched dataset: `"all"` for all units, `"treated"` for just treated units, or `"control"` for just control units. Default is `"all"`.

`distance` a string containing the name that should be given to the variable containing the distance measure in the data frame output. Default is `"distance"`, but `"prop.score"` or similar might be a good alternative if propensity scores were used in matching. Ignored if a distance measure was not supplied or estimated in the call to `matchit`.

`weights` a string containing the name that should be given to the variable containing the matching weights in the data frame output. Default is `"weights"`.

`subclass` a string containing the name that should be given to the variable containing the subclasses or matched pair membership in the data frame output. Default is `"subclass"`.

`id` a string containing the name that should be given to the variable containing the unit IDs in the data frame output. Default is `"id"`. Only used with `get_matches`; for `match.data`, the units IDs are stored in the row names of the returned data frame.

`data` a data frame containing the original dataset to which the computed output variables (`distance`, `weights`, and/or `sublcass`) should be appended. If empty, `match.data` will attempt to find the dataset using the environment of the `matchit` object, which can make this unreliable if `match.data` is used in a fresh R session or if the original dataset changed between calling `matchit` and `match.data`. It is always safest to supply a data frame, which should have as many rows as and be in the same order as the data in the original call to `matchit`. The same goes for `get_matches`, which calls `match.data` internally.

`include.s.weights`
: `logical`; whether to multiply the estimated weights by the sampling weights supplied to `matchit()`, if any. Default is `TRUE`. If `FALSE`, the weights in the `match.data` or `get_matches` output should be multiplied by the sampling weights before being supplied to the function estimating the treatment effect in the matched data.

`drop.unmatched`
: `logical`; whether the returned data frame should contain all units (`FALSE`) or only units that were matched (i.e., have a matching weight greater than zero) (`TRUE`). Default is `TRUE` to drop unmatched units.

Details

`match.data` creates a dataset with one row per unit. It will be identical to the dataset supplied except that several new columns will be added containing information related to the matching. When `drop.unmatched = TRUE`, the default, units with weights of zero, which are those units that were discarded by common support or the caliper or were simply not matched, will be dropped from the dataset, leaving only the subset of matched units. The idea is for the output of `match.data` to be used as the dataset input in calls to `glm` or similar to estimate treatment effects in the matched sample. Unless 1:1 matching without replacement was performed, it is important to include the weights in the estimation of the effect and its standard error. The subclasses, when created, can be used as fixed effects, random effects, or as clusters for clustered standard errors. Subclasses will only be included if there is a `subclass` component in the `matchit` object, which does not occur with matching with replacement, in which case `get_matches` should be used.

`get_matches` is similar to `match.data`; the primary difference occurs when matching is performed with replacement, i.e., when units do not belong to a single matched pair. In this case, the output of `get_matches` will be a dataset that contains one row per unit for each pair they are a part of. For example, if matching was performed with replacement and a control unit was matched to two treated units, that control unit will have two rows in the output dataset, one for each pair it is a part of. Weights are computed for each row, and are equal to the inverse of the number of control units in each control unit's subclass. Unmatched units are dropped. An additional column with unit IDs will be created (named using the `id` argument) to identify when the same unit is present in multiple rows. This dataset structure allows for the inclusion of both subclass membership and repeated use of units, unlike the output of `match.data`, which lacks subclass membership when matching is done with replacement. A `match.matrix` component of the `matchit` object must be present to use `get_matches`; in some forms of matching, it is absent, in which case `match.data` should be used instead.

Value

A data frame containing the data supplied in the `data` argument or in the original call to `matchit` with the computed output variables appended as additional columns, named according the arguments above. For `match.data`, the `group` and `drop.unmatched` arguments control whether only subsets of the data are returned. See Details above for how `match.data` and `get_matches` differ. Note that `get_matches` sorts the data by subclass and treatment status, unlike `match.data`, which uses the order of the data.

If `data` or the original dataset supplied to `matchit` was a `data.table` or `tbl`, the `match.data` output will have the same class, but the `get_matches` output will always be a base R `data.frame`.

See Also

matchit

Examples

```
data("lalonde")

# 4:1 matching w/replacement
m.out1 <- matchit(treat ~ age + educ + married +
                    race + nodegree + re74 + re75,
                  data = lalonde, replace = TRUE,
                  caliper = .05, ratio = 4)

m.data1 <- match.data(m.out1, data = lalonde,
                      distance = "prop.score")
dim(m.data1) #one row per matched unit
head(m.data1, 10)

g.matches1 <- get_matches(m.out1, data = lalonde,
                          distance = "prop.score")
dim(g.matches1) #multiple rows per matched unit
head(g.matches1, 10)
```

matchit *Matching for Causal Inference*

Description

matchit is the main function of **MatchIt** and performs pairing, subset selection, and subclassification with the aim of creating treatment and control groups balanced on included covariates. MatchIt implements the suggestions of Ho, Imai, King, and Stuart (2007) for improving parametric statistical models by preprocessing data with nonparametric matching methods. MatchIt implements a wide range of sophisticated matching methods, making it possible to greatly reduce the dependence of causal inferences on hard-to-justify, but commonly made, statistical modeling assumptions. The software also easily fits into existing research practices since, after preprocessing with MatchIt, researchers can use whatever parametric model they would have used without MatchIt, but produce inferences with substantially more robustness and less sensitivity to modeling assumptions.

This page documents the overall use of matchit, but for specifics of how matchit works with individual matching methods, see the individual pages linked in the Details section below.

Usage

```
matchit(formula, data = NULL, method = "nearest",
        distance = "glm", link = "logit",
        distance.options = list(), estimand = "ATT",
        exact = NULL, mahvars = NULL, discard = "none",
        reestimate = FALSE, s.weights = NULL,
        replace = FALSE, m.order = NULL,
        caliper = NULL, std.caliper = TRUE, ratio = 1,
        verbose = FALSE, ...)
```

```
## S3 method for class 'matchit'
print(x, ...)
```

Arguments

`formula` a two-sided `formula` object containing the treatment and covariates to be used in creating the distance measure used in the matching. This formula will be supplied to the functions that estimate the distance measure. The formula should be specified as `A ~ X1 + X2 + ...` where `A` represents the treatment variable and `X1` and `X2` are covariates.

`data` a data frame containing the variables named in `formula` and possible other arguments. If not found in `data`, the variables will be sought in the environment.

`method` the matching method to be used. The allowed methods are `"nearest"` for nearest neighbor matching (on the propensity score by default), `"optimal"` for optimal pair matching, `"full"` for optimal full matching, `"genetic"` for genetic matching, `"exact"` for exact matching, and `"subclass"` for subclassification. When set to `NULL`, no matching will occur, but propensity score estimation and common support restrictions will still occur if requested. See the linked pages for each method for more details on what these methods do, how the arguments below are used by each on, and what additional arguments are allowed.

`distance` the distance measure to be used. Can be either a string containing the name of a distance measure or a vector of already-computed distance measures. The distance measures should be values whose pairwise difference is the distance between two units, e.g., propensity scores for propensity score matching. See `distance` for allowable options. The default is `"glm"` for propensity scores estimated with logistic regression using `glm`. Ignored for some methods; see individual methods pages for how the distance measure is used.

`link` when `distance` is specified as a string, an additional argument controlling the link function used in estimating the distance measure. Allowable options depend on the specific `distance` value specified. See `distance` for allowable options with each option. The default is `"logit"`, which, along with `distance = "glm"`, identifies the default measure as logistic regression propensity scores.

`distance.options`
a named list containing additional arguments supplied to the function that estimates the distance measure as determined by the argument to `distance`. See `distance` for an example of its use.

`estimand` a string containing the name of the target estimand desired. Can be one of `"ATT"` or `"ATC"`. Some methods accept `"ATE"` as well. Default is `"ATT"`. See Details and the individual methods pages for information on how this argument is used.

`exact` for methods that allow it, for which variables exact matching should take place. Can be specified as a string containing the names of variables in `data` to be used or a one-sided formula with the desired variables on the right-hand side (e.g., `~ X3 + X4`). See the individual methods pages for information on whether and how this argument is used.

mahvars for methods that allow it, for which variables Mahalanobis matching should take place when a distance measure other than "mahalanobis" is used. Usually used to perform Mahalanobis distance matching within propensity score calipers, where the propensity scores are computed using formula and distance. Can be specified as a string containing the names of variables in data to be used or a one-sided formula with the desired variables on the right-hand side (e.g., ~ X3 + X4). See the individual methods pages for information on whether and how this argument is used.

discard a string containing a method for discarding units outside a region of common support. When a propensity score is estimated or supplied to distance, the options are "none", "treated", "control", or "both". For "none", no units are discarded for common support. Otherwise, units whose propensity scores fall outside the corresponding region are discarded. Can also be a logical vector where TRUE indicates the unit is to be discarded. Default is "none" for no common support restriction. See Details.

reestimate if discard is not "none", whether to re-estimate the propensity score in the remaining sample. Default is FALSE to use the propensity scores estimated in the original sample.

s.weights an optional numeric vector of sampling weights to be incorporated into propensity score models and balance statistics. Can also be specified as a string containing the name of variable in data to be used or a one-sided formula with the variable on the right-hand side (e.g., ~ SW). Not all propensity score models accept sampling weights; see distance for information on which do and do not.

replace for methods that allow it, whether matching should be done with replacement (TRUE), where control units are allowed to be matched to several treated units, or without replacement (FALSE), where control units can only be matched to one treated unit each. See the individual methods pages for information on whether and how this argument is used. Default is FALSE for matching without replacement.

m.order for methods that allow it, the order that the matching takes place. Allowable options depend on the matching method but include "largest", where matching takes place in descending order of distance measures; "smallest", where matching takes place in ascending order of distance measures; "random", where matching takes place in a random order; and "data" where matching takes place based on the order of units in the data. When m.order = "random", results may differ across different runs of the same code unless a seed is set and specified with set.seed. See the individual methods pages for information on whether and how this argument is used. The default of NULL corresponds to "largest" when a propensity score is estimated or supplied and "data" otherwise.

caliper for methods that allow it, the width(s) of the caliper(s) to use in matching. Should be a numeric vector with each value named according to the variable to which the caliper applies. To apply to the distance measure, the value should be unnamed. See the individual methods pages for information on whether and how this argument is used. The default is NULL for no caliper.

std.caliper logical; when a caliper is specified, whether the the caliper is in standard deviation units (TRUE) or raw units (FALSE). Can either be of length

1, applying to all calipers, or of length equal to the length of `caliper`. Default is `TRUE`.

`ratio` for methods that allow it, how many control units should be matched to each treated unit in k:1 matching. Should be a single integer value. See the individual methods pages for information on whether and how this argument is used. The default is 1 for 1:1 matching.

`verbose` `logical`; whether information about the matching process should be printed to the console. What is printed depends on the matching method. Default is `FALSE` for no printing other than warnings.

`...` additional arguments passed to the functions used in the matching process. See the individual methods pages for information on what additional arguments are allowed for each method. Ignored for `print`.

`x` a `matchit` object.

Details

Details for the various matching methods can be found at the following help pages:

- `method_nearest` for nearest neighbor matching
- `method_optimal` for optimal pair matching
- `method_full` for optimal full matching
- `method_genetic` for genetic matching
- `method_exact` for exact matching
- `method_subclass` for subclassification

The pages contain information on what the method does, which of the arguments above are allowed with them and how they are interpreted, and what additional arguments can be supplied to further tune the method. Note that the default method with no arguments supplied other than `formula` and `data` is 1:1 nearest neighbor matching without replacement on a propensity score estimated using a logistic regression of the treatment on the covariates. This is not the same default offered by other matching programs, such as those in **Matching**, `teffects` in Stata, or `PROC PSMATCH` in SAS, so care should be taken if trying to replicate the results of those programs.

When `method = NULL`, no matching will occur, but any propensity score estimation and common support restriction will. This can be a simple way to estimate the propensity score for use in future matching specifications without having to reestimate it each time. The `matchit` output with no matching can be supplied to `summary` to examine balance prior to matching on any of the included covariates and on the propensity score if specified. All arguments other than `distance`, `discard`, and `reestimate` will be ignored.

See the `distance` argument for details on the several ways to specify the `distance` and `link` arguments to estimate propensity scores and create distance measures.

When the treatment variable is not a `0/1` variable, it will be coerced to one and returned as such in the `matchit` output (see section Value, below). The following rules are used: 1) if `0` is one of the values, it will be considered the control and the other value the treated; otherwise, 2) if the variable is a factor, `levels(treat)[1]` will be considered control and the other variable the treated; otherwise, 3) `sort(unique(treat))[1]` will be considered control and the other value the treated. It is safest to ensure the treatment variable is a `0/1` variable.

The `discard` option implements a common support restriction. It can only be used when a distance measure is estimated or supplied, i.e., when `distance` is specified as something other than "mahalanobis", and is ignored for some matching methods. When specified as "treated", treated units whose distance measure is outside the range of distance measures of the control units will be discarded. When specified as "control", control units whose distance measure is outside the range of distance measures of the treated units will be discarded. When specified as "both", treated and controls units whose distance measure is outside the intersection of the range of distance measures of the treated units and the range of distance measures of the control units will be discarded. When reestimate = TRUE and distance corresponds to a propensity score-estimating function, the propensity scores are re-estimated in the remaining units prior to being used for matching or calipers.

Caution should be used when interpreting effects estimated with various values of estimand. Setting estimand = "ATT" doesn't necessarily mean the average treatment effect in the treated is being estimated; it just means that for matching methods, treated units will be untouched and given weights of 1 and control units will be matched to them (and the opposite for estimand = "ATC"). If a caliper is supplied or treated units are removed for common support or some other reason (e.g., lacking matches when using exact matching), the actual estimand targeted is not the ATT but the treatment effect in the matched sample. The argument to estimand simply triggers which units are matched to which, and for subclassification-based methods (exact matching, CEM, full matching, and subclassification), determines the formula used to compute the subclassification weights.

Computing Weights: Matching weights are computed in one of two ways depending on whether matching was done with replacement or not.

For matching without replacement, each unit is assigned to a subclass, which represents the pair they are a part of (in the case of k:1 matching) or the subclass they belong to (in the case of exact matching, coarsened exact matching, full matching, or subclassification). The formula for computing the weights depends on the argument supplied to estimand. A new subclass "propensity score" (p) is computed as the proportion of units in each subclass that are in the treated group, and all units in that subclass are assigned that propensity score. Weights are then computed using the standard formulas for inverse probability weights: for the ATT, weights are 1 for the treated units and p/(1-p) for the control units; for the ATC, weights are (1-p)/p for the treated units and 1 for the control units; for the ATE, weights are 1/p for the treated units and 1/(1-p) for the control units.

For matching with replacement, units are not assigned to unique subclasses. For the ATT, each treated unit gets a weight of 1. Each control unit is weighted as the sum of the inverse of the number of control units matched to the same treated unit across its matches. For example, if a control unit was matched to a treated unit that had two other control units matched to it, and that same control was matched to a treated unit that had one other control unit matched to it, the control unit in question would get a weight of $1/3 + 1/2 = 5/6$. For the ATC, the same is true with the treated and control labels switched. The weights are computed using the match.matrix component of the matchit output object.

In each treatment group, weights are divided by the mean of the nonzero weights in that treatment group to make the weights sum to the number of units in that treatment group. If sampling weights are included through the s.weights argument, they will be included in the matchit output object but not incorporated into the matching weights. When estimating a treatment effect, multiply the matching weights by the sampling weights.

Value

When method is something other than "subclass", a matchit object with the following components:

`match.matrix` a matrix containing the matches. The rownames correspond to the treated units and the values in each row are the names (or indices) of the control units matched to each treated unit. When treated units are matched to different numbers of control units (e.g., with exact matching or matching with a caliper), empty spaces will be filled with `NA`.

`subclass` a factor containing matching pair/stratum membership for each unit. Unmatched units will have a value of `NA`. Not included when `replace = TRUE`.

`weights` a numeric vector of estimated matching weights. Unmatched units will have a weight of zero.

`model` the fit object of the model used to estimate propensity scores when `distance` is specified and not `"mahalanobis"` or a numeric vector. When `reestimate = TRUE`, this is the model estimated after discarding units.

`X` a data frame of covariates mentioned in `formula`, `exact`, and `mahvars`.

`call` the `matchit()` call.

`info` information on the matching method and distance measures used.

`estimand` the argument supplied to `estimand`.

`formula` the `formula` supplied.

`treat` a vector of treatment status converted to zeros (0) and ones (1) if not already in that format.

`distance` a vector of distance values (i.e., propensity scores) when `distance` is specified and not `"mahalanobis"`.

`discarded` a logical vector denoting whether each observation was discarded (`TRUE`) or not (`FALSE`) by the argument to `discard`.

`s.weights` the vector of sampling weights supplied to the `s.weights` argument, if any.

`exact` a one-sided formula containing the variables, if any, supplied to `exact`.

`mahvars` a one-sided formula containing the variables, if any, supplied to `mahvars`.

`nn` a matrix of the sample sizes of the treated and control groups before and after matching. See `summary.matchit` for details.

When `method = "subclass"`, a `matchit.subclass` object with the same components as above except that `match.matrix` is excluded and two additional components, `q.cut` and `qn`, are included, containing a vector of the distance measure cutpoints used to define the subclasses and a matrix of the subclass sample sizes, respectively. See `method_subclass` for details.

Author(s)

Daniel Ho <`dho@law.stanford.edu`>; Kosuke Imai <`imai@harvard.edu`>; Gary King <`king@harvard.edu`>; Elizabeth Stuart <`estuart@jhsph.edu`>

Version 4.0.0 update by Noah Greifer <`noah.greifer@gmail.com`>

References

Ho, D. E., Imai, K., King, G., & Stuart, E. A. (2007). Matching as Nonparametric Preprocessing for Reducing Model Dependence in Parametric Causal Inference. Political Analysis, 15(3), 199–236. doi: 10.1093/pan/mpl013

Ho, D. E., Imai, K., King, G., & Stuart, E. A. (2011). MatchIt: Nonparametric Preprocessing for Parametric Causal Inference. Journal of Statistical Software, 42(8). doi: 10.18637/jss.v042.i08

See Also

summary.matchit for balance assessment after matching. plot.matchit for plots of covariate balance and propensity score overlap after matching.

vignette("MatchIt") for an introduction to matching with MatchIt; vignette("matching-methods") for descriptions of the variety of matching methods and options available; vignette("assessing-balance") for information on assessing the quality of a matching specification; vignette("estimating-effects") for instructions on how to estimate treatment effects after matching; and vignette("sampling-weights") for a guide to using **MatchIt** with sampling weights.

Examples

```
data("lalonde")

# Default: 1:1 NN PS matching w/o replacement

m.out1 <- matchit(treat ~ age + educ + race + nodegree +
                   married + re74 + re75, data = lalonde)
m.out1
summary(m.out1)

# 1:1 NN Mahalanobis distance matching w/ replacement and
# exact matching on married and race

m.out2 <- matchit(treat ~ age + educ + race + nodegree +
                   married + re74 + re75, data = lalonde,
                   distance = "mahalanobis", replace = TRUE,
                   exact = ~ married + race)
m.out2
summary(m.out2)

# 2:1 NN Mahalanobis distance matching within caliper defined
# by a probit pregression PS

m.out3 <- matchit(treat ~ age + educ + race + nodegree +
                   married + re74 + re75, data = lalonde,
                   distance = "glm", link = "probit",
                   mahvars = ~ age + educ + re74 + re75,
                   caliper = .1, ratio = 2)
m.out3
summary(m.out3)

# Optimal full PS matching for the ATE within calipers on
# PS, age, and educ

m.out4 <- matchit(treat ~ age + educ + race + nodegree +
                   married + re74 + re75, data = lalonde,
                   method = "full", estimand = "ATE",
                   caliper = c(.1, age = 2, educ = 1),
                   std.caliper = c(TRUE, FALSE, FALSE))
m.out4
summary(m.out4)

# Subclassification on a logistic PS with 10 subclasses after
# discarding controls outside common support of PS

s.out1 <- matchit(treat ~ age + educ + race + nodegree +
                   married + re74 + re75, data = lalonde,
                   method = "subclass", distance = "glm",
                   discard = "control", subclass = 10)
s.out1
summary(s.out1)
```

`method_exact` *Exact Matching*

Description

In `matchit`, setting `method = "exact"` performs exact matching. With exact matching, a complete cross of the covariates is used to form subclasses defined by each combination of the covariate levels. Any subclass that doesn't contain both treated and control units is discarded, leaving only subclasses containing treatment and control units that are exactly equal on the included covariates. The benefits of exact matching are that confounding due to the covariates included is completely eliminated, regardless of the functional form of the treatment or outcome models. The problem is that typically many units will be discarded, sometimes dramatically reducing precision and changing the target population of inference. To use exact matching in combination with another matching method (i.e., to exact match on some covariates and some other form of matching on others), use the `exact` argument with that method.

This page details the allowable arguments with `method = "exact"`. See `matchit` for an explanation of what each argument means in a general context and how it can be specified.

Below is how `matchit` is used for exact matching:

```
matchit(formula, data = NULL, method = "exact",
        estimand = "ATT", s.weights = NULL,
        verbose = FALSE, ...)
```

Arguments

`formula` a two-sided `formula` object containing the treatment and covariates to be used in creating the subclasses defined by a full cross of the covariate levels.

`data` a data frame containing the variables named in `formula`. If not found in `data`, the variables will be sought in the environment.

`method` set here to `"exact"`.

`estimand` a string containing the desired estimand. Allowable options include `"ATT"`, `"ATC"`, and `"ATE"`. The estimand controls how the weights are computed; see the Computing Weights section at `matchit` for details.

`s.weights` the variable containing sampling weights to be incorporated into balance statistics. These weights do not affect the matching process.

`verbose` `logical`; whether information about the matching process should be printed to the console.

`...` ignored.

The arguments `distance` (and related arguments), `exact`, `mahvars`, `discard` (and related arguments), `replace`, `m.order`, `caliper` (and related arguments), and `ratio` are ignored with a warning.

Outputs

All outputs described in matchit are returned with `method = "exact"` except for `match.matrix`. This is because matching strata are not indexed by treated units as they are in some other forms of matching.

References

In a manuscript, you don't need to cite another package when using `method = "exact"` because the matching is performed completely within **MatchIt**. For example, a sentence might read:

Exact matching was performed using the MatchIt package (Ho, Imai, King, & Stuart, 2011) in R.

See Also

matchit for a detailed explanation of the inputs and outputs of a call to `matchit`. The `exact` argument can be used with other methods to perform exact matching in combination with other matching methods.

Examples

```
data("lalonde")

# Exact matching on age, race, married, and educ
m.out1 <- matchit(treat ~ age + race + married + educ, data = lalonde,
                  method = "exact")
m.out1
summary(m.out1)
```

---

method_full *Optimal Full Matching*

Description

In matchit, setting `method = "full"` performs optimal full matching, which is a form of subclassification wherein all units, both treatment and control (i.e., the "full" sample), are assigned to a subclass and receive at least one match. The matching is optimal in the sense that that sum of the absolute distances between the treated and control units in each subclass are as small as possible. The method relies on and is a wrapper for the fullmatch function in the **optmatch** package.

Advantages of optimal full matching include that the matching order is not required to be specified, units do not need to be discarded, and it is less likely that extreme within-subclass distances will be large, unlike with standard subclassification. The primary output of full matching is a set of matching weights that can be applied to the matched sample; in this way, full matching can be seen as a robust alternative to propensity score weighting, robust in the sense that the propensity score model does not need to be correct to estimate the treatment effect without bias.

This page details the allowable arguments with `method = "fullmatch"`. See matchit for an explanation of what each argument means in a general context and how it can be specified.

Below is how `matchit` is used for optimal full matching:

```
matchit(formula, data = NULL, method = "full",
        distance = "glm", link = "logit",
        distance.options = list(), estimand = "ATT",
        exact = NULL, mahvars = NULL, discard = "none",
        reestimate = FALSE, s.weights = NULL,
        caliper = NULL, std.caliper = TRUE,
        verbose = FALSE, ...)
```

Arguments

`formula` a two-sided `formula` object containing the treatment and covariates to be used in creating the distance measure used in the matching. This formula will be supplied to the functions that estimate the distance measure.

`data` a data frame containing the variables named in `formula`. If not found in `data`, the variables will be sought in the environment.

`method` set here to `"full"`.

`distance` the distance measure to be used. See `distance` for allowable options. When set to `"mahalanobis"`, optimal full Mahalanobis distance matching will be performed on the variables named in `formula`.

`link` when `distance` is specified as a string and not `"mahalanobis"`, an additional argument controlling the link function used in estimating the distance measure. See `distance` for allowable options with each option.

`distance.options` a named list containing additional arguments supplied to the function that estimates the distance measure as determined by the argument to `distance`.

`estimand` a string containing the desired estimand. Allowable options include `"ATT"`, `"ATC"`, and `"ATE"`. The estimand controls how the weights are computed; see the Computing Weights section at `matchit` for details.

`exact` for which variables exact matching should take place. Exact matching is processed using `exactMatch` in **optmatch**.

`mahvars` for which variables Mahalanobis distance matching should take place when a distance measure other than `"mahalanobis"` is used (e.g., for caliper matching or to discard units for common support). If specified, the distance measure will not be used in matching.

`discard` a string containing a method for discarding units outside a region of common support. Only allowed when `distance` is not `"mahalanobis"`.

`reestimate` if `discard` is not `"none"`, whether to re-estimate the propensity score in the remaining sample prior to matching.

`s.weights` the variable containing sampling weights to be incorporated into propensity score models and balance statistics.

`caliper` the width(s) of the caliper(s) used for caliper matching. See Notes and Examples.

`std.caliper` `logical`; when calipers are specified, whether they are in standard deviation units (`TRUE`) or raw units (`FALSE`).

`verbose` `logical`; whether information about the matching process should be printed to the console.

`...` additional arguments passed to `fullmatch` in **optmatch**. Allowed arguments include `min.controls`, `max.controls`, `omit.fraction`, `mean.controls`, and `tol`. See the `fullmatch` documentation for details.

The arguments `replace`, `m.order`, and `ratio` are ignored with a warning.

Details

Mahalanobis Distance Matching: Mahalanobis distance matching can be done one of two ways:

1) If no propensity score needs to be estimated, `distance` should be set to `"mahalanobis"`, and Mahalanobis distance matching will occur on all the variables in `formula`. Arguments to `discard` and `mahvars` will be ignored, and a caliper can only be placed on named variables. For example, to perform simple Mahalanobis distance matching, the following could be run:

```
matchit(treat ~ X1 + X2, method = "nearest",
        distance = "mahalanobis")
```

With this code, the Mahalanobis distance is computed using `X1` and `X2`, and matching occurs on this distance. The `distance` component of the `matchit` output will be empty.

2) If a propensity score needs to be estimated for any reason, e.g., for common support with `discard` or for creating a caliper, `distance` should be whatever method is used to estimate the propensity score or a vector of distance measures, i.e., it should not be `"mahalanobis"`. Use `mahvars` to specify the variables used to create the Mahalanobis distance. For example, to perform Mahalanobis within a propensity score caliper, the following could be run:

```
matchit(treat ~ X1 + X2 + X3, method = "nearest",
        distance = "glm", caliper = .25,
        mahvars = ~ X1 + X2)
```

With this code, `X1`, `X2`, and `X3` are used to estimate the propensity score (using the `"glm"` method, which by default is logistic regression), which is used to create a matching caliper. The actual matching occurs on the Mahalanobis distance computed only using `X1` and `X2`, which are supplied to `mahvars`. Units whose propensity score difference is larger than the caliper will not be paired, and some treated units may therefore not receive a match. The estimated propensity scores will be included in the `distance` component of the `matchit` output. See Examples.

When sampling weights are supplied through the `s.weights` argument, the covariance matrix of the covariates used in the Mahalanobis distance is **not** weighted by the sampling weights.

Outputs

All outputs described in `matchit` are returned with `method = "full"` except for `match.matrix`. This is because matching strata are not indexed by treated units as they are in some other forms of matching.

Note

Due to what appears to be a bug in **optmatch** (version 0.9-13), calipers can only be used when `min.controls` is left at its default.

References

In a manuscript, be sure to cite the following paper if using `matchit` with `method = "full"`:

Hansen, B. B., & Klopfer, S. O. (2006). Optimal Full Matching and Related Designs via Network Flows. Journal of Computational and Graphical Statistics, 15(3), 609–627. doi: 10.1198/106186006X137047

For example, a sentence might read:

Optimal full matching was performed using the MatchIt package (Ho, Imai, King, & Stuart, 2011) in R, which calls functions from the optmatch package (Hansen & Klopfer, 2006).

Theory is also developed in the following article:

Hansen, B. B. (2004). Full Matching in an Observational Study of Coaching for the SAT. Journal of the American Statistical Association, 99(467), 609–618. doi: 10.1198/016214504000000647

See Also

`matchit` for a detailed explanation of the inputs and outputs of a call to `matchit`.

`fullmatch` in **optmatch**, which is the workhorse.

`method_optimal` for optimal pair matching, which is a special case of optimal full matching, and which relies on similar machinery. Results from `method = "optimal"` can be replicated with `method = "full"` by setting `min.controls`, `max.controls`, and `mean.controls` to the desired `ratio`.

Examples

```
data("lalonde")

# Optimal full PS matching
m.out1 <- matchit(treat ~ age + educ + race + nodegree +
                    married + re74 + re75, data = lalonde,
                  method = "full")
m.out1
summary(m.out1)

# Optimal full Mahalanobis distance matching within a PS caliper
m.out2 <- matchit(treat ~ age + educ + race + nodegree +
                    married + re74 + re75, data = lalonde,
                  method = "full", caliper = .01,
                  mahvars = ~ age + educ + re74 + re75)
m.out2
summary(m.out2)

# Optimal full Mahalanobis distance matching within calipers
# of 500 on re74 and re75
m.out3 <- matchit(treat ~ age + educ + re74 + re75,
                  data = lalonde, distance = "mahalanobis",
                  method = "full",
                  caliper = c(re74 = 500, re75 = 500),
                  std.caliper = FALSE)
m.out3
summary(m.out3, addlvariables = ~race + nodegree + married,
        data = lalonde)
```

`method_genetic` *Genetic Matching*

Description

In matchit, setting method = "genetic" performs genetic matching. Genetic matching is a form of nearest neighbor matching where distances are computed as the generalized Mahalanobis distance, which is a generalization of the Mahalanobis distance with a scaling factor for each covariate that represents the importance of that covariate to the distance. A genetic algorithm is used to select the scaling factors. The scaling factors are chosen as those which maximize a criterion related to covariate balance, which can be chosen, but which by default is the smallest p-value in covariate balance tests among the covariates. This method relies on and is a wrapper for the GenMatch and Match functions in the **Matching** package, which uses genoud from the **rgenoud** package to perform the optimization using the genetic algorithm.

This page details the allowable arguments with method = "genmatch". See matchit for an explanation of what each argument means in a general context and how it can be specified.

Below is how matchit is used for genetic matching:

```
matchit(formula, data = NULL, method = "genetic",
        distance = "glm", link = "logit",
        distance.options = list(), estimand = "ATT",
        exact = NULL, mahvars = NULL, discard = "none",
        reestimate = FALSE, s.weights = NULL,
        replace = FALSE, m.order = NULL,
        caliper = NULL, ratio = 1, verbose = FALSE,
        ...)
```

Arguments

formula a two-sided formula object containing the treatment and covariates to be used in creating the distance measure used in the matching. This formula will be supplied to the functions that estimate the distance measure and is used to determine the covariates whose balance is to be optimized.

data a data frame containing the variables named in formula. If not found in data, the variables will be sought in the environment.

method set here to "genetic".

distance the distance measure to be used. See distance for allowable options. When set to "mahalanobis", only the covariates in formula are supplied to the generalized Mahalanobis distance matrix to have their scaling factors chosen. Otherwise, the distance measure is included with the covariates in formula to be supplied to the generalized Mahalanobis distance matrix.

link when distance is specified as a string and not "mahalanobis", an additional argument controlling the link function used in estimating the distance measure. See distance for allowable options with each option.

distance.options

a named list containing additional arguments supplied to the function that estimates the distance measure as determined by the argument to distance.

`estimand` a string containing the desired estimand. Allowable options include `"ATT"` and `"ATC"`. See Details.

`exact` for which variables exact matching should take place.

`mahvars` when a distance measure other than `"mahalanobis"` is used (e.g., for caliper matching or to discard units for common support), which covariates should be supplied to the generalized Mahalanobis distance matrix. If unspecified, all variables in `formula` will be supplied to the distance matrix. Use `mahvars` to only supply a subset. Even if `mahvars` is specified, balance will be optimized on all covariates in `formula`.

`discard` a string containing a method for discarding units outside a region of common support. Only allowed when `distance` is not `"mahalanobis"`.

`reestimate` if `discard` is not `"none"`, whether to re-estimate the propensity score in the remaining sample prior to matching.

`s.weights` the variable containing sampling weights to be incorporated into propensity score models and balance statistics. These are also supplied to `GenMatch` for use in computing the balance t-test p-values in the process of matching.

`replace` whether matching should be done with replacement.

`m.order` the order that the matching takes place. The default for `distance = "mahalanobis"` is `"data"`. Otherwise, the default is `"largest"`. See `matchit` for allowable options.

`caliper` the width(s) of the caliper(s) used for caliper matching. See Details and Examples.

`std.caliper` `logical`; when calipers are specified, whether they are in standard deviation units (`TRUE`) or raw units (`FALSE`).

`ratio` how many control units should be matched to each treated unit for k:1 matching. Should be a single integer value.

`verbose` `logical`; whether information about the matching process should be printed to the console. When `TRUE`, output from `GenMatch` with `print.level = 2` will be displayed. Default is `FALSE` for no printing other than warnings.

`...` additional arguments passed to `GenMatch` in **Matching**. Potentially useful options include `pop.size`, `max.generations`, and `fit.func`. If `pop.size` is not specified, a warning from **Matching** will be thrown reminding you to change it. Note that the `ties` and `CommonSupport` arguments are set to `FALSE` and cannot be changed.

Details

In genetic matching, covariates play three roles: 1) as the variables on which balance is optimized, 2) as the variables in the generalized Mahalanobis distance between units, and 3) in estimating the propensity score. Variables supplied to `formula` are always used for role (1), as the variables on which balance is optimized. When `distance` is not `"mahalanobis"`, the covariates are also used to estimate the propensity score (unless it is supplied). When `mahvars` is specified, the named variables will form the covariates that go into the distance matrix. Otherwise, the variables in `formula` along with the propensity score will go into the distance matrix. This leads to three ways to use `distance` and `mahvars` to perform the matching:

1) When `distance = "mahalanobis"`, no propensity score is estimated, and the covariates in `formula` are used to form the generalized Mahalanobis distance matrix. In this sense,

"mahalanobis" signals that no propensity score is to be estimated and that the matching variables are those in formula, consistent with setting distance = "mahalanobis" with other methods.

2) When distance is not "mahalanobis" and mahvars *is not* specified, the covariates in formula along with the propensity score are used to form the generalized Mahalanobis distance matrix. This is the default and most typical use of method = "genetic" in matchit.

3) When distance is not "mahalanobis" and mahvars *is* specified, the covariates in mahvars are used to form the generalized Mahalanobis distance matrix. The covariates in formula are used to estimate the propensity score and have their balance optimized by the genetic algorithm. The propensity score is not included in the generalized Mahalanobis distance matrix.

When a caliper is specified, any variables mentioned in caliper, possibly including the propensity score, will be added to the matching variables used to form the generalized Mahalanobis distance matrix. This is because **Matching** doesn't allow for the separation of caliper variables and matching variables in genetic matching.

Estimand: The estimand argument controls whether control units are selected to be matched with treated units (estimand = "ATT") or treated units are selected to be matched with control units (estimand = "ATC"). The "focal" group (e.g., the treated units for the ATT) is typically made to be the smaller treatment group, and a warning will be thrown if it is not set that way unless replace = TRUE. Setting estimand = "ATC" is equivalent to swapping all treated and control labels for the treatment variable. When estimand = "ATC", the default m.order is "smallest", and the match.matrix component of the output will have the names of the control units as the rownames and be filled with the names of the matched treated units (opposite to when estimand = "ATT"). Note that the argument supplied to estimand doesn't necessarily correspond to the estimand actually targeted; it is merely a switch to trigger which treatment group is considered "focal". Note that while GenMatch() and Matching() support the ATE as an estimand, matchit() only supports the ATT and ATC for genetic matching.

Outputs

All outputs described in matchit are returned with method = "genetic". When repalce = TRUE, the subclass component is omitted.

References

In a manuscript, be sure to cite the following papers if using matchit with method = "genetic":

Diamond, A., & Sekhon, J. S. (2013). Genetic matching for estimating causal effects: A general multivariate matching method for achieving balance in observational studies. Review of Economics and Statistics, 95(3), 932–945. doi: 10.1162/REST_a_00318

Sekhon, J. S. (2011). Multivariate and Propensity Score Matching Software with Automated Balance Optimization: The Matching package for R. Journal of Statistical Software, 42(1), 1–52. doi: 10.18637/jss.v042.i07

For example, a sentence might read:

Genetic matching was performed using the MatchIt package (Ho, Imai, King, & Stuart, 2011) in R, which calls functions from the Matching package (Diamond & Sekhon, 2013; Sekhon, 2011).

See Also

matchit for a detailed explanation of the inputs and outputs of a call to matchit.

GenMatch and Match in **Matching**, which do the work.

Examples

```
data("lalonde")

# 1:1 genetic matching with PS as a covariate
m.out1 <- matchit(treat ~ age + educ + race + nodegree +
                    married + re74 + re75, data = lalonde,
                  method = "genetic",
                  pop.size = 10) #use much larger pop.size
m.out1
summary(m.out1)

# 2:1 genetic matching with replacement without PS
m.out2 <- matchit(treat ~ age + educ + race + nodegree +
                    married + re74 + re75, data = lalonde,
                  method = "genetic", replace = TRUE,
                  ratio = 2, distance = "mahalanobis",
                  pop.size = 10) #use much larger pop.size
m.out2
summary(m.out2)

# 1:1 genetic matching on just age, educ, re74, and re75
# within calipers on PS and educ; other variables are
# used to estimate PS
m.out3 <- matchit(treat ~ age + educ + race + nodegree +
                    married + re74 + re75, data = lalonde,
                  method = "genetic",
                  mahvars = ~ age + educ + re74 + re75,
                  caliper = c(.05, educ = 2),
                  std.caliper = c(TRUE, FALSE),
                  pop.size = 10) #use much larger pop.size
m.out3
summary(m.out3)
```

method_nearest *Nearest Neighbor Matching*

Description

In matchit, setting method = "nearest" performs greedy nearest neighbor matching. A distance is computed between each treated unit and each control unit, and, one by one, each treated unit is assigned a control unit as a match. The matching is ”greedy” in the sense that there is no action taken to optimize an overall criterion; each match is selected without considering the other matches that may occur subsequently.

This page details the allowable arguments with method = "nearest". See matchit for an explanation of what each argument means in a general context and how it can be specified.

Below is how matchit is used for nearest neighbor matching:

```
matchit(formula, data = NULL, method = "nearest",
        distance = "glm", link = "logit",
        distance.options = list(), estimand = "ATT",
        exact = NULL, mahvars = NULL, discard = "none",
        reestimate = FALSE, s.weights = NULL,
        replace = TRUE, m.order = NULL, caliper = NULL,
        ratio = 1, min.controls = NULL,
        max.controls = NULL, verbose = FALSE, ...)
```

Arguments

`formula` a two-sided `formula` object containing the treatment and covariates to be used in creating the distance measure used in the matching.

`data` a data frame containing the variables named in `formula`. If not found in `data`, the variables will be sought in the environment.

`method` set here to `"nearest"`.

`distance` the distance measure to be used. See `distance` for allowable options.

`link` when `distance` is specified as a string and not `"mahalanobis"`, an additional argument controlling the link function used in estimating the distance measure. See `distance` for allowable options with each option.

`distance.options`
a named list containing additional arguments supplied to the function that estimates the distance measure as determined by the argument to `distance`.

`estimand` a string containing the desired estimand. Allowable options include `"ATT"` and `"ATC"`. See Details.

`exact` for which variables exact matching should take place.

`mahvars` for which variables Mahalanobis distance matching should take place when a distance measure other than `"mahalanobis"` is used (e.g., for caliper matching or to discard units for common support). If specified, the distance measure will not be used in matching.

`discard` a string containing a method for discarding units outside a region of common support. Only allowed when `distance` is not `"mahalanobis"`.

`reestimate` if `discard` is not `"none"`, whether to re-estimate the propensity score in the remaining sample prior to matching.

`s.weights` the variable containing sampling weights to be incorporated into propensity score models and balance statistics.

`replace` whether matching should be done with replacement.

`m.order` the order that the matching takes place. The default for `distance = "mahalanobis"` is `"data"`. Otherwise, the default is `"largest"`. See `matchit` for allowable options.

`caliper` the width(s) of the caliper(s) used for caliper matching. See Details and Examples.

`std.caliper` `logical`; when calipers are specified, whether they are in standard deviation units (`TRUE`) or raw units (`FALSE`).

`ratio` how many control units should be matched to each treated unit for k:1 matching. For variable ratio matching, see section ”Variable Ratio Matching” in Details below.

`min.controls, max.controls`
: for variable ratio matching, the minimum and maximum number of controls units to be matched to each treated unit. See section "Variable Ratio Matching" in Details below.

`verbose`
: `logical`; whether information about the matching process should be printed to the console. When `TRUE`, a progress bar implemented using **RcppProgress** will be displayed.

`...`
: ignored.

Details

Mahalanobis Distance Matching: Mahalanobis distance matching can be done one of two ways:

1) If no propensity score needs to be estimated, `distance` should be set to `"mahalanobis"`, and Mahalanobis distance matching will occur on all the variables in `formula`. Arguments to `discard` and `mahvars` will be ignored, and a caliper can only be placed on named variables. For example, to perform simple Mahalanobis distance matching, the following could be run:

```
matchit(treat ~ X1 + X2, method = "nearest",
        distance = "mahalanobis")
```

With this code, the Mahalanobis distance is computed using `X1` and `X2`, and matching occurs on this distance. The `distance` component of the `matchit` output will be empty.

2) If a propensity score needs to be estimated for any reason, e.g., for common support with `discard` or for creating a caliper, `distance` should be whatever method is used to estimate the propensity score or a vector of distance measures, i.e., it should not be `"mahalanobis"`. Use `mahvars` to specify the variables used to create the Mahalanobis distance. For example, to perform Mahalanobis within a propensity score caliper, the following could be run:

```
matchit(treat ~ X1 + X2 + X3, method = "nearest",
        distance = "glm", caliper = .25,
        mahvars = ~ X1 + X2)
```

With this code, `X1`, `X2`, and `X3` are used to estimate the propensity score (using the `"glm"` method, which by default is logistic regression), which is used to create a matching caliper. The actual matching occurs on the Mahalanobis distance computed only using `X1` and `X2`, which are supplied to `mahvars`. Units whose propensity score difference is larger than the caliper will not be paired, and some treated units may therefore not receive a match. The estimated propensity scores will be included in the `distance` component of the `matchit` output. See Examples.

When sampling weights are supplied through the `s.weights` argument, the covariance matrix of the covariates used in the Mahalanobis distance is weighted by the sampling weights.

Estimand: The `estimand` argument controls whether control units are selected to be matched with treated units (`estimand = "ATT"`) or treated units are selected to be matched with control units (`estimand = "ATC"`). The "focal" group (e.g., the treated units for the ATT) is typically made to be the smaller treatment group, and a warning will be thrown if it is not set that way unless `replace = TRUE`. Setting `estimand = "ATC"` is equivalent to swapping all treated and control labels for the treatment variable. When `estimand = "ATC"`, the default `m.order` is `"smallest"`, and the `match.matrix` component of the output will have the names of the control units as the rownames and be filled with the names of the matched treated units (opposite to when `estimand = "ATT"`). Note that the argument supplied to `estimand` doesn't necessarily correspond to the estimand actually targeted; it is merely a switch to trigger which treatment group is considered "focal".

Variable Ratio Matching: `matchit` can perform variable ratio "extremal" matching as described by Ming and Rosenbaum (2000). This method tends to result in better balance than fixed ratio matching at the expense of some precision. When `ratio > 1`, rather than requiring all treated units to receive `ratio` matches, each treated unit is assigned a value that corresponds to the number of control units they will be matched to. These values are controlled by the arguments `min.controls` and `max.controls`, which correspond to α and β, respectively, in Ming and Rosenbaum (2000), and trigger variable ratio matching to occur. Some treated units will receive `min.controls` matches and others will receive `max.controls` matches (and one unit may have an intermediate number of matches); how many units are assigned each number of matches is determined by the algorithm described in Ming and Rosenbaum (2000, p119). `ratio` controls how many total control units will be matched: `n1 * ratio` control units will be matched, where `n1` is the number of treated units, yielding the same total number of matched controls as fixed ratio matching does.

Variable ratio matching cannot be used with Mahalanobis distance matching. The calculations of the numbers of control units each treated unit will be matched to occurs without consideration of `caliper` or `discard`. `ratio` does not have to be an integer but must be greater than 1 and less than `n0/n1`, where `n0` and `n1` are the number of control and treated units, respectively. Setting `ratio = n0/n1` performs a crude form of full matching where all control units are matched. If `min.controls` is not specified, it is set to 1 by default. `min.controls` must be less than `ratio` and `max.controls` must be greater than `ratio`. See Examples below for an example their use.

Outputs

All outputs described in `matchit` are returned with `method = "nearest"`. When `replace = TRUE`, the `subclass` component is omitted.

References

In a manuscript, you don't need to cite another package when using `method = "nearest"` because the matching is performed completely within **MatchIt**. For example, a sentence might read:

Nearest neighbor matching was performed using the MatchIt package (Ho, Imai, King, & Stuart, 2011) in R.

See Also

`matchit` for a detailed explanation of the inputs and outputs of a call to `matchit`.

`method_optimal` for optimal pair matching, which is similar to nearest neighbor matching except that an overall distance criterion is minimized.

Examples

```
data("lalonde")

# 1:1 greedy NN matching on the PS
m.out1 <- matchit(treat ~ age + educ + race + nodegree +
                    married + re74 + re75, data = lalonde,
                  method = "nearest")
m.out1
```

```
summary(m.out1)

# 3:1 NN Mahalanobis distance matching with
# replacement within a PS caliper
m.out2 <- matchit(treat ~ age + educ + race + nodegree +
                    married + re74 + re75, data = lalonde,
                  method = "nearest", replace = TRUE,
                  mahvars = ~ age + educ + re74 + re75,
                  ratio = 3, caliper = .02)
m.out2
summary(m.out2)

# 1:1 NN Mahalanobis distance matching within calipers
# on re74 and re75 and exact matching on married and race
m.out3 <- matchit(treat ~ age + educ + re74 + re75, data = lalonde,
                  method = "nearest", distance = "mahalanobis",
                  exact = ~ married + race,
                  caliper = c(re74 = .2, re75 = .15))
m.out3
summary(m.out3)

# 2:1 variable ratio NN matching on the PS
m.out4 <- matchit(treat ~ age + educ + race + nodegree +
                    married + re74 + re75, data = lalonde,
                  method = "nearest", ratio = 2,
                  min.controls = 1, max.controls = 12)
m.out4
summary(m.out4)

# Some units received 1 match and some received 12
table(table(m.out4$subclass[m.out4$treat == 0]))
```

method_optimal *Optimal Pair Matching*

Description

In matchit, setting method = "optimal" performs optimal pair matching. The matching is optimal in the sense that that sum of the absolute pairwise distances in the matched sample are as small as possible. The method relies on and is a wrapper for the pairmatch function in the **optmatch** package. Advantages of optimal pair matching include that the matching order is not required to be specified and it is less likely that extreme within-pair distances will be large, unlike with nearest neighbor matching. Generally, however, as a subset selection method, optimal pair matching tends to perform similarly to nearest neighbor matching in that similar subsets of units will be selected to be matched.

This page details the allowable arguments with method = "optmatch". See matchit for an explanation of what each argument means in a general context and how it can be specified.

Below is how matchit is used for optimal pair matching:

```
matchit(formula, data = NULL, method = "optimal",
        distance = "glm", link = "logit",
        distance.options = list(), estimand = "ATT",
        exact = NULL, mahvars = NULL, discard = "none",
        reestimate = FALSE, s.weights = NULL,
        ratio = 1, verbose = FALSE, ...)
```

Arguments

`formula` a two-sided `formula` object containing the treatment and covariates to be used in creating the distance measure used in the matching. This formula will be supplied to the functions that estimate the distance measure.

`data` a data frame containing the variables named in `formula`. If not found in `data`, the variables will be sought in the environment.

`method` set here to `"optimal"`.

`distance` the distance measure to be used. See `distance` for allowable options. When set to `"mahalanobis"`, optimal Mahalanobis distance matching will be performed on the variables named in `formula`.

`link` when `distance` is specified as a string and not `"mahalanobis"`, an additional argument controlling the link function used in estimating the distance measure. See `distance` for allowable options with each option.

`distance.options` a named list containing additional arguments supplied to the function that estimates the distance measure as determined by the argument to `distance`.

`estimand` a string containing the desired estimand. Allowable options include `"ATT"` and `"ATC"`. See Details.

`exact` for which variables exact matching should take place. Exact matching is processed using `exactMatch` in **optmatch**.

`mahvars` for which variables Mahalanobis distance matching should take place when a distance measure other than `"mahalanobis"` is used (e.g., to discard units for common support). If specified, the distance measure will not be used in matching.

`discard` a string containing a method for discarding units outside a region of common support. Only allowed when `distance` is not `"mahalanobis"`.

`reestimate` if `discard` is not `"none"`, whether to re-estimate the propensity score in the remaining sample prior to matching.

`s.weights` the variable containing sampling weights to be incorporated into propensity score models and balance statistics.

`ratio` how many control units should be matched to each treated unit for k:1 matching. Should be a single integer value.

`verbose` `logical`; whether information about the matching process should be printed to the console. What is printed depends on the matching method. Default is `FALSE` for no printing other than warnings.

`...` additional arguments passed to `pairmatch` in **optmatch**. Only the `tols` argument, which is eventually passed to `fullmatch`, is allowed.

The arguments `replace`, `caliper`, and `m.order` are ignored with a warning.

Details

Mahalanobis Distance Matching: Mahalanobis distance matching can be done one of two ways:

1) If no propensity score needs to be estimated, `distance` should be set to `"mahalanobis"`, and Mahalanobis distance matching will occur on all the variables in `formula`. Arguments to `discard` and `mahvars` will be ignored. For example, to perform simple Mahalanobis distance matching, the following could be run:

```
matchit(treat ~ X1 + X2, method = "nearest",
        distance = "mahalanobis")
```

With this code, the Mahalanobis distance is computed using `X1` and `X2`, and matching occurs on this distance. The `distance` component of the `matchit` output will be empty.

2) If a propensity score needs to be estimated for common support with `discard`, `distance` should be whatever method is used to estimate the propensity score or a vector of distance measures, i.e., it should not be `"mahalanobis"`. Use `mahvars` to specify the variables used to create the Mahalanobis distance. For example, to perform Mahalanobis after discarding units outside the common support of the propensity score in both groups, the following could be run:

```
matchit(treat ~ X1 + X2 + X3, method = "nearest",
        distance = "glm", discard = "both",
        mahvars = ~ X1 + X2)
```

With this code, `X1`, `X2`, and `X3` are used to estimate the propensity score (using the `"glm"` method, which by default is logistic regression), which is used to identify the common support. The actual matching occurs on the Mahalanobis distance computed only using `X1` and `X2`, which are supplied to `mahvars`. The estimated propensity scores will be included in the `distance` component of the `matchit` output.

When sampling weights are supplied through the `s.weights` argument, the covariance matrix of the covariates used in the Mahalanobis distance is **not** weighted by the sampling weights.

Estimand: The `estimand` argument controls whether control units are selected to be matched with treated units (`estimand = "ATT"`) or treated units are selected to be matched with control units (`estimand = "ATC"`). The "focal" group (e.g., the treated units for the ATT) is typically made to be the smaller treatment group, and a warning will be thrown if it is not set that way unless `replace = TRUE`. Setting `estimand = "ATC"` is equivalent to swapping all treated and control labels for the treatment variable. When `estimand = "ATC"`, the `match.matrix` component of the output will have the names of the control units as the rownames and be filled with the names of the matched treated units (opposite to when `estimand = "ATT"`). Note that the argument supplied to `estimand` doesn't necessarily correspond to the estimand actually targeted; it is merely a switch to trigger which treatment group is considered "focal".

Outputs

All outputs described in `matchit` are returned with `method = "optimal"`.

Note

Calipers may eventually be compatible with this method, but due to what appears to be a bug in **optmatch** (version 0.9-13), they are currently disabled.

References

In a manuscript, be sure to cite the following paper if using `matchit` with `method = "optimal"`:

Hansen, B. B., & Klopfer, S. O. (2006). Optimal Full Matching and Related Designs via Network Flows. Journal of Computational and Graphical Statistics, 15(3), 609–627. doi: 10.1198/106186006X137047

For example, a sentence might read:

Optimal pair matching was performed using the MatchIt package (Ho, Imai, King, & Stuart, 2011) in R, which calls functions from the optmatch package (Hansen & Klopfer, 2006).

See Also

`matchit` for a detailed explanation of the inputs and outputs of a call to `matchit`.

`pairmatch` in **optmatch**, which is the workhorse.

`method_full` for optimal full matching, of which optimal pair matching is a special case, and which relies on similar machinery.

Examples

```
data("lalonde")

# 1:1 optimal PS matching with exact matching on race
m.out1 <- matchit(treat ~ age + educ + race + nodegree +
                    married + re74 + re75, data = lalonde,
                  method = "optimal", exact = ~race)
m.out1
summary(m.out1)

#2:1 optimal Mahalanobis distance matching
m.out2 <- matchit(treat ~ age + educ + race + nodegree +
                    married + re74 + re75, data = lalonde,
                  method = "optimal", distance = "mahalanobis",
                  ratio = 2)
m.out2
summary(m.out2)
```

`method_subclass` *Subclassification*

Description

In `matchit`, setting `method = "subclass"` performs subclassification on the distance measure (i.e., propensity score). Treatment and control units are placed into subclasses based on quantiles of the propensity score in the treated group, in the control group, or overall, depending on the desired estimand. Weights are computed based on the proportion of treated units in each subclass. Subclassification implemented here does not rely on any other package.

This page details the allowable arguments with `method = "subclass"`. See `matchit` for an explanation of what each argument means in a general context and how it can be specified.

Below is how `matchit` is used for subclassification:

```
matchit(formula, data = NULL, method = "subclass",
        distance = "glm", link = "logit",
        distance.options = list(), estimand = "ATT",
        discard = "none", reestimate = FALSE,
        s.weights = NULL, verbose = FALSE, ...)
```

Arguments

`formula` a two-sided `formula` object containing the treatment and covariates to be used in creating the distance measure used in the subclassification.

`data` a data frame containing the variables named in `formula`. If not found in `data`, the variables will be sought in the environment.

`method` set here to `"subclass"`.

`distance` the distance measure to be used. See `distance` for allowable options. `distance = "mahalanobis"` is not allowed.

`link` when `distance` is specified as a string, an additional argument controlling the link function used in estimating the distance measure. See `distance` for allowable options with each option.

`distance.options` a named list containing additional arguments supplied to the function that estimates the distance measure as determined by the argument to `distance`.

`estimand` the target `estimand`. If `"ATT"`, the default, subclasses are formed based on quantiles of the distance measure in the treated group; if `"ATC"`, subclasses are formed based on quantiles of the distance measure in the control group; if `"ATE"`, subclasses are formed based on quantiles of the distance measure in the full sample. The estimand also controls how the subclassification weights are computed; see the Computing Weights section at `matchit` for details.

`discard` a string containing a method for discarding units outside a region of common support.

`reestimate` if `discard` is not `"none"`, whether to re-estimate the propensity score in the remaining sample prior to subclassification.

`s.weights` the variable containing sampling weights to be incorporated into propensity score models and balance statistics.

`verbose` `logical`; whether information about the matching process should be printed to the console.

`...` additional arguments that control the subclassification:

`subclass` either the number of subclasses desired or a vector of quantiles used to divide the distance measure into subclasses. Default is 6.

`min.n` the minimum number of units of each treatment group that are to be assigned each subclass. If the distance measure is divided in such a way that fewer than `min.n` units of a treatment group are assigned a given subclass, units from other subclasses will be reassigned to fill the deficient subclass. Default is 1.

The arguments `exact`, `mahvars`, `replace`, `m.order`, `caliper` (and related arguments), and `ratio` are ignored with a warning.

Details

After subclassification, effect estimates can be computed separately in the subclasses and combined, or a single marginal effect can be estimated by using the weights in the full sample. When using the weights, the method is sometimes referred to as marginal mean weighting through stratification (MMWS; Hong, 2010) or fine stratification weighting (Desai et al., 2017). The weights can be interpreted just like inverse probability weights.

Changing `min.n` can change the quality of the weights. Generally, a low `min.w` will yield better balance because subclasses only contain units with relatively similar distance values, but may yield higher variance because extreme weights can occur due to there being few members of a treatment group in some subclasses.

Note that subclassification weights can also be estimated using **WeightIt**, which provides some additional methods for estimating propensity scores. Where propensity score-estimation methods overlap, both packages will yield the same weights.

Outputs

All outputs described in `matchit` are returned with `method = "subclass"` except that `match.matrix` is excluded and two additional component, `q.cut` and `qn`, are included, containing a vector of the distance measure cutpoints used to define the subclasses and a matrix of the subclass sample sizes, respectively. Note that when `min.n > 0`, the subclass assignments may not strictly obey the quantiles listed in `q.cut`.

References

In a manuscript, you don't need to cite another package when using `method = "subclass"` because the subclassification is performed completely within **MatchIt**. For example, a sentence might read:

Propensity score subclassification was performed using the MatchIt package (Ho, Imai, King, & Stuart, 2011) in R.

It may be a good idea to cite Hong (2010) or Desai et al. (2017) if the treatment effect is estimated using the subclassification weights.

Desai, R. J., Rothman, K. J., Bateman, B. . T., Hernandez-Diaz, S., & Huybrechts, K. F. (2017). A Propensity-score-based Fine Stratification Approach for Confounding Adjustment When Exposure Is Infrequent: Epidemiology, 28(2), 249–257. doi: 10.1097/EDE.0000000000000595

Hong, G. (2010). Marginal mean weighting through stratification: Adjustment for selection bias in multilevel data. Journal of Educational and Behavioral Statistics, 35(5), 499–531. doi: 10.3102/1076998609359785

See Also

`matchit` for a detailed explanation of the inputs and outputs of a call to `matchit`.

`method_full` for optimal full matching, which is similar to subclassification except that the number of subclasses and subclass membership are chosen to optimize the within-subclass distance.

Examples

```
data("lalonde")

# PS subclassification for the ATT with 7 subclasses
```

```
s.out1 <- matchit(treat ~ age + educ + race + nodegree +
                    married + re74 + re75, data = lalonde,
                  method = "subclass", subclass = 7)
s.out1
summary(s.out1, subclass = TRUE)

# PS subclassification for the ATE with 10 subclasses
# and at least 2 units in each group per subclass
s.out2 <- matchit(treat ~ age + educ + race + nodegree +
                    married + re74 + re75, data = lalonde,
                  method = "subclass", subclass = 10,
                  estimand = "ATE", min.n = 2)
s.out2
summary(s.out2)
```

plot.matchit — *Generate Balance Plots after Matching and Subclassification*

Description

Generates plots displaying distributional balance and overlap on covariates and propensity scores before and after matching and subclassification. For displaying balance solely on covariate standardized mean differences, see `plot.summary.matchit`. The plots here can be used to assess to what degree covariate and propensity score distributions are balanced and how weighting and discarding affect the distribution of propensity scores.

Usage

```
## S3 method for class 'matchit'
plot(x, type = "qq", interactive = TRUE,
     which.xs = NULL, ...)

## S3 method for class 'matchit.subclass'
plot(x, type = "qq", interactive = TRUE,
     which.xs = NULL, subclass, ...)
```

Arguments

`x`	a `matchit` object; the output of a call to `matchit`.
`type`	the type of plot to display. Options include `"qq"`, `"ecdf"`, `"jitter"`, and `"histogram"`. See Details. Default is `"qq"`. Abbreviations allowed.
`interactive`	`logical`; whether the graphs should be displayed in an interactive way. Only applies for `type = "qq"`, `"ecdf"`, and `"jitter"`. See Details.
`which.xs`	with `type = "qq"` or `"ecdf"`, for which covariate(s) plots should be displayed. Factor variables should be named by the original variable name rather than the names of individual dummy variables created after expansion with `model.matrix`.
`subclass`	with subclassification and `type = "qq"` or `"ecdf"`, whether to display balance for individual subclasses, and, if so, for which ones. Can be `TRUE` (display plots for all subclasses), `FALSE` (display plots only in aggregate), or the indices (e.g., `1:6`) of the specific subclasses for which to display

balance. When unspecified, if `interactive = TRUE`, you will be asked for which subclasses plots are desired, and otherwise, plots will be displayed only in aggregate.

`...` arguments passed to `plot` to control the appearance of the plot. Not all options are accepted.

Details

`plot.matchit` makes one of three different plots depending on the argument supplied to `type`.

With `type = "qq"`, empirical quantile-quantile (eQQ) plots are created for each covariate before and after matching. The plots involve interpolating points in the smaller group based on the weighted quantiles of the other group. When points are approximately on the 45-degree line, the distributions in the treatment and control groups are approximately equal. Major deviations indicate departures from distributional balance. With variable with fewer than 5 unique values, points are jittered to more easily visualize counts. When `interactive = TRUE`, plots for three variables will be displayed at a time, and the prompt in the console allows you to move on to the next set of variables. When `interactive = FALSE`, multiple pages are plotted at the same time, but only the last few variables will be visible in the displayed plot. To see only a few specific variables at a time, use the `which.xs` argument to display eQQ plots for just those variables. If fewer than three (after expanding factors into their dummies), `interactive` is ignored.

With `type = "ecdf"`, empirical cumulative density function (eCDF) plots are created for each covariate before and after matching. Two eCDF lines are produced in each plot: a gray one for control units and a black one for treated units. Each point on the lines corresponds to the proportion of units (or proportionate share of weights) less than or equal to the corresponding covariate value (on the x-axis). Deviations between the lines on the same plot indicates distributional imbalance between the treatment groups for the covariate. The eCDF and eQQ statistics in `summary.matchit` correspond to these plots: the eCDF max (also known as the Kolmogorov-Smirnov statistic) and mean are the largest and average vertical distance between the lines, and the eQQ max and mean are the largest and average horizontal distance between the lines.

With `type = "jitter"`, a jitter plot is displayed for distance values before and after matching. This method requires a distance variable (e.g., a propensity score) to have been estimated or supplied in the call to `matchit`. The plot displays individuals values for matched and unmatched treatment and control units arranged horizontally by their propensity scores. Points are jitter so counts are easier to see. The size of the points increases when they receive higher weights. When `interactive = TRUE`, you can click on points in the graph to identify their rownames and indices to further probe extreme values, for example. With subclassification, vertical lines representing the subclass boundaries are overlay on the plots.

With `type = "histogram"`, a histogram of distance values is displayed for the treatment and control groups before and after matching. This method requires a distance variable (e.g., a propensity score) to have been estimated or supplied in the call to `matchit`. With subclassification, vertical lines representing the subclass boundaries are overlay on the plots.

With all methods, sampling weights are incorporated into the weights if present.

Value

A plot is displayed, and `x` is invisibly returned.

Note

Sometimes, bugs in the plotting functions can cause strange layout or size issues. Running `frame()` or `dev.off()` can be used to reset the plotting pane (note the latter will delete any plots in the plot history).

See Also

`summary.matchit` for numerical summaries of balance, including those that rely on the eQQ and eCDF plots.

`plot.summary.matchit` for plotting standardized mean differences in a Love plot.

`bal.plot` in the **cobalt** package for displaying distributional balance in several other ways that are more easily customizable and produce **ggplot2** objects. **cobalt** functions natively support `matchit` objects.

Examples

```
data("lalonde")
m.out <- matchit(treat ~ age + educ + married +
                   race + re74, data = lalonde,
                 method = "nearest")
plot(m.out, type = "qq", interactive = FALSE,
     which.xs = c("age", "educ", "married"))
plot(m.out, type = "histogram")

s.out <- matchit(treat ~ age + educ + married +
                   race + nodegree + re74 + re75,
                 data = lalonde, method = "subclass")
plot(s.out, type = "ecdf", interactive = FALSE,
     which.xs = c("age", "educ", "married"),
     subclass = 3)
plot(s.out, type = "jitter", interactive = FALSE)
```

`plot.summary.matchit` *Generate a Love Plot of Standardized Mean Differences*

Description

Generates a Love plot, which is a dot plot with variable names on the y-axis and standardized mean differences on the x-axis. Each point represents the standardized mean difference of the corresponding covariate in the matched or unmatched sample. Love plots are a simple way to display covariate balance before and after matching. The plots are generated using `dotchart` and `points`.

Usage

```
## S3 method for class 'summary.matchit'
plot(x, abs = TRUE, var.order = "data",
       threshold = c(.1, .05), position = "bottomright", ...)
```

Arguments

x
: a summary.matchit object; the output of a call to summary.matchit. The standardize argument must be set to TRUE (which is the default) in the call to summary.

abs
: logical; whether the standardized mean differences should be displayed in absolute value (TRUE, default) or not FALSE.

var.order
: how the variables should be ordered. Allowable options include "data", ordering the variables as they appear in the summary output; "unmatched", ordered the variables based on their standardized mean differences before matching; "matched", ordered the variables based on their standardized mean differences after matching; and "alphabetical", ordering the variables alphabetically. Default is "data". Abbreviations allowed.

threshold
: numeric values at which to place vertical lines indicating a balance threshold. These can make it easier to see for which variables balance has been achieved given a threshold. Multiple values can be supplied to add multiple lines. When abs = FALSE, the lines will be displayed on both sides of zero. The lines are drawn with abline with the linetype (lty) argument corresponding to the order of the entered variables (see options at par). The default is c(.1,.05) for a solid line (lty = 1) at .1 and a dashed line (lty = 2) at .05, indicating acceptable and good balance, respectively. Enter a value as NA to skip that value of lty (e.g., c(NA,.05) to have only a dashed vertical line at .05).

position
: the position of the legend. Should be one of the allowed keyword options supplied to x in legend (e.g., "right", "bottomright", etc.). Default is "bottomright". Set to NULL for no legend to be included. Note that the legend will cover up points if you are not careful; setting var.order appropriately can help in avoiding this.

...
: ignored.

Details

For matching methods other than subclassification, plot.summary.matchit uses x$sum.all[,"Std. Mean Diff."] and x$sum.matched[,"Std. Mean Diff."] as the x-axis values. For subclassification, in addition to points for the unadjusted and aggregate subclass balance, numerals representing balance in individual subclasses are plotted if subclass = TRUE in the call to summary. Aggregate subclass standardized mean differences are taken from x$sum.across[,"Std. Mean Diff."] and the subclass-specific mean differences are taken from x$sum.subclass.

Value

A plot is displayed, and x is invisibly returned.

Author(s)

Noah Greifer

See Also

summary.matchit, dotchart

\pkgfuncobaltlove.plot in the **cobalt** package is a more flexible and sophisticated function to make Love plots and is also natively compatible with matchit objects.

Examples

```
data("lalonde")
m.out <- matchit(treat ~ age + educ + married +
                   race + re74, data = lalonde,
                 method = "nearest")
plot(summary(m.out, interactions = TRUE),
     var.order = "unmatched")

s.out <- matchit(treat ~ age + educ + married +
                   race + nodegree + re74 + re75,
                 data = lalonde, method = "subclass")
plot(summary(s.out, subclass = TRUE),
     var.order = "unmatched", abs = FALSE)
```

---

summary.matchit *View a balance summary of a* matchit *object*

Description

Computes and prints balance statistics for matchit and matchit.subclass objects. Balance should be assessed to ensure the matching or subclassification was effective at eliminating treatment group imbalance and should be reported in the write-up of the results of the analysis.

Usage

```
## S3 method for class 'matchit'
summary(object, interactions = FALSE,
        addlvariables = NULL, standardize = TRUE,
        data = NULL, pair.dist = TRUE, un = TRUE,
        improvement = TRUE, ...)
## S3 method for class 'matchit.subclass'
summary(object, interactions = FALSE,
        addlvariables = NULL, standardize = TRUE,
        data = NULL, pair.dist = FALSE, subclass = FALSE,
        un = TRUE, improvement = TRUE, ...)

## S3 method for class 'summary.matchit'
print(x, digits = max(3, getOption("digits") - 3),
      ...)
## S3 method for class 'summary.matchit.subclass'
print(x, digits = max(3, getOption("digits") - 3),
      ...)
```

Arguments

object a matchit object; the output of a call to matchit.

interactions logical; whether to compute balance statistics for two-way interactions and squares of covariates. Default is FALSE.

`addlvariables` additional variable for which balance statistics are to be computed along with the covariates in the `matchit` object. Can be entered in one of three ways: as a data frame of covariates with as many rows as there were units in the original `matchit` call, as a string containing the names of variables in `data`, or as a right-sided `formula` with the additional variables (and possibly their transformations) found in `data`, the environment, or the `matchit` object. Balance on squares and interactions of the additional variables will be included if `interactions = TRUE`.

`standardize` `logical`; whether to compute standardized (`TRUE`) or unstandardized (`FALSE`) statistics. The standardized statistics are the standardized mean difference and the median, mean, and maximum of the difference in the (weighted) empirical cumulative distribution functions (ECDFs). The unstandardized statistics are the raw mean difference and the median, mean, and maximum of the quantile-quantile (QQ) difference. See Details below. Default is `TRUE`.

`data` a optional data frame containing variables named in `addlvariables` if specified as a string or formula.

`pair.dist` `logical`; whether to compute average absolute pair distances. For matching methods that don't include a `match.matrix` component in the output (i.e., exact matching, coarsened exact matching, full matching, and subclassification), computing pair differences can take a long time, especially for large datasets and with many covariates. For other methods (i.e., nearest neighbor, optimal, and genetic matching), computation is fairly quick. Default is `FALSE` for subclassification and `TRUE` otherwise.

`un` `logical`; whether to compute balance statistics for the unmatched sample. Default `TRUE`; set to `FALSE` for more concise output.

`improvement` `logical`; whether to compute the percent reduction in imbalance. Default `TRUE`; set to `FALSE` for more concise output.

`subclass` after subclassification, whether to display balance for individual subclasses, and, if so, for which ones. Can be `TRUE` (display balance for all subclasses), `FALSE` (display balance only in aggregate), or the indices (e.g., `1:6`) of the specific subclasses for which to display balance. When anything other than `FALSE`, aggregate balance statistics will not be displayed. Default is `FALSE`.

`digits` the number of digits to round balance statistics to.

`x` a `summay.matchit` or `summary.matchit.subclass` object; the output of a call to `summary`.

`...` ignored.

Details

`summary` computes a balance summary of a `matchit` object. This include balance before and after matching or subclassification, as well as the percent improvement in balance. The variables for which balance statistics are computed are those included in the `formula`, `exact`, and `mahvars` arguments to `matchit`, as well as the distance measure if `distance` is not `"mahalanobis"`. The `X` component of the `matchit` object is used to supply the covariates.

The standardized mean differences are computed both before and after matching or subclassification as the difference in treatment group means divided by a standardization factor computed in the unmatched (original) sample. The standardization factor depends on the argument supplied to `estimand` in `matchit`: for `"ATT"`, it is the standard deviation in the treated group; for "ATC", it is the standard deviation in the control group; for "ATE", it is the square root of the average of the variances within each treatment group. The post-matching mean difference is computed with weighted means in the treatment groups using the matching or subclassification weights.

The variance ratio is computed as the ratio of the treatment group variances. Variance ratios are not computed for binary variables because their variance is a function solely of their mean. After matching, weighted variances are computed using the formula used in cov.wt. The percent reduction in bias is computed using the log of the variance ratios.

The eCDF difference statistics are computed by creating a (weighted) eCDF for each group and taking the difference between them for each covariate value. The eCDF is a function that outputs the (weighted) proportion of units with covariate values at or lower than the input value. The maximum eCDF difference is the same thing as the Kolmogorov-Smirnov statistic. The values are bounded at zero and one, with values closer to zero indicating good overlap between the covariate distributions in the treated and control groups. For binary variables, all eCDF differences are equal to the (weighted) difference in proportion and are computed that way.

The QQ difference statistics are computed by creating two samples of the same size by interpolating the values of the larger one. The values are arranged in order for each sample. The QQ difference for each quantile is the difference between the observed covariate values at that quantile between the two groups. The difference is on the scale of the original covariate. Values close to zero indicate good overlap between the covariate distributions in the treated and control groups. A weighted interpolation is used for post-matching QQ differences. For binary variables, all QQ differences are equal to the (weighted) difference in proportion and are computed that way.

The pair distance is the average of the absolute differences of a variable between pairs. For example, if a treated unit was paired with four control units, that set of units would contribute four absolute differences to the average. Within a subclass, each combination of treated and control unit forms a pair that contributes once to the average. The pair distance is described in Stuart and Green (2008) and is the value that is minimized when using optimal (full) matching. When standardize = TRUE, the standardized versions of the variables are used, where the standardization factor is as described above for the standardized mean differences. Pair distances are not computed in the unmatched sample (because there are no pairs). Because pair distance can take a while to compute, especially with large datasets or for many covariates, setting pair.dist = FALSE is one way to speed up summary.

The effective sample size (ESS) is a measure of the size of a hypothetical unweighted sample with roughly the same precision as a weighted sample. When non-uniform matching weights are computed (e.g., as a result of full matching, matching with replacement, or subclassification), the ESS can be used to quantify the potential precision remaining in the matched sample. The ESS will always be less than or equal to the matched sample size, reflecting the loss in precision due to using the weights. With non-uniform weights, it is printed in the sample size table; otherwise, it is removed because it does not contain additional information above the matched sample size.

After subclassification, the aggregate balance statistics are computed using the subclassification weights rather than averaging across subclasses.

All balance statistics (except pair differences) are computed incorporating the sampling weights upplied to matchit, if any. The unadjusted balance statistics include the sampling weights and the adjusted balance statistics use the matching weights multiplied by the sampling weights.

When printing, NA values are replaced with periods (.), and the pair distance column in the unmatched and percent balance improvement components of the output are omitted.

Value

For matchit objects, a summary.matchit object, which is a list with the following components:

`call` the original call to matchit

`nn` a matrix of the sample sizes in the original (unmatched) and matched samples

`sum.all` if un = TRUE, a matrix of balance statistics for each covariate in the original (unmatched) sample

`sum.matched` a matrix of balance statistics for each covariate in the matched sample

`reduction` if improvement = TRUE, a matrix of the percent reduction in imbalance for each covariate in the matched sample

For match.subclass objects, a summary.matchit.subclass object, which is a list as above with the following additional components:

`call` the original call to matchit

`sum.all` if un = TRUE, a matrix of balance statistics for each covariate in the original sample

`sum.subclass` if subclass is not FALSE, a list of matrices of balance statistics for each subclass

`sum.across` a matrix of balance statistics for each covariate computed using the subclassification weights

`reduction` if improvement = TRUE, a matrix of the percent reduction in imbalance for each covariate in the matched sample

`qn` a matrix of sample sizes within each subclass

`nn` a matrix of the sample sizes in the original (unmatched) and matched samples

See Also

summary for the generic method; plot.summary.matchit for making a Love plot from summary output.

bal.tab in the **cobalt** package, which also displays balance for matchit objects.

Examples

```
data("lalonde")
m.out <- matchit(treat ~ age + educ + married +
                   race + re74, data = lalonde,
                 method = "nearest", exact = ~ married,
                 replace = TRUE)
summary(m.out, interactions = TRUE)

s.out <- matchit(treat ~ age + educ + married +
                   race + nodegree + re74 + re75,
                 data = lalonde, method = "subclass")
summary(s.out, addlvariables = ~log(age) + I(re74==0))
summary(s.out, subclass = TRUE)
```