

Package ‘optmatch’

September 28, 2009

Version 0.6-0

Date 2009-09-25

Title Functions for optimal matching

Author Ben B. Hansen <ben.hansen@umich.edu> and Mark Fredrickson
<mark.m.fredrickson@gmail.com>, with embedded Fortran code due to Dimitri P. Bertsekas
<dimitrib@mit.edu> and Paul Tseng

Maintainer Ben B. Hansen <optmatch@ctools.umich.edu>

Depends R (>= 2.6.0)

Suggests boot

Description Functions for optimal matching, including full matching

License file LICENSE

URL <http://www.r-project.org>, <http://www.stat.lsa.umich.edu/~bbh/optmatch.html>

Repository CRAN

Date/Publication 2009-09-28 07:51:29

R topics documented:

caliper	2
fullmatch	3
mahal.dist	6
makedist	8
matched	10
matched.distances	12
mdist	13
minControlsCap	15
nuclearplants	16
pairmatch	18
plantdist	20

pscore.dist	20
relaxinfo	22
stratumStructure	22

Index	25
--------------	-----------

caliper	<i>Prepare matching distances suitable for matching within calipers</i>
---------	---

Description

Encodes calipers, or maximum allowable distances within which to match, calling `mdist()` on the supplied arguments to calculate the distance on which the caliper is based. (One then applies `pairmatch()` or `fullmatch()` to the value of `caliper`, or to the sum of it and another distance, to match within calipers.)

Usage

```
caliper(width, ..., exclude = c(), penalty = Inf)
```

Arguments

<code>width</code>	The width of the caliper: how wide of a margin to allow in matches.
<code>...</code>	Arguments are passed to <code>mdist()</code> , which forms the basic distance matrix. The caliper is then fit to this matrix. See <code>mdist</code> for more information.
<code>exclude</code>	A character vector of observations (corresponding to rownames) to exclude from the caliper.
<code>penalty</code>	Attach a penalty (to be multiplied to the distance value) for pairs outside of the caliper. The default of <code>Inf</code> prevents matches between such pairs; finite values of <code>penalty</code> permit such matches but discourage them.

Details

`width` provides the size of the caliper, the allowable distance for matching. The unit of `width` will depend on the additional arguments, used to create a basic distance matrix using `mdist`. For a formula, `width` will be in the unit of the selected covariate. For a GLM x argument, the `width` will be relative to the computed propensity score. Careful consideration should be given to what this unit means in your analysis. Similarly, a previous matching matrix will use those units.

Value

Object of class `optmatch.dlist`, which is suitable to be given as `distance` argument to `fullmatch` or `pairmatch`. For more information, see `pscore.dist`

Author(s)

Mark M. Fredrickson and Ben B. Hansen

References

P.~R. Rosenbaum and D.~B. Rubin (1985), 'Constructing a control group using multivariate matched sampling methods that incorporate the propensity score', *The American Statistician*, **39** 33–38.

See Also

[mdist](#), [fullmatch](#), [pairmatch](#)

Examples

```
data(nuclearplants)

### Caliper of .2 pooled SDs in the propensity score
ppty <- glm(pr~.(pr+cost), family=binomial(), data=nuclearplants)
(pptycaliper <- caliper(ppty, width = .2))

identical(pptycaliper, # If not writing 'width=',
caliper(.2,ppty))      # give your width first.

### Caliper on a pre-formed distance
ppty.dist <- mdist(ppty)
identical(caliper(ppty.dist, width = .2), pptycaliper)

### caliper on the Mahalanobis distance
caliper(width = 3, pr ~ t1 + t2, data = nuclearplants)

### Mahalanobis distance matching with a caliper
### of 1 pooled SD in the propensity score:
( mhd.pptyc <- caliper(ppty, width = 1) +
  mdist(pr ~ t1 + t2, data = nuclearplants) )
pairmatch(mhd.pptyc)

### Excluding observations from caliper requirements:
caliper(width = 3, pr ~ t1 + t2, data = nuclearplants, exclude = c("A", "f"))
```

fullmatch

Optimal full matching

Description

Given two groups, such as a treatment and a control group, and a treatment-by-control discrepancy matrix indicating desirability and permissibility of potential matches, create optimal full matches of members of the groups. Optionally, incorporate restrictions on matched sets' ratios of treatment to control units.

Usage

```
fullmatch(distance, min.controls = 0, max.controls = Inf,
omit.fraction = NULL, tol = 0.001, subclass.indices = NULL)
```

Arguments

- `distance` A matrix of nonnegative discrepancies, each indicating the permissibility and desirability of matching the unit corresponding to its row (a 'treatment') to the unit corresponding to its column (a 'control'); or, better, a list of such matrices, as produced by `mdist`.
- `min.controls` The minimum ratio of controls to treatments that is to be permitted within a matched set: should be nonnegative and finite. If `min.controls` is not a whole number, the reciprocal of a whole number, or zero, then it is rounded *down* to the nearest whole number or reciprocal of a whole number. When matching within subclasses, `min.controls` may be a named numeric vector separately specifying the minimum permissible ratio of controls to treatments for each subclass. The names of this vector should include names of all matrices in the list `distance`.
- `max.controls` The maximum ratio of controls to treatments that is to be permitted within a matched set: should be positive and numeric. If `max.controls` is not a whole number, the reciprocal of a whole number, or `Inf`, then it is rounded *up* to the nearest whole number or reciprocal of a whole number. When matching within subclasses, `max.controls` may be a named numeric vector separately specifying the maximum permissible ratio of controls to treatments in each subclass.
- `omit.fraction` Optionally, specify what fraction of controls or treated subjects are to be rejected. If `omit.fraction` is a positive fraction less than one, then `fullmatch` leaves up to that fraction of the control reservoir unmatched. If `omit.fraction` is a negative number greater than -1, then `fullmatch` leaves up to `omit.fraction` of the treated group unmatched. Positive values are only accepted if `max.controls` ≥ 1 ; negative values, only if `min.controls` ≤ 1 . If `omit.fraction` is not specified, then only those treated and control subjects without permissible matches among the control and treated subjects, respectively, are omitted. When matching within subclasses (so that `distance` is a list of matrices, as produced by `makedist` or `mahal.dist` or `pscore.dist`), `omit.fraction` specifies the fraction of controls to be rejected in each subproblem, a parameter that can be made to differ by subclass by setting `omit.fraction` equal to a named numeric vector of fractions.
- `tol` Because of internal rounding, `fullmatch` may solve a slightly different matching problem than the one specified, in which the match generated by `fullmatch` may not coincide with an optimal solution of the specified problem. `tol` times the number of subjects to be matched specifies the extent to which `fullmatch`'s output is permitted to differ from an optimal solution to the original problem, as measured by the sum of discrepancies for all treatments and controls placed into the same matched sets.
- `subclass.indices` An old argument included for back-compatibility; no longer needed.

Details

Finite entries in matrix slots of `distance` indicate permissible matches, with smaller discrepancies indicating more desirable matches. Matrix `distance` must have row and column names.

Consider using `mdist` to generate the distances. `fullmatch` tries to guess the order in which units would have been given in a data frame, and to order the factor that it returns accordingly. If the dimnames of `distance`, or the matrices it lists, are not simply row numbers of the data frame you're working with, then you should compare the names of `fullmatch`'s output to your row names in order to be sure things are in the proper order. You can relieve yourself of these worries by using `mdist` (or `makedist`, `pscore.dist`, or `mahal.dist`, which [as of version 0.6] are dispatched as needed by `mdist`) to produce the distances, as it passes the ordering of units to `fullmatch`, which then uses it to order its outputs.

The value of `tol` can have a substantial effect on computation time; with smaller values, computation takes longer. Not every tolerance can be met, and how small a tolerance is too small varies with the machine and with the details of the problem. If `fullmatch` can't guarantee that the tolerance is as small as the given value of argument `tol`, then matching proceeds but a warning is issued.

Value

Primarily, a named vector of class `c('optmatch', 'factor')`. Elements of this vector correspond to members of the treatment and control groups in reference to which the matching problem was posed, and are named accordingly; the names are taken from the row and column names of `distance`. Each element of the vector is either `NA`, indicating unavailability of any suitable matches for that element, or the concatenation of: (i) a character abbreviation of the name of the subclass, if matching within subclasses, or the string 'm' if not; (ii) the string `.`; and (iii) a nonnegative integer or the string `NA`. In this last place, positive whole numbers indicate placement of the unit into a matched set and `NA` indicates that all or part of the matching problem given to `fullmatch` was found to be infeasible. The functions `matched`, `unmatched`, and `matchfailed` distinguish these scenarios.

Secondarily, `fullmatch` returns various data about the matching process and its result, stored as attributes of the named vector which is its primary output. In particular, the `exceedances` attribute gives upper bounds, not necessarily sharp, for the amount by which the sum of distances between matched units in the result of `fullmatch` exceeds the least possible sum of distances between matched units in a feasible solution to the matching problem given to `fullmatch`. (Such a bound is also printed by `print.optmatch` and `summary.optmatch`.)

Author(s)

Ben Hansen

References

- Hansen, B.B. and Klopfer, S.O. (2006), 'Optimal full matching and related designs via network flows', *Journal of Computational and Graphical Statistics*, **15**, 609–627.
- Hansen, B.B. (2004), 'Full Matching in an Observational Study of Coaching for the SAT', *Journal of the American Statistical Association*, **99**, 609–618.
- Rosenbaum, P. (1991), 'A Characterization of Optimal Designs for Observational Studies', *Journal of the Royal Statistical Society, Series B*, **53**, 597–610.

See Also

`matched`, `mdist`, `caliper`

Examples

```

data(nuclearplants)
### Full matching on a Mahalanobis distance
mhd <- mdist(pr ~ t1 + t2, data = nuclearplants)
( fm1 <- fullmatch(mhd) )
summary(fm1)
### Full matching with restrictions
( fm2 <- fullmatch(mhd, min=.5, max=4) )
summary(fm2)
### Full matching to half of available controls
( fm3 <- fullmatch(mhd, omit.fraction=.5) )
summary(fm3)
### Full matching within a propensity score caliper.
ppty <- glm(pr~.(pr+cost), family=binomial(), data=nuclearplants)
### Note that units without counterparts within the
### caliper are automatically dropped.
( fm4 <- fullmatch(mhd+caliper(1, ppty)) )
summary(fm4)

### Propensity balance assessment. Requires RIttools package.
## Not run: library(RIttools) ; summary(fm4,ppty)

### Creating a data frame with the matched sets attached.
### mdist(), caliper() and the like cooperate with fullmatch()
### to make sure observations are in the proper order:
all.equal(names(fm4), row.names(nuclearplants))
### So our data frame including the matched sets is just
cbind(nuclearplants, matches=fm4)

### In contrast, if your matching distance is an ordinary matrix
### (as earlier versions of optmatch required), you'll
### have to align it by observation name with your data set.
cbind(nuclearplants, matches = fm4[row.names(nuclearplants)])

```

mahal.dist

Assemble Mahalanobis distances and prepare for matching

Description

Calculates squared Mahalanobis distances between treatment and control observations on given variables, assembling them into a discrepancy matrix (or matrices) from which `pairmatch()` or `fullmatch()` can determine optimal matches. (If vectors x and y encode two observations' values of the specified variables, then the squared Mahalanobis distance between them is

$$D^2 = (x - y)' \Sigma^{-1} (x - y)$$

.)

Usage

```
mahal.dist(distance.fmla, data, structure.fmla = NULL, inverse.cov = NULL)
```

Arguments

`distance.fmla` A formula with variables to be combined in the Mahalanobis distance on its right-hand side and the treatment variable on its left.

`data` Data frame in which `distance.fmla` and (if specified) `structure.fmla` are to be evaluated.

`structure.fmla` Optional formula argument specifying subclasses within which matches are to be performed. If omitted, no subclassification is done. If it is given, its left-hand side gives the treatment variable and its RHS gives variables on which to stratify the sample prior to matching.

`inverse.cov` The inverse covariance of the variables to be combined into the Mahalanobis distance (optional).

Details

Mahalanobis distance tracks the discrepancy between points on a number of given variables, after standardizing the variables and taking account of their covariances. It is best suited to variables whose joint distribution resembles a multivariate Normal.

The purpose of giving a `structure.fmla` argument is to speed up large problems. Variables appearing on its right-hand side will be interacted to create the subclasses. If `structure.fmla` is given then its LHS is used to define treatment and control groups (and one doesn't have to put anything on the LHS of `distance.fmla`).

The function attempts to calculate the inverse covariance itself, so ordinarily you shouldn't need to give it one. If you'll be calling the function repeatedly, however, it may speed things up to compute and store the inverse covariance once, rather than each time this function is called; in that case you can save time by giving the `inverse.covariance` argument.

Value

Object of class `optmatch.dlist`, which is suitable to be given as `distance` argument to `fullmatch` or `pairmatch`.

Specifically, a list of matrices, one for each subclass defined by the interaction of variables appearing on the right hand side of `structure.fmla`. Each of these is a number of treatments by number of controls matrix of propensity distances. The list also carries some metadata as attributes, data that is not of direct interest to the user but is useful to `fullmatch()` and `pairmatch()`.

Author(s)

Ben B. Hansen

See Also

`makedist`, `pscore.dist`, `fullmatch`, `pairmatch`

Examples

```

data(nuclearplants)
mhd1 <- mahal.dist(pr~date+cum.n, nuclearplants)
lapply(mhd1, round)
attributes(mhd1)
fullmatch(mhd1)
##- Mahalanobis within subclasses defined by levels of pt
mhd2 <- mahal.dist(~date+cum.n, nuclearplants, pr~pt)
lapply(mhd2, round)
fullmatch(mhd2)
##- Trick mahal.dist into returning absolute differences on a scalar.
mhd3 <- mahal.dist(pr~date, nuclearplants,
inverse.cov=matrix(1,1,1,dimnames=list("date", "date")))
mhd3[[1]]
##- Matching within calipers of 3 years
fullmatch(mhd1/(mhd3<3))

```

makedist

Assemble match distances from a data frame

Description

Helper function to produce first arguments to `fullmatch()`, reducing memory requirements for `fullmatch()` and heading off certain user errors.

Usage

```

makedist(structure.fmla, data, fn = function(trtvar, dat, ...) {
  matrix(0, sum(trtvar), sum(!trtvar), dimnames = list(names(trtvar)[trtvar], names(!trtvar)[!trtvar]), ...)
}, ...)

```

Arguments

<code>structure.fmla</code>	A formula defined w.r.t. data frame <code>data</code> , with a treatment variable on the LHS and either '1', if no stratification prior to matching, or variables defining pre-matching stratification on the RHS.
<code>data</code>	A data frame in which <code>structure.fmla</code> is evaluated and <code>fn</code> operates.
<code>fn</code>	A user-supplied function to compute distances. See details and examples.
<code>...</code>	Additional arguments to <code>fn</code>

Details

`fn` should be a function with first two arguments `trtvar`, a treatment variable, and `dat`, a data frame. There may be additional arguments. If the function uses variables in `dat`, these should be referenced using names from the input `trtvar`, particularly if the sample is being split into strata (ie `structure.fmla` has a non-trivial RHS). When this happens, `fn` will be passed a `trtvar`

input observations for only a subset of the rows of `dat`, so it has to use `trtvar` to decide which rows of `dat` to operate on; it does this by lining up names of the (shorter) vector `trtvar` with row names of `dat`.

Value

A list of matrices, one for each subclass defined by the interaction of variables appearing on the right hand side of `structure.fmla`. Each of these is a number of treatments by number of controls matrix of distances, with the distance between treatments and controls calculated by the user-given function `fn`.

The list also has some attributes that are not of direct interest to the user, but are used by `fullmatch()`.

Note

Use of this function to prepare input to `fullmatch()` prevents two common problems. First, the function encourages you to stratify a large data set and match within strata, then calculates distances only between potential matches within the same stratum. Whether or not you stratify, the function has the advantage of keeping track of the order of observations in a data frame from which distances are generated. It passes this info to `fullmatch()`, which makes sure to return the vector a vector with the ordering of the generating data frame. It's still possible to create matrices of distances without this function, but then `fullmatch()` has no way of knowing the order of observations in whatever data frame you're working from, since that info is lost in the transition to a distance matrix.

Author(s)

Ben Hansen

See Also

[fullmatch](#), [pscore.dist](#), [mahal.dist](#)

Examples

```
data(nuclearplants)

##-- A distance function used in P. Rosenbaum's (2002) book
rankdiffs <- function(trtvar, dat, vars)
{
  dmt <- matrix(0, sum(trtvar), sum(!trtvar))
  for (vv in vars) {
    vvr <- rank(dat[names(trtvar), vv])
    dmt <- dmt + abs(outer(vvr[trtvar], vvr[!trtvar], "-"))
  }
  round(dmt)
}

##-- Gives a warning because this fn doesn't assign dimnames
(rdd1 <- makedist(pr~1, nuclearplants[nuclearplants$pt==0,], rankdiffs, c("cap", "date")))
fullmatch(rdd1)
##-- fullmatch() knows its value should be ordered as the nuclearplants data set is
rdd1$m
```

```

##-- now fullmatch() doesn't know the proper order of units and has to guess
fullmatch(rdd1$m)
(rdd2 <- makedist(pr~pt, nuclearplants, rankdiffs, c("cap","date")))
fullmatch(rdd2)

##- Distance on a propensity score
scalardiffs <- function(trtvar,data,scalarname) {
  sclr <- data[names(trtvar), scalarname]
  names(sclr) <- names(trtvar)
  abs(outer(sclr[trtvar],sclr[!trtvar], '-'))
}
nuclearplants$pscore <- glm(pr~.(pr+cost), family=binomial(),
                           data=nuclearplants)$linear.predictors
##-- Distance for propensity score matching w/o prior stratification
psd1 <- makedist(pr~1, nuclearplants, scalardiffs, "pscore")
fullmatch(psd1)
##-- Distance for propensity score matching within levels of "pt"
psd2 <- makedist(pr~pt, nuclearplants, scalardiffs, "pscore")
fullmatch(psd2)

```

 matched

Identification of units placed into matched sets

Description

Given a bipartite matching (object of class `optmatch`), create a logical vector of the same length indicating which units were and were not placed into matched sets.

Usage

```

matched(matchobject)
unmatched(matchobject)
matchfailed(matchobject)

```

Arguments

`matchobject` Vector of class `optmatch` (especially as generated by a call to `fullmatch`).

Details

`matched` and `unmatched` indicate which elements of `matchobject` do and do not belong to matched sets, as indicated by their character representations in `matchobject`.

When `fullmatch` has been presented with an inconsistent combination of constraints and discrepancies between potential matches, so that there exists no matching (i) with finite total discrepancy within matched sets that (ii) respects the given constraints, then the matching problem is said to be infeasible. TRUEs in the output of `matchfailed` indicate that this has occurred.

Value

A logical vector (without names).

Note

To understand the output of `matchfailed` element-wise, note that `fullmatch` handles a matching problem in three steps. First, if `fullmatch` has been directed to match within subclasses, then it divides its matching problem into a subproblem for each subclass. Second, `fullmatch` removes from each subproblem those individual units that lack permissible potential matches (i.e. potential matches from which they are separated by a finite discrepancy). Such “isolated” units are flagged in such a way as to be indicated by `unmatched`, but not by `matchfailed`. Third, `fullmatch` presents each subproblem, with isolated elements removed, to an optimal matching routine. If such a reduced subproblem is found at this stage to be infeasible, then each unit contributing to it is so flagged as to be indicated by `matchfailed`.

Author(s)

Ben Hansen

See Also

[fullmatch](#)

Examples

```
plantdist <- matrix(nrow=7, ncol=19, byrow=TRUE, data=c(
  28, 0, 3, 22, 14, 30, 17, 28, 26, 28, 20, 22, 23, 26, 21, 18, 34, 40, 28,
  24, 3, 0, 22, 10, 27, 14, 26, 24, 24, 16, 19, 20, 23, 18, 16, 31, 37, 25,
  10, 18, 14, 18, 4, 12, 6, 11, 9, 10, 14, 12, 6, 14, 22, 10, 16, 22, 28,
  7, 28, 24, 8, 14, 2, 10, 6, 12, 0, 24, 22, 4, 24, 32, 20, 18, 16, 38,
  17, 20, 16, 32, 18, 26, 20, 18, 12, 24, 0, 2, 20, 6, 8, 4, 14, 20, 14,
  20, 31, 28, 35, 20, 29, 22, 20, 14, 26, 12, 9, 22, 5, 15, 12, 9, 11, 12,
  14, 32, 29, 30, 18, 24, 17, 16, 10, 22, 12, 10, 17, 6, 16, 14, 4, 8, 17),
  dimnames=list(c("A", "B", "C", "D", "E", "F", "G"),
  c("H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R",
  "S", "T", "U", "V", "W", "X", "Y", "Z")))

mxpl.fm0 <- fullmatch(plantdist) # A feasible matching problem
c(sum(matched(mxpl.fm0)), sum(unmatched(mxpl.fm0)))
sum(matchfailed(mxpl.fm0))
mxpl.fm1 <- fullmatch(plantdist, # An infeasible problem
  max.controls=3, min.controls=3)
c(sum(matched(mxpl.fm1)), sum(unmatched(mxpl.fm1)))
sum(matchfailed(mxpl.fm1))

mxpl.si <- factor(c('a', 'a', 'c', rep('d', 4), 'b', 'c', 'c', rep('d', 16)))
names(mxpl.si) <- LETTERS[1:26]
# Subclass a contains two treated units but no controls;
# subclass b contains only a control unit;
# subclass c contains one treated and two control units;
# subclass d contains the remaining twenty units.
```

```

mcl <- c(Inf, Inf, 1, Inf)
names(mcl) <- c('a', 'b', 'c', 'd')

mxpl.fm2 <- fullmatch(plantdist,      # contains four subproblems,
                     subclass.indices= # three of which are infeasible
                     mxpl.si,
                     max.controls=mcl)
sum(matched(mxpl.fm2))

table(unmatched(mxpl.fm2), matchfailed(mxpl.fm2))

mxpl.fm2[matchfailed(mxpl.fm2)]

mxpl.fm2[unmatched(mxpl.fm2) & # isolated units return as
         !matchfailed(mxpl.fm2)] # unmatched but not matchfailed

```

matched.distances *Determine distances between matched units*

Description

From a match (as produced by `pairmatch` or `fullmatch`) and a distance, extract the distances of matched units from their matched counterparts.

Usage

```
matched.distances(matchobj, distance, preserve.unit.names=FALSE)
```

Arguments

<code>matchobj</code>	Value of a call to <code>pairmatch</code> or <code>fullmatch</code> .
<code>distance</code>	Either a distance matrix or the value of a call to <code>pscore.dist</code> , <code>mahal.dist</code> , or <code>makedist</code> .
<code>preserve.unit.names</code>	Logical. If true, for each matched set <code>matched.distances</code> returns the submatrix of the distance matrix corresponding to it; if false, a vector containing the distances in that submatrix is returned.

Value

A list of numeric vectors (or matrices) of distances, one for each matched set. Note that a matched set with 1 treatment and k controls, or with k treatments and 1 control, has k, not k+1, distances.

Author(s)

Ben B. Hansen

Examples

```
data(plantdist)
plantsfm <- fullmatch(plantdist)
(plantsfm.d <- matched.distances(plantsfm, plantdist, pres=TRUE))
unlist(lapply(plantsfm.d, max))
mean(unlist(plantsfm.d))
```

mdist	<i>Creates matrices of mdists (distances) between observations for matching</i>
-------	---

Description

A generic function, with several supplied methods, for creating distance matrices between observations to be used in the match process. Using these matrices, `pairmatch()` or `fullmatch()` can determine optimal matches.

Usage

```
mdist(x, structure.fmla = NULL, ...)
```

Arguments

<code>x</code>	The object to use as the basis for forming the mdist matrix. Methods exist for formulas, functions, and generalized linear models.
<code>structure.fmla</code>	A formula denoting the treatment variable on the left hand side and an optional grouping expression on the right hand side. For example, <code>z ~ 1</code> indicates no grouping. <code>z ~ s</code> subsets the data only computing distances within the subsets formed by <code>s</code> . See method notes for for any additional formula options.
<code>...</code>	Additional method arguments. Most methods require a 'data' argument.

Details

The `mdist` method provides three ways to construct a mdist matrix (or list of mdist matrices): `function`, `glm`, and `formula`.

The `mdist.function` method takes a function of two arguments. When called, this function will receive the treatment observations as the first argument and the control observations as the second argument. As an example, the following computes the raw differences between values of `t1` for treatment units (here, nuclear plants with `pr==1`) and controls (here, plants with `pr==0`), returning the result as a distance matrix: `sdiffs <- function(treatments, controls) { abs(outer(treatments$t1, controls$t1, '-')) }`

The `mdist.function` method does similar things as the earlier `optmatch` function `makedist`, although the interface is a bit different.

The `mdist.formula` computes the squared Mahalanobis distance between observations using the supplied formula. In addition to the distance formula (the first argument), this method can also

take a structure formula to denote strata in the observations, e.g. `~ s` would group the observations by the factor `s`.

The `mdist.glm` takes an argument of class `glm` as the first argument. It computes the deviations between observations using the `mad` function. See `pscore.dist` for more information.

Value

Object of class `optmatch.dlist`, which is suitable to be given as `distance` argument to `fullmatch` or `pairmatch`. For more information, see `pscore.dist`

Author(s)

Mark M. Fredrickson

References

P.-R. Rosenbaum and D.~B. Rubin (1985), ‘Constructing a control group using multivariate matched sampling methods that incorporate the propensity score’, *The American Statistician*, **39** 33–38.

See Also

`makedist`, `mahal.dist`, `fullmatch`, `pairmatch`, `pscore.dist`

Examples

```
data(nuclearplants)

### A propensity score distance:
aGlm <- glm(pr~.(pr+cost), family=binomial(), data=nuclearplants)
mdist(aGlm)

### A Mahalanobis distance:
mdist(pr ~ t1 + t2, data = nuclearplants)

### Absolute difference on a scalar-distance:

sdiffs <- function(treatments, controls) {
  abs(outer(treatments$t1, controls$t1, `~`))
}

(absdist <- mdist(sdiffs, structure.fmla = pr ~ 1, data = nuclearplants))

### Using pairmatch on the scalar example:
pairmatch(absdist)
```

minControlsCap *Set thinning and thickening caps for full matching*

Description

Functions to find the largest value of min.controls, or the smallest value of max.controls, for which a full matching problem is feasible. These are determined by constraints embedded in the matching problem's distance matrix.

Usage

```
minControlsCap(distance, max.controls = NULL, subclass.indices = NULL)
maxControlsCap(distance, min.controls = NULL, subclass.indices = NULL)
```

Arguments

`distance` Either a matrix of nonnegative, numeric discrepancies, or a list of such matrices. (See [fullmatch](#) for details.)

`max.controls` Optionally, set limits on the maximum number of controls per matched set. (Only makes sense for minControlsCap.)

`min.controls` Optionally, set limits on the minimum number of controls per matched set. (Only makes sense for maxControlsCap.)

`subclass.indices`
 This argument no longer supported (nor necessary).

Details

The function works by repeated application of full matching, so on large problems it can be time-consuming.

Value

For minControlsCap, `strictest.feasible.min.controls` and `given.max.controls`.
 For maxControlsCap, `given.min.controls` and `strictest.feasible.max.controls`.

`strictest.feasible.min.controls`
 The largest values of the [fullmatch](#) argument `min.controls` that yield a full match;

`given.max.controls`
 The `max.controls` argument given to minControlsCap or, if none was given, a vector of Infs.

`given.min.controls`
 The `min.controls` argument given to maxControlsCap or, if none was given, a vector of 0s;

`strictest.feasible.max.controls`
 The smallest values of the [fullmatch](#) argument `max.controls` that yield a full match.

Note

Essentially this is just a line search. I've done several things to speed it up, but not everything that might be done. At present, not very thoroughly tested either: you might check the final results to make sure that `fullmatch` works with the values of `min.controls` (or `max.controls`) suggested by these functions, and that it ceases to work if you increase (decrease) those values. Comments appreciated.

Author(s)

Ben B. Hansen

References

Hansen, B.B. and S. Olsen Klopfer (2006), 'Optimal full matching and related designs via network flows', *Journal of Computational and Graphical Statistics* **15**, 609–627.

See Also

`fullmatch`

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==> Define data, use random,
##-- or do help(data=index) for the standard data sets.
plantdist <- matrix(nrow=7, ncol=19,byrow=TRUE,data=c(
  28, 0, 3,22,14,30,17,28,26,28,20,22,23,26,21,18,34,40,28,
  24, 3, 0,22,10,27,14,26,24,24,16,19,20,23,18,16,31,37,25,
  10,18,14,18, 4,12, 6,11, 9,10,14,12, 6,14,22,10,16,22,28,
  7,28,24, 8,14, 2,10, 6,12, 0,24,22, 4,24,32,20,18,16,38,
  17,20,16,32,18,26,20,18,12,24, 0, 2,20, 6, 8, 4,14,20,14,
  20,31,28,35,20,29,22,20,14,26,12, 9,22, 5,15,12, 9,11,12,
  14,32,29,30,18,24,17,16,10,22,12,10,17, 6,16,14, 4, 8,17),
  dimnames=list(c("A","B","C","D","E","F","G"),
  c("H","I","J","K","L","M","N","O","P","Q","R",
  "S","T","U","V","W","X","Y","Z"))

(tmn <- minControlsCap(plantdist)$strictest)
maxControlsCap(plantdist, min=tmn)
splitdist <- list(one=plantdist[1:3, 1:9], two=plantdist[4:7, 10:19])
(tmn <- minControlsCap(splitdist)$strictest)
maxControlsCap(splitdist, min=tmn)
```

Description

The `nuclearplants` data frame has 32 rows and 11 columns.

The data relate to the construction of 32 light water reactor (LWR) plants constructed in the U.S.A in the late 1960's and early 1970's. The data was collected with the aim of predicting the cost of construction of further LWR plants. 6 of the power plants had partial turnkey guarantees and it is possible that, for these plants, some manufacturers' subsidies may be hidden in the quoted capital costs.

Usage

```
nuclearplants
```

Format

This data frame contains the following columns:

- cost** The capital cost of construction in millions of dollars adjusted to 1976 base.
- date** The date on which the construction permit was issued. The data are measured in years since January 1 1990 to the nearest month.
- t1** The time between application for and issue of the construction permit.
- t2** The time between issue of operating license and construction permit.
- cap** The net capacity of the power plant (MWe).
- pr** A binary variable where 1 indicates the prior existence of a LWR plant at the same site.
- ne** A binary variable where 1 indicates that the plant was constructed in the north-east region of the U.S.A.
- ct** A binary variable where 1 indicates the use of a cooling tower in the plant.
- bw** A binary variable where 1 indicates that the nuclear steam supply system was manufactured by Babcock-Wilcox.
- cum.n** The cumulative number of power plants constructed by each architect-engineer.
- pt** A binary variable where 1 indicates those plants with partial turnkey guarantees.

Source

The data were obtained from the `boot` package, for which they were in turn taken from Cox and Snell (1981). Although the data themselves are the same as those in the `nuclear` data frame in the `boot` package, the row names of the data frame have been changed. (The new row names were selected to ease certain demonstrations in `optmatch`.)

This documentation page is also adapted from the `boot` package, written by Angelo Canty and ported to R by Brian Ripley.

References

Cox, D.R. and Snell, E.J. (1981) *Applied Statistics: Principles and Examples*. Chapman and Hall.

 pairmatch

Optimal 1:1 and 1:k matching

Description

Given a treatment group, a larger control reservoir, and discrepancies between each treatment and control unit, finds a pairing of treatment units to controls that minimizes the sum of discrepancies.

Usage

```
pairmatch(distance, controls = 1, tol = 0.001)
```

Arguments

<code>distance</code>	A matrix of nonnegative discrepancies, each indicating the permissibility and desirability of matching the unit corresponding to its row (a 'treatment') to the unit corresponding to its column (a 'control'); or a list of such matrices made using <code>mdist</code> . Finite discrepancies indicate permissible matches, with smaller discrepancies indicating more desirable matches. Matrix <code>distance</code> , or the matrix elements of <code>distance</code> , must have row and column names.
<code>controls</code>	The number of controls to be matched to each treatment.
<code>tol</code>	Tolerance – see <code>fullmatch</code> for details.

Details

This is a wrapper to `fullmatch`; see its documentation for more information.

`fullmatch` tries to guess the order in which units would have been given in a data frame, and to order the factor that it returns accordingly. If the `dimnames` of `distance`, or the matrices it lists, are not simply row numbers of the data frame you're working with, then you should compare the names of `fullmatch`'s output to your row names in order to be sure things are in the proper order. You can relieve yourself of these worries by using `mdist` to produce the distances, as it passes the ordering of units to `fullmatch`, which then uses it to order its outputs.

The value of `tol` can have a substantial effect on computation time; with smaller values, computation takes longer. Not every tolerance can be met, and how small a tolerance is too small varies with the machine and with the details of the problem. If `fullmatch` can't guarantee that the tolerance is as small as the given value of argument `tol`, then matching proceeds but a warning is issued.

Value

Primarily, a named vector of class `c('optmatch', 'factor')`. Elements of this vector correspond to members of the treatment and control groups in reference to which the matching problem was posed, and are named accordingly; the names are taken from the row and column names of `distance`. Each element of the vector is the concatenation of: (i) a character abbreviation of `subclass.indices`, if that argument was given, or the string 'm' if it was not; (ii) the string `.`; and (iii) a nonnegative integer or the string `NA`. In this last place, positive whole numbers indicate placement of the unit into a matched set, a number beginning with zero indicates

a unit that was not matched, and NA indicates that all or part of the matching problem given to `fullmatch` was found to be infeasible. Secondly, `fullmatch` returns various data about the matching process and its result, stored as attributes of the named vector which is its primary output. In particular, the `exceedances` attribute gives upper bounds, not necessarily sharp, for the amount by which the sum of distances between matched units in the result of `fullmatch` exceeds the least possible sum of distances between matched units in a feasible solution to the matching problem given to `fullmatch`. (Such a bound is also printed by `print.optmatch` and by `summary.optmatch`.)

References

Hansen, B.B. and Klopfer, S.O. (2006), ‘Optimal full matching and related designs via network flows’, *Journal of Computational and Graphical Statistics*, **15**, 609–627.

See Also

[matched](#), [caliper](#), [fullmatch](#), [makedist](#)

Examples

```
data(nuclearplants)
### Pair matching on a Mahalanobis distance
mhd <- mdist(pr ~ t1 + t2, data = nuclearplants)
( pm1 <- pairmatch(mhd) )
summary(pm1)
### Pair matching within a propensity score caliper.
ppty <- glm(pr~.(pr+cost), family=binomial(), data=nuclearplants)
(pm2 <- pairmatch(mhd+caliper(1,ppty)))
summary(pm2)

### Propensity balance assessment. Requires RIttools package.
## Not run: library(RIttools) ; summary(pm2,ppty)

### 1:2 matched triples
tm <- pairmatch(mhd, controls=2)
summary(tm)

### Creating a data frame with the matched sets attached.
### mdist(), caliper() and the like cooperate with pairmatch()
### to make sure observations are in the proper order:
all.equal(names(tm), row.names(nuclearplants))
### So our data frame including the matched sets is just
cbind(nuclearplants, matches=tm)

### In contrast, if your matching distance is an ordinary matrix
### (as earlier versions of optmatch required), you'll
### have to align it by observation name with your data set.
cbind(nuclearplants, matches = tm[row.names(nuclearplants)])
```

plantdist

Dissimilarities of Some U.S. Nuclear Plants

Description

This matrix gives discrepancies between light water nuclear power plants of two types, seven built on the site of an existing plant and 19 built on new sites. The discrepancies summarize differences in two covariates that are predictive of the cost of building a plant.

Usage

```
data(plantdist)
```

Format

A 7 by 19 numeric matrix.

Source

The data appear in Cox, D.R. and Snell, E.J. (1981), *Applied Statistics: Principles and Examples*, p.82 (Chapman and Hall), and are due to W.E. Mooz.

References

Rosenbaum, P.R. (2002), *Observational Studies*, Second Edition, p.307 (Springer).

pscore.dist

Assemble propensity distances and prepare for matching

Description

Extracts scores from a fitted propensity scoring model, assembling them into a discrepancy matrix (or matrices) from which `pairmatch()` or `fullmatch()` can determine optimal matches.

Usage

```
pscore.dist(glmobject, structure.fmla = NULL, standardization.scale=sd)
```

Arguments

<code>glmobject</code>	A fitted propensity modeling object, produced by a call to <code>glm()</code> or, say, <code>bayesglm</code> (from package “arm”) or <code>brglm()</code> (from the “brglm” package).
<code>structure.fmla</code>	Optional formula argument specifying subclasses within which matches are to be performed. If omitted, no subclassification is done. If it is given, the RHS of this formula gives variables on which to stratify the sample prior to matching.
<code>standardization.scale</code>	Scalar-valued function to be applied separately to treatment and control propensity scores to determine their scale; propensity distances will be scaled by a pooling of these two values. Defaults to <code>sd</code> , which gives scaling by the pooled sd of propensity scores. (Another good choice is <code>mad</code> , which is robust to outliers but gives results similar to <code>sd</code> in the absence of outliers.)

Details

`glmobject` need not necessarily be the result of a call to `glm`. It should be a list with elements: `y`, a vector that is positive for treatment subjects and nonpositive for controls; `linear.predictors`, containing the propensity scores; and `data`, the data frame from which propensities were made.

The purpose of giving a `structure.fmla` argument is to speed up large problems. Variables appearing on its right-hand side will be interacted to create these subclasses; the same variables should also have appeared on the RHS of the formula used to specify the propensity model.

This is a wrapper to `makedist`, so it keeps track of metadata useful for matching, as described on the help page for that function.

Value

Object of class `optmatch.dlist`, which is suitable to be given as `distance` argument to `fullmatch` or `pairmatch`.

Specifically, a list of matrices, one for each subclass defined by the interaction of variables appearing on the right hand side of `structure.fmla`. Each of these is a number of treatments by number of controls matrix of propensity distances. The distances are differences of the linear predictor from the propensity model, rather than differences of estimated probabilities, avoiding compression of estimated propensities near 0 and 1 (Rosenbaum and Rubin, 1985). They will have been scaled by the pooled SD of propensity scores in the treatment and control groups, so that a caliper of .25 pooled SDs on the propensity score can be coded as `value/(value<=.25)`; see the examples. The list also carries some metadata as attributes, data that are not of direct interest to the user but are useful to `fullmatch()` and `pairmatch()`.

Author(s)

Ben B. Hansen

References

P.~R. Rosenbaum and D.~B. Rubin (1985), ‘Constructing a control group using multivariate matched sampling methods that incorporate the propensity score’, *The American Statistician*, **39** 33–38.

See Also

`makedist`, `mahal.dist`, `fullmatch`, `pairmatch`, `caliper`

Examples

```
data(nuclearplants)
psm <- glm(pr~.(pr+cost), family=binomial(), data=nuclearplants)
psd <- pscore.dist(psm)
fullmatch(psd)
fullmatch(caliper(.25, psd)) # Propensity matching with calipers
```

`relaxinfo`

Display licence information about embedded code

Description

Function to display licence information regarding code embedded in `optmatch`

Usage

```
relaxinfo()
```

Value

Some information about licences of code and algorithms on which `fullmatch` depends.

Author(s)

Ben B. Hansen

`stratumStructure`

Return structure of matched sets

Description

Tabulate treatment:control ratios occurring in matched sets, and the frequency of their occurrence.

Usage

```
stratumStructure(stratum, trtgrp=NULL, min.controls=0, max.controls=Inf)
```

Arguments

<code>stratum</code>	Matched strata, as returned by <code>fullmatch</code> or <code>pairmatch</code>
<code>trtgrp</code>	Dummy variable for treatment group membership. (Not required if <code>stratum</code> is an <code>optmatch</code> object, as returned by <code>fullmatch</code> or <code>pairmatch</code> .)
<code>min.controls</code>	For display, the number of treatment group members per stratum will be truncated at the reciprocal of <code>min.controls</code> .
<code>max.controls</code>	For display, the number of control group members will be truncated at <code>max.controls</code> .

Value

A table showing frequency of occurrence of those treatment:control ratios that occur.

The ‘effective sample size’ of the stratification, in matched pairs. Given as an attribute of the table, named ‘`comparable.num.matched.pairs`’; see Note.

Note

For comparing treatment and control groups both of size 10, say, a stratification consisting of two strata, one with 9 treatments and 1 control, has a smaller ‘effective sample size’, intuitively, than a stratification into 10 matched pairs, despite the fact that both contain 20 subjects in total. `stratumStructure` first summarizes this aspect of the structure of the stratification it is given, then goes on to identify one number as the stratification’s effective sample size. The ‘`comparable.num.matched.pairs`’ attribute returned by `stratumStructure` is the sum of harmonic means of the sizes of the treatment and control subgroups of each stratum, a general way of calibrating such differences as well as differences in the number of subjects contained in a stratification. For example, by this metric the 9:1, 1:9 stratification is comparable to 3.6 matched pairs.

Why should effective sample size be calculated this way? The phrase ‘effective sample size’ suggests the observations are taken to be similar in information content. Modeling them as random variables, this suggests that they be assumed to have the same variance, σ , conditional on what stratum they reside in. If that is the case, and if also treatment and control observations differ in expectation by a constant that is the same for each stratum, then it can be shown that the optimum weights with which to combine treatment-control contrasts across strata, s , are proportional to the stratum-wise harmonic means of treatment and control counts, $h_s = [(n_{ts}^{-1} + n_{cs}^{-1})/2]^{-1}$ (Kalton, 1968). The thus-weighted average of contrasts then has variance $2\sigma^2 / \sum_s h_s$. This motivates the use of $\sum_s h_s$ as a measure of effective sample size. Since for a matched pair s , $h_s = 1$, $\sum_s h_s$ can be thought of as the number of matched pairs needed to attain comparable precision. (Alternately, the stratification might be taken into account when comparing treatment and control groups using fixed effects in an ordinary least-squares regression, as in Hansen (2004). This leads to the same result. A still different formulation, in which outcomes are not modeled as random variables but as assignment to treatment or control is, again suggests the same weighting across strata, and a measure of precision featuring $\sum_s h_s$ in a similar role; see Hansen and Bowers (2008).)

Author(s)

Ben Hansen

References

- Kalton, G. (1968), ‘Standardization: A technique to control for extraneous variables’, *Applied Statistics*, **17**, 118–136.
- Hansen, B.B. (2004), ‘Full Matching in an Observational Study of Coaching for the SAT’, *Journal of the American Statistical Association*, **99**, 609–618.
- Hansen B.B. and Bowers, J. (2008), ‘Covariate balance in simple, stratified and clustered comparative studies’, *Statistical Science*, **23**, to appear.

See Also

[matched](#), [fullmatch](#)

Examples

```
data(plantdist)

plantsfm <- fullmatch(plantdist) # A full match with unrestricted
                                # treatment-control balance
plantsfm1 <- fullmatch(plantdist,min.controls=2, max.controls=3)

stratumStructure(plantsfm)
stratumStructure(plantsfm1)
stratumStructure(plantsfm, max.controls=4)
```

Index

- *Topic **datasets**
 - nuclearplants, 16
 - plantdist, 20
 - *Topic **manip**
 - matched, 10
 - *Topic **misc**
 - stratumStructure, 22
 - *Topic **nonparametric**
 - caliper, 2
 - fullmatch, 3
 - mahal.dist, 6
 - makedist, 8
 - matched.distances, 12
 - mdist, 13
 - pairmatch, 18
 - pscore.dist, 20
 - *Topic **optimize**
 - fullmatch, 3
 - minControlsCap, 15
 - pairmatch, 18
 - relaxinfo, 22
- caliper, 2, 5, 19, 22
- fullmatch, 2, 3, 3, 7, 9, 11, 14–16, 18, 19, 21–24
- mad, 21
- mahal.dist, 5, 6, 9, 14, 22
- makedist, 5, 7, 8, 13, 14, 19, 21, 22
- matched, 5, 10, 19, 24
- matched.distances, 12
- matchfailed, 5
- matchfailed (*matched*), 10
- maxControlsCap (*minControlsCap*), 15
- mdist, 2–5, 13, 18
- minControlsCap, 15
- nuclearplants, 16
- pairmatch, 2, 3, 7, 14, 18, 21–23
- plantdist, 20
- pscore.dist, 2, 5, 7, 9, 14, 20
- relaxinfo, 22
- sd, 21
- stratumStructure, 22
- unmatched, 5
- unmatched (*matched*), 10