Package 'optmatch'

September 19, 2024

Version 0.10.8

Title Functions for Optimal Matching

Description Distance based bipartite matching using minimum cost flow, oriented to matching of treatment and control groups in observational studies ('Hansen' and 'Klopfer' 2006 <doi:10.1198/106186006X137047>). Routines are provided to generate distances from generalised linear models (propensity score matching), formulas giving variables on which to limit matched distances, stratified or exact matching directives, or calipers, alone or in combination.

LazyData true

Depends R (>= 3.5.0)

LinkingTo Rcpp

Imports Rcpp, dplyr, stats, tibble, methods, graphics, rlemon

Suggests RItools, boot, biglm, survey, testthat (>= 0.11.0), roxygen2, brglm, arm, knitr, rmarkdown, markdown, pander, xtable, rrelaxiv, magrittr

Enhances CBPS, haven

License MIT + file LICENSE

License_is_FOSS yes

License_restricts_use no

URL http://optmat.ch, https://github.com/markmfredrickson/optmatch

BugReports https://github.com/markmfredrickson/optmatch/issues

Additional_repositories https://errickson.net/rrelaxiv

Collate 'DenseMatrix.R' 'InfinitySparseMatrix.R' 'MCFSolutions.R' 'Ops.optmatch.dlist.R' 'Optmatch-package.R' 'RcppExports.R' 'abs.optmatch.dlist.R' 'boxplotMethods.R' 'caliper.R' 'complementarySlackness.R' 'data.R' 'deprecated.R' 'distUnion.R' 'edgelist.R' 'exactMatch.R' 'feasible.R' 'fill.NAs.R' 'fmatch.R' 'fullmatch.R' 'makedist.R' 'match_on.R' 'matched.R' 'matched.distances.R' 'matchfailed.R'

'max.controls.cap.R' 'mdist.R' 'min.controls.cap.R'
'optmatchS3.R' 'pairmatch.R' 'print.optmatch.R'
'print.optmatch.dlist.R' 'scores.R' 'solve_reg_fm_prob.R'
'solver.R' 'strata.R' 'stratumStructure.R' 'summary.ism.R'
'summary.optmatch.R' 'utilities.R' 'zzz.R'
'zzzDistanceSpecification.R'

VignetteBuilder knitr

RoxygenNote 7.3.2

Encoding UTF-8

NeedsCompilation yes

Author Ben Hansen [aut],
Mark Fredrickson [aut],
Josh Errickson [cre, aut],
Josh Buckner [aut],
Adam Rauh [ctb]

Maintainer Josh Errickson <jerrick@umich.edu>

Repository CRAN

Date/Publication 2024-09-19 06:20:02 UTC

Contents

`+,InfinitySparseMatrix,InfinitySparseMatrix-method`
Element-wise addition

Description

`e1 + e2` returns the element-wise sum of two InfinitySparseMatrix objects. If either element is inf then the resulting element will be inf.

`e1 - e2` returns the element-wise subtraction of two InfinitySparseMatrix objects. If either element is inf then the resulting element will be inf.

`e1 * e2` returns the element-wise multiplication of two InfinitySparseMatrix objects. If either element is inf then the resulting element will be inf.

`e1 / e2` returns the element-wise division of two InfinitySparseMatrix objects. If either element is inf then the resulting element will be inf.

Usage

```
## S4 method for signature 'InfinitySparseMatrix,InfinitySparseMatrix'
e1 + e2

## S4 method for signature 'InfinitySparseMatrix,InfinitySparseMatrix'
e1 - e2

## S4 method for signature 'InfinitySparseMatrix,InfinitySparseMatrix'
e1 * e2

## S4 method for signature 'InfinitySparseMatrix,InfinitySparseMatrix'
e1 / e2
```

Arguments

e1	an InfinitySparseMatrix object
e2	an InfinitySparseMatrix object

Value

an InfinitySparseMatrix object representing the element-wise sum of the two ISM objects

---

`antiExactMatch` *Specify a matching problem where units in a common factor cannot be matched.*

Description

This function builds a distance specification where treated units are infinitely far away from control units that share the same level of a given factor variable. This can be useful for ensuring that matched groups come from qualitatively different groups.

Usage

```
antiExactMatch(x, z)
```

Arguments

x	A factor across which matches should be allowed.
z	A logical or binary vector the same length as x indicating treatment and control for each unit in the study. TRUE or 1 represents a treatment unit, FALSE of 0 represents a control unit. NA units are excluded.

Details

The exactMatch function provides a way of specifying a matching problem where only units within a factor level may be matched. This function provides the reverse scenario: a matching problem in which only units across factor levels are permitted to match. Like exactMatch, the results of this function will most often be used as a within argument to match_on or another distance specification creation function to limit the scope of the final distance specification (i.e., disallowing any match between units with the same value on the factor variable x).

Value

A distance specification that encodes the across factor level constraint.

See Also

exactMatch, match_on, caliper, fullmatch, pairmatch

Examples

```
data(nuclearplants)

# force entries to be within the same factor:
em <- fullmatch(exactMatch(pr ~ pt, data = nuclearplants), data = nuclearplants)
table(nuclearplants$pt, em)

# force treated and control units to have different values of `pt`:
z <- nuclearplants$pr
names(z) <- rownames(nuclearplants)
aem <- fullmatch(antiExactMatch(nuclearplants$pt, z), data = nuclearplants)
table(nuclearplants$pt, aem)
```

---

as.InfinitySparseMatrix

Convert an object to InfinitySparseMatrix

Description

Convert an object to InfinitySparseMatrix

Usage

```
as.InfinitySparseMatrix(x)
```

Arguments

x	An object which can be coerced into InfinitySparseMatrix, typically a matrix.

Value

An InfinitySparseMatrix

as.list.BlockedInfinitySparseMatrix

Splits a BlockedInfinitySparseMatrix into a list of InfinitySparseMatrices

Description

Splits a BlockedInfinitySparseMatrix into a list of InfinitySparseMatrices

Usage

```
## S3 method for class 'BlockedInfinitySparseMatrix'
as.list(x, ...)
```

Arguments

x	a BlockedInfinitySparseMatrix
...	Ignored

Value

A list of InfinitySparseMatrices

BlockedInfinitySparseMatrix-class

Blocked Infinity Sparse Matrix

Description

Blocked Infinity Sparse Matrices are similar to Infinity Sparse Matrices, but they also keep track of the groups of units via an additional slot, `groups`

Slots

`groups` factor vector containing groups, with unit names as labels, when possible

`colnames` vector containing names for all control units. This will either be a character vector or NULL if units have no names

`rownames` vector containing names for all treated units. This will either be a character vector or NULL if units have no names

`cols` vector of integers corresponding to control units

`rows` vector of integers corresponding to treated units

`dimension` integer vector containing the number of treated and control units, in that order

`call` function call used to create the `InfinitySparseMatrix`

Author(s)

Mark M. Fredrickson

See Also

match_on, exactMatch, fullmatch, InfinitySparseMatrix-class

`c,SubProbInfo-method` *Combine objects*

Description

Combine objects

Usage

```
## S4 method for signature 'SubProbInfo'
c(x, ...)

## S4 method for signature 'NodeInfo'
c(x, ...)

## S4 method for signature 'ArcInfo'
c(x, ...)

## S4 method for signature 'MCFSolutions'
c(x, ...)

## S4 method for signature 'FullmatchMCFSolutions'
c(x, ...)
```

Arguments

x	object of particular class
...	Various objects

Value

Combined objects

c.optmatch *Combine Optmatch objects*

Description

Combine Optmatch objects

Usage

```
## S3 method for class 'optmatch'
c(...)
```

Arguments

...	Optmatch objects to be concatenated

Value

A combined Optmatch object

caliper *Prepare matching distances suitable for matching within calipers.*

Description

Encodes calipers, or maximum allowable distances within which to match. The result of a call to `caliper` is itself a distance specification between treated and control units that can be used with `pairmatch()` or `fullmatch()`. Calipers can also be combined with other distance specifications for richer matching problems.

Usage

```
caliper(x, width, exclude = c(), compare = `<=`, values = FALSE)

## S4 method for signature 'InfinitySparseMatrix'
caliper(x, width, exclude = c(), compare = `<=`, values = FALSE)

## S4 method for signature 'matrix'
caliper(x, width, exclude = c(), compare = `<=`, values = FALSE)

## S4 method for signature 'optmatch.dlist'
caliper(x, width, exclude = c(), compare = `<=`, values = FALSE)
```

Arguments

x	A distance specification created with match_on or similar.
width	The width of the caliper: how wide of a margin to allow in matches. Be careful in setting the width. Vector valued arguments will be recycled for each of the finite entries in x (and no order is guaranteed for x for some types of distance objects).
exclude	(Optional) A character vector of observations (corresponding to row and column names) to exclude from the caliper.
compare	A function that decides that whether two observations are with the caliper. The default is `<=`. `<` is a common alternative.
values	Should the returned object be made of all zeros (values = FALSE, the default) or should the object include the values of the original object (values = TRUE)?

Details

caliper is a generic function with methods for any of the allowed distance specifications: user created matrices, the results of match_on, the results of exactMatch, or combinations (using `+`) of these objects.

width provides the size of the caliper, the allowable distance for matching. If the distance between a treated and control pair is less than or equal to this distance, it is allowed kept; otherwise, the pair is discarded from future matching. The default comparison of "equal or less than can" be changed to any other comparison function using the comparison argument.

It is important to understand that width argument is defined on the scale of these distances. For univariate distances such as propensity scores, it is common to specify calipers in standard deviations. If a caliper of this nature is desired, you must either find the standard deviation directly or use the match_on function with its caliper argument. Since match_on has access to the underlying univariate scores, for example for the GLM method, it can determine the caliper width in standard deviations.

If you wish to exclude specific units from the caliper requirements, pass the names of these units in the exclude argument. These units will be allowed to match any other unit.

Value

A matrix like object that is suitable to be given as distance argument to fullmatch or pairmatch. The caliper will be only zeros and Inf values, indicating a possible match or no possible match, respectively.

You can combine the results of caliper with other distances using the `+` operator. See the examples for usage.

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

exactMatch, match_on, fullmatch, pairmatch

Examples

```
data(nuclearplants)


### Caliper of 100 MWe on plant capacity
caliper(match_on(pr~cap, data=nuclearplants, method="euclidean"), width=100)

### Caliper of 1/2 a pooled SD of plant capacity
caliper(match_on(pr~cap, data=nuclearplants), width=.5)

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

pptycaliper <- caliper(ppty.dist, width = .2)

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

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

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

### Returning values directly (equal up to the the attributes)
all(abs((caliper(ppty.dist, 1) + ppty.dist) -
        caliper(ppty.dist, 1, values = TRUE)) < .Machine$Double.eps)
```

---

cbind.InfinitySparseMatrix

Combine InfinitySparseMatrices or BlockedInfinitySparseMatrices by row or column

---

Description

This matches the syntax and semantics of cbind and rbind for matrices.

Usage

```
## S3 method for class 'InfinitySparseMatrix'
cbind(x, y, ...)
```

```
## S3 method for class 'InfinitySparseMatrix'
rbind(x, y, ...)

## S3 method for class 'BlockedInfinitySparseMatrix'
cbind(x, y, ...)

## S3 method for class 'BlockedInfinitySparseMatrix'
rbind(x, y, ...)
```

Arguments

x	An InfinitySparseMatrix or BlockedInfinitySparseMatrix, agreeing with y in the appropriate dimension.
y	An InfinitySparseMatrix or BlockedInfinitySparseMatrix, agreeing with x in the appropriate dimension.
...	Other arguments ignored.

Value

A combined InfinitySparseMatrix or BlockedInfinitySparseMatrix

Author(s)

Mark Fredrickson

---

compare_optmatch *Compares the equality of optmatch objects, ignoring attributes and group names.*

Description

This checks the equality of two optmatch objects. The only bits that matter are unit names and the grouping. Other bits such as attributes, group names, order, etc are ignored.

Usage

```
compare_optmatch(o1, o2)
```

Arguments

o1	First optmatch object.
o2	Second optmatch object.

Details

The names of the units can differ on any unmatched units, e.g., units whose value in the optmatch object is NA. If matched objects have differing names, this is automatically FALSE.

Note this ignores the names of the subgroups. So four members in subgroups either c("a", "a", "b", "b") or c("b", "b", "a", "a") would be identical to this call.

Value

TRUE if the two matches have the same memberships.

---

dbind *Diagonally bind together subgroup-specific distances*

Description

This function generates a single block-diagonal distance matrix given several distance matrices defined on subgroups.

Usage

```
dbind(..., force_unique_names = FALSE)
```

Arguments

...	Any number of distance objects which can be converted to InfinitySparseMatrix, such as class matrix, DenseMatrix, InfinitySparseMatrix, or BlockedInfinitySparseMatrix, or lists containing distance objects.
force_unique_names	Default FALSE. When row or column names are not unique among all distances, if FALSE, throw a warning and rename all rows and columns to ensure unique names. If TRUE, error on non-unique names.

Details

When you've generated several distances matrices on subgroups in your analysis, you may wish to combine them into a single block-diagonal distance matrix. The dbind function facilitates this.

Any BlockedInfinitySparseMatrix include in ... will be broken into individual InfinitySparseMatrix before being joined back together. For example, if b is a BlockedInfinitySparseMatrix with 2 subgroups and m is a distance without subgroups, then dbind(b, m) will be a BlockedInfinitySparseMatrix with 3 subgroups.

If there are any shared names (either row or column) among all distances passed in, by default all matrices will be renamed to ensure unique names by appending "X." to each distance, where "X" is ascending lower case letters ("a.", "b.", etc). Setting the force_unique_names argument to TRUE errors on this instead.

If the matrices need to be renamed and there are more than 26 separate matrices, after the first 26 single "X." prefixs, they will continue as "YX.", e.g "aa.", "ab.", "ac.". If more than 676 separate

matrices, the prefix wil continue to "ZYX.", e.g. "aaa.", "aab.", "aac.". This scheme supports up to 18,278 unique matrices.

Note that you do **not** have to combine subgroup distances into a single blocked distance using this function to ultimately obtain a single matching set. Instead, take a look at the vignette `vignette("matching-within-subgro` `package = "optmatch")` for details on combining multiple matches.

Value

A `BlockedInfinitySparseMatrix` containing a block-diagonal distance matrix. If only a single distance is passed to `dbind` and it is not already a `BlockedInfinitySparseMatrix`, the result will be an `InfinitySparseMatrix` instead.

Examples

```
data(nuclearplants)
m1 <- match_on(pr ~ cost, data = subset(nuclearplants, pt == 0),
               caliper = 1)
m2 <- match_on(pr ~ cost, data = subset(nuclearplants, pt == 1),
               caliper = 1.3)
blocked <- dbind(m1, m2)

dists <- list(m1, m2)

blocked2 <- dbind(dists)
identical(blocked, blocked2)
```

---

dimnames,InfinitySparseMatrix-method

Get and set dimnames for InfinitySparseMatrix objects

Description

InfinitySparseMatrix objects represent sparse matching problems with treated units as rows of a matrix and controls units as the columns of the matrix. The names of the units can be retrieved and set using these methods.

Usage

```
## S4 method for signature 'InfinitySparseMatrix'
dimnames(x)

## S4 replacement method for signature 'InfinitySparseMatrix,list'
dimnames(x) <- value

## S4 replacement method for signature 'InfinitySparseMatrix,NULL'
dimnames(x) <- value
```

Arguments

x	An InfinitySparseMatrix object.
value	A list with two entries: the treated names and control names, respectively.

Value

A list with treated and control names.

---

distUnion *Combine multiple distance specifications into a single distance specification.*

Description

Creates a new distance specification from the union of two or more distance specifications. The constituent distances specifications may have overlapping treated and control units (identified by the rownames and colnames respectively).

Usage

```
distUnion(...)
```

Arguments

...	The distance specifications (as created with with match_on, exactMatch, or other distance creation function).

Details

For combining multiple distance specifications with common controls, but different treated units, rbind provides a way to combine the different objects. Likewise, cbind provides a way to combine distance specifications over common treated units, but different control units.

distUnion can combine distance units that have common treated and control units into a coherent single distance object. If there are duplicate treated-control entries in multiple input distances, the first entry will be used.

Value

An InfinitySparseMatrix object with all treated and control units from the arguments combined. Duplicate entries are resolved in favor of the earliest argument (e.g., distUnion(A, B) will favor entries in A over entries in B).

See Also

match_on, exactMatch, fullmatch, pairmatch, cbind, rbind

effectiveSampleSize *Compute the effective sample size of a match.*

Description

The effective sample size is the sum of the harmonic means of the number units in treatment and control for each matched group. For k matched pairs, the effective sample size is k. As matched groups become more unbalanced, the effective sample size decreases.

Usage

```
effectiveSampleSize(x, z = NULL)

## S3 method for class 'factor'
effectiveSampleSize(x, z = NULL)

## Default S3 method:
effectiveSampleSize(x, z = NULL)

## S3 method for class 'table'
effectiveSampleSize(x, z = NULL)
```

Arguments

x	An optmatch object, the result of fullmatch or pairmatch.
z	A treatment indicator, a vector the same length as match. This is only required if the match object does not contain the contrast.group' attribute.

Value

The equivalent number of pairs in this match.

See Also

summary.optmatch, stratumStructure

evaluate_primal *Compute value of primal problem given flows and arc costs*

Description

Compute value of primal problem given flows and arc costs

Usage

```
evaluate_primal(distances, solution)
```

Arguments

distances	An InfinitySparseMatrix giving distances
solution	A MCFSolutions object

Value

The value of the primal problem, i.e. sum of products of distances with flow along arcs in solution

Author(s)

Hansen

---

exactMatch *Generate an exact matching set of subproblems.*

Description

An exact match is one based on a factor. Within a level, all observations are allowed to be matched. An exact match can be combined with another distance matrix to create a set of matching subproblems.

Usage

```
exactMatch(x, ...)

## S4 method for signature 'vector'
exactMatch(x, treatment)

## S4 method for signature 'formula'
exactMatch(x, data = NULL, subset = NULL, na.action = NULL, ...)
```

Arguments

x	A factor vector or a formula, used to select method.
...	Additional arguments for methods.
treatment	A logical or binary vector the same length as x indicating treatment and control for each unit in the study. TRUE or 1 represents a treatment unit, FALSE or 0 represents a control unit. NA units are excluded.
data	A data.frame or matrix that contains the variables used in the formula x.
subset	an optional vector specifying a subset of observations to be used
na.action	A function which indicates what should happen when the data contain NAs

Details

exactMatch creates a block diagonal matrix of 0s and Infs. The pairs with 0 entries are within the same level of the factor and legitimate matches. Inf indicates units in different levels. exactMatch replaces the structure.fmla argument to several functions in previous versions of optmatch. For the factor method, the two vectors x and treatment must be the same length. The vector x is interpreted as indicating the grouping factors for the data, and the vector treatment indicates whether a unit is in the treatment or control groups. At least one of these two vectors must have names. For the formula method, the data argument may be omitted, in which case the method attempts to find the variables in the environment from which the function was called. This behavior, and the arguments subset and na.action, mimics the behavior of lm.

Value

A matrix like object, which is suitable to be given as distance argument to fullmatch or pairmatch. The exact match will be only zeros and Inf values, indicating a possible match or no possible match, respectively. It can be added to a another distance matrix to create a subclassed matching problem.

Author(s)

Mark M. Fredrickson

See Also

caliper, antiExactMatch, match_on, fullmatch, pairmatch

Examples

```
data(nuclearplants)

### First generate a standard propensity score
ppty <- glm(pr~.-(pr+cost), family = binomial(), data = nuclearplants)
ppty.distances <- match_on(ppty)

### Only allow matches within the partial turn key plants
pt.em <- exactMatch(pr ~ pt, data = nuclearplants)
as.matrix(pt.em)

### Blunt matches:
match.pt.em <- fullmatch(pt.em)
print(match.pt.em, grouped = TRUE)

### Combine the propensity scores with the subclasses:
match.ppty.em <- fullmatch(ppty.distances + pt.em)
print(match.ppty.em, grouped = TRUE)
```

`fill.NAs` *Create missingness indicator variables and non-informatively fill in missing values*

Description

Given a `data.frame` or `formula` and data, `fill.NAs()` returns an expanded data frame, including a new missingness flag for each variable with missing values and replacing each missing entry with a value representing a reasonable default for missing values in its column. Functions in the formula are supported, with transformations happening before `NA` replacement. The expanded data frame is useful for propensity modeling and balance checking when there are covariates with missing values.

Usage

```
fill.NAs(x, data = NULL, all.covs = FALSE, contrasts.arg = NULL)
```

Arguments

`x`	Can be either a data frame (in which case the data argument should be `NULL`) or a formula (in which case data must be a data.frame)
`data`	If x is a formula, this must be a data.frame. Otherwise it will be ignored.
`all.covs`	Should the response variable be imputed? For formula `x`, this is the variable on the left hand side. For `data.frame` `x`, the response is considered the first column.
`contrasts.arg`	(from `model.matrix`) A list, whose entries are values (numeric matrices or character strings naming functions) to be used as replacement values for the `contrasts` replacement function and whose names are the names of columns of `data` containing `factor`s.

Details

`fill.NAs` prepares data for use in a model or matching procedure by filling in missing values with minimally invasive substitutes. Fill-in is performed column-wise, with each column being treated individually. For each column that is missing, a new column is created of the form "ColumnName.NA" with indicators for each observation that is missing a value for "ColumnName". Rosenbaum and Rubin (1984, Sec. 2.4 and Appendix B) discuss propensity score models using this data structure.

The replacement value used to fill in a missing value is simple mean replacement. For transformations of variables, e.g. `y ~ x1 * x2`, the transformation occurs first. The transformation column will be `NA` if any of the base columns are `NA`. Fill-in occurs next, replacing all missing values with the observed column mean. This includes transformation columns.

Data can be passed to `fill.NAs` in two ways. First, you can simply pass a `data.frame` object and `fill.NAs` will fill every column. Alternatively, you can pass a `formula` and a `data.frame`. Fill-in will only be applied to columns specifically used in the formula. Prior to fill-in, any functions in the formula will be expanded. If any arguments to the functions are `NA`, the function value will also be `NA` and subject to fill-in.

By default, `fill.NAs` does not impute the response variable. This is to encourage more sophisticated imputation schemes when the response is a treatment indicator in a matching problem. This behavior can be overridden by setting `all.covs = TRUE`.

Value

A `data.frame` with all `NA` values replaced with mean values and additional indicator columns for each column including missing values. Suitable for directly passing to `lm` or other model building functions to build propensity scores.

Author(s)

Mark M. Fredrickson and Jake Bowers

References

Rosenbaum, Paul R. and Rubin, Donald B. (1984) 'Reducing Bias in Observational Studies using Subclassification on the Propensity Score,' *Journal of the American Statistical Association*, **79**, 516 – 524.

Von Hipple, Paul T. (2009) 'How to impute interactions, squares, and other transformed variables,' *Sociological Methodology*, **39**(1), 265 – 291.

See Also

`match_on`, `lm`

Examples

```
data(nuclearplants)
### Extract some representative covariates:
np.missing <- nuclearplants[c('t1', 't2', 'ne', 'ct', 'cum.n')]

 ### create some missingness in the covariates
 n <- dim(np.missing)[1]
 k <- dim(np.missing)[2]

 for (i in 1:n) {
   missing <- rbinom(1, prob = .1, size = k)
   if (missing > 0) {
     np.missing[i, sample(k, missing)] <- NA
   }
 }

### Restore outcome and treatment variables:
np.missing <- data.frame(nuclearplants[c('cost', 'pr')], np.missing)

### Fit a propensity score but with missing covariate data flagged
### and filled in, as in Rosenbaum and Rubin (1984, Appendix):
np.filled <- fill.NAs(pr ~ t1 * t2, np.missing)
# Look at np.filled to establish what missingness flags were created
head(np.filled)
```

```
(np.glm <- glm(pr ~ ., family=binomial, data=np.filled))
(glm(pr ~ t1 + t2 + `t1:t2` + t1.NA + t2.NA,
               family=binomial, data=np.filled))
# In a non-interactive session, the following may help, as long as
# the formula passed to `fill.NAs` (plus any missingness flags) is
# the desired formula for the glm.
(glm(formula(terms(np.filled)), family=binomial, data=np.filled))

### produce a matrix of propensity distances based on the propensity model
### with fill-in and flagging. Then perform pair matching on it:
pairmatch(match_on(np.glm, data=np.filled), data=np.filled)

## fill NAs without using treatment contrasts by making a list of contrasts for
## each factor ## following hints from https://stackoverflow.com/a/4569239/161808

np.missing$t1F<-factor(np.missing$t1)
cov.factors <- sapply(np.missing[,c("t1F","t2")],is.factor)
cov.contrasts <- lapply(
  np.missing[,names(cov.factors)[cov.factors],drop=FALSE],
  contrasts, contrasts = FALSE)

## make a data frame filling the missing covariate values, but without
## excluding any levels of any factors
np.noNA2<-fill.NAs(pr~t1F+t2,data=np.missing,contrasts.arg=cov.contrasts)
```

`findSubproblems` *List subproblems of a distance*

Description

Get all the subproblems from a distance specification

Usage

```
findSubproblems(d)
```

Arguments

`d`	a distance specification

Value

list of distance specifications

Author(s)

Mark M. Fredrickson

`fullmatch` *Optimal full matching*

Description

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

Usage

```
fullmatch(
  x,
  min.controls = 0,
  max.controls = Inf,
  omit.fraction = NULL,
  mean.controls = NULL,
  tol = 0.001,
  data = NULL,
  solver = "",
  ...
)

full(
  x,
  min.controls = 0,
  max.controls = Inf,
  omit.fraction = NULL,
  mean.controls = NULL,
  tol = 0.001,
  data = NULL,
  solver = "",
  ...
)
```

Arguments

`x`	Any valid input to `match_on`. `fullmatch` will use `x` and any optional arguments to generate a distance before performing the matching. If `x` is a numeric vector, there must also be passed a vector `z` indicating grouping. Both vectors must be named. Alternatively, a precomputed distance may be entered. A matrix of non-negative 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 distance specification as produced by `match_on`.

`min.controls` The minimum ratio of controls to treatments that is to be permitted within a matched set: should be non-negative 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 (such as those created by `exactMatch`), `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 subproblems `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 (such as those created by `exactMatch`), `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 neither `omit.fraction` or `mean.controls` are specified, then only those treated and control subjects without permissible matches among the control and treated subjects, respectively, are omitted.

When matching within subclasses (such as those created by `exactMatch`), `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.

At most one of `mean.controls` and `omit.fraction` can be non-`NULL`.

`mean.controls` Optionally, specify the average number of controls per treatment to be matched. Must be no less than than `min.controls` and no greater than the either `max.controls` or the ratio of total number of controls versus total number of treated. Some controls will likely not be matched to ensure meeting this value. If neither `omit.fraction` or `mean.controls` are specified, then only those treated and control subjects without permissible matches among the control and treated subjects, respectively, are omitted.

When matching within subclasses (such as those created by `exactMatch`), `mean.controls` specifies the average number of controls per treatment per subproblem, a parameter that can be made to differ by subclass by setting `mean.controls` equal to a named numeric vector.

At most one of `mean.controls` and `omit.fraction` can be non-`NULL`.

`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.

`data` Optional `data.frame` or `vector` to use to get order of the final matching factor. If a `data.frame`, the `rownames` are used. If a vector, the `names` are first tried, otherwise the contents is considered to be a character vector of names. Useful to pass if you want to combine a match (using, e.g., `cbind`) with the data that were used to generate it (for example, in a propensity score matching).

`solver` Choose which solver to use. Currently implemented are RELAX-IV and LEMON. Default of `""`, a blank string, will use RELAX-IV if the **rrelaxiv** package is installed, otherwise will use LEMON.

To explicitly use RELAX-IV, pass string "RELAX-IV".

To use LEMON, pass string "LEMON". Optionally, to specify which algorithm LEMON will use, pass the function LEMON with argument for the algorithm name, "CycleCancelling", "CapacityScaling", "CostScaling", and "NetworkSimplex". See this site for details on their differences: `https://lemon.cs.elte.hu/pub/doc/latest/a00606.html`. CycleCancelling is the default.

The CycleCancelling algorithm seems to produce results most closely resembling those of optmatch versions prior to 1.0. We have observed the other LEMON algorithms to produce different results when the `mean.controls` is unspecified, or specified in such a way as to produce an infeasible matching problem. When using a LEMON algorithm other than CycleCancelling, we recommend setting the `fullmatch_try_recovery` option to `FALSE`.

`...` Additional arguments, passed to `match_on` (e.g. `within`) or to specific methods.

Details

If passing an already created discrepancy matrix, finite entries indicate permissible matches, with smaller discrepancies indicating more desirable matches. The matrix must have row and column names.

If it is desirable to create the discrepancies matrix beforehand (for example, if planning on running several different matching schemes), consider using `match_on` to generate the distances. This generic function has several useful methods for handling propensity score models, computing Mahalanobis distances (and other arbitrary distances), and using user supplied functions. These distances can also be combined with those generated by `exactMatch` and `caliper` to create very nuanced matching specifications.

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.

By default, `fullmatch` will attempt, if the given constraints are infeasible, to find a feasible problem using the same constraints. This will almost surely involve using a more restrictive `omit.fraction` or `mean.controls`. (This will never automatically omit treatment units.) Note that this does not guarantee that the returned match has the least possible number of omitted subjects, it only gives a match that is feasible within the given constraints. It may often be possible to loosen the `omit.fraction` or `mean.controls` constraint and still find a feasible match. The auto recovery is controlled by `options("fullmatch_try_recovery")`.

In full matching problems permitting many-one matches (`min.controls` less than 1), the number of controls contributing to matches can exceed what was requested by setting a value of `mean.controls` or `omit.fraction`. I.e., in this setting `mean.controls` sets the minimum ratio of number of controls to number of treatments placed into matched sets.

If the program detects that (what it thinks is) a large problem, a warning is issued. Unless you have an older computer, there's a good chance that you can handle larger problems (at the cost of increased computation time). To check the large problem threshold, use `getMaxProblemSize`; to re-set it, use `setMaxProblemSize`.

Value

A `optmatch` object (`factor`) indicating matched groups.

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.

Examples

```
data(nuclearplants)
### Full matching on a Mahalanobis distance.
( fm1 <- fullmatch(pr ~ t1 + t2, data = nuclearplants) )
summary(fm1)

### Full matching with restrictions.
( fm2 <- fullmatch(pr ~ t1 + t2, min.controls = .5, max.controls = 4, data = nuclearplants) )
summary(fm2)

### Full matching to half of available controls.
( fm3 <- fullmatch(pr ~ t1 + t2, omit.fraction = .5, data = nuclearplants) )
summary(fm3)

### Full matching attempts recovery when the initial restrictions are infeasible.
### Limiting max.controls = 1 allows use of only 10 of 22 controls.
( fm4 <- fullmatch(pr ~ t1 + t2, max.controls = 1, data=nuclearplants) )
summary(fm4)
### To recover restrictions
optmatch_restrictions(fm4)

### 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.
### For more complicated models, create a distance matrix and pass it to fullmatch.
mhd <- match_on(pr ~ t1 + t2, data = nuclearplants) + caliper(match_on(ppty), width = 1)
( fm5 <- fullmatch(mhd, data = nuclearplants) )
```

```
summary(fm5)

### Propensity balance assessment. Requires RItools package.
if (require(RItools)) summary(fm5,ppty)

### The order of the names in the match factor is the same
### as the nuclearplants data.frame since we used the data argument
### when calling fullmatch. The order would be unspecified otherwise.
cbind(nuclearplants, matches = fm5)

### Match in subgroups only. There are a few ways to specify this.
m1 <- fullmatch(pr ~ t1 + t2, data=nuclearplants,
                within=exactMatch(pr ~ pt, data=nuclearplants))
m2 <- fullmatch(pr ~ t1 + t2 + strata(pt), data=nuclearplants)
### Matching on propensity scores within matching in subgroups only:
m3 <- fullmatch(glm(pr ~ t1 + t2, data=nuclearplants, family=binomial),
                data=nuclearplants,
                within=exactMatch(pr ~ pt, data=nuclearplants))
m4 <- fullmatch(glm(pr ~ t1 + t2 + pt, data=nuclearplants,
                    family=binomial),
                data=nuclearplants,
                within=exactMatch(pr ~ pt, data=nuclearplants))
m5 <- fullmatch(glm(pr ~ t1 + t2 + strata(pt), data=nuclearplants,
                    family=binomial), data=nuclearplants)
# Including `strata(foo)` inside a glm uses `foo` in the model as
# well, so here m4 and m5 are equivalent. m3 differs in that it does
# not include `pt` in the glm.
```

`getMaxProblemSize` *What is the maximum allowed problem size?*

Description

To prevent users from starting excessively large matching problems, the maximum problem size is limited by `options("optmatch_max_problem_size")`. This function a quick helper to assist fetching this value as a scalar. If the option isn't set, the function falls back to the default value, hard coded in the `optmatch` package.

Usage

```
getMaxProblemSize()
```

Value

logical

See Also

`options`, `setMaxProblemSize`

Examples

```
optmatch:::getMaxProblemSize() > 1 & optmatch:::getMaxProblemSize() < 1e100
```

`InfinitySparseMatrix-class`

Objects for sparse matching problems.

Description

`InfinitySparseMatrix` is a special class of distance specifications. Finite entries indicate possible matches, while infinite or NA entries indicated non-allowed matches. This data type can be more space efficient for sparse matching problems. Usually, users will create distance specification using `match_on`, `caliper`, or `exactMatch`. The ordering of units in an `InfinitySparseMatrix` is not guaranteed to be maintained after subsetting and/or other operations are performed.

Slots

`colnames` vector containing names for all control units. This will either be a character vector or NULL if units have no names

`rownames` vector containing names for all treated units. This will either be a character vector or NULL if units have no names

`cols` vector of integers corresponding to control units

`rows` vector of integers corresponding to treated units

`dimension` integer vector containing the number of treated and control units, in that order

`call` function call used to create the `InfinitySparseMatrix`

Author(s)

Mark M. Fredrickson

See Also

`match_on`, `caliper`, `exactMatch`, `fullmatch`, `pairmatch`

LEMON *(Internal) Helper function for accessing algorithms in LEMON solver*

Description

(Internal) Helper function for accessing algorithms in LEMON solver

Usage

```
LEMON(algorithm = "CycleCancelling")
```

Arguments

algorithm	LEMON algorithm to use. Choices are "CycleCancelling", "CapacityScaling", "CostScaling", "NetworkSimplex". Default is "CycleCancelling".

Value

String of the form "LEMON.<algorithm>"

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(x)

unmatched()

matchfailed(x)
```

Arguments

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

Details

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

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

```
data(plantdist)

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]
mxpl.exactmatch <- exactMatch(mxpl.si, c(rep(1, 7), rep(0, 26 - 7)))
# 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.
# only valid subproblems will be used

mcl <- c(1, Inf)

mxpl.fm2 <- fullmatch(plantdist + mxpl.exactmatch,
                      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

`matchobj`	Value of a call to `pairmatch` or `fullmatch`.
`distance`	Either a distance matrix or the value of a call to or `match_on`.
`preserve.unit.names`	Logical. If TRUE, for each matched set `matched.distances` returns the submatrix of the distance matrix corresponding to it; if FALSE, a vector containing the distances in that submatrix is returned.

Details

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

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))
```

match_on *Create treated to control distances for matching problems*

Description

A function with which to produce matching distances, for instance Mahalanobis distances, propensity score discrepancies or calipers, or combinations thereof, for pairmatch or fullmatch to subsequently "match on". Conceptually, the result of a call match_on is a treatment-by-control matrix of distances. Because these matrices can grow quite large, in practice match_on produces either an ordinary dense matrix or a special sparse matrix structure (that can make use of caliper and exact matching constraints to reduce storage requirements). Methods are supplied for these sparse structures, InfinitySparseMatrixes, so that they can be manipulated and modified in much the same way as dense matrices.

Usage

```
match_on(x, within = NULL, caliper = NULL, exclude = NULL, data = NULL, ...)

## S3 method for class 'glm'
match_on(
  x,
  within = NULL,
  caliper = NULL,
  exclude = NULL,
  data = NULL,
  standardization.scale = NULL,
  ...
)

## S3 method for class 'bigglm'
match_on(
  x,
  within = NULL,
  caliper = NULL,
  exclude = NULL,
  data = NULL,
  standardization.scale = NULL,
  ...
)

## S3 method for class 'formula'
match_on(
  x,
  within = NULL,
  caliper = NULL,
  exclude = NULL,
  data = NULL,
```

```
  subset = NULL,
  method = "mahalanobis",
  ...
)

## S3 method for class '`function`'
match_on(
  x,
  within = NULL,
  caliper = NULL,
  exclude = NULL,
  data = NULL,
  z = NULL,
  ...
)

## S3 method for class 'numeric'
match_on(x, within = NULL, caliper = NULL, exclude = NULL, data = NULL, z, ...)

## S3 method for class 'InfinitySparseMatrix'
match_on(x, within = NULL, caliper = NULL, exclude = NULL, data = NULL, ...)

## S3 method for class 'matrix'
match_on(x, within = NULL, caliper = NULL, exclude = NULL, data = NULL, ...)
```

Arguments

x	A model formula, fitted glm or other object implicitly specifying a distance; see blurbs on specific methods in Details.
within	A valid distance specification, such as the result of exactMatch or caliper. Finite entries indicate which distances to create. Including this argument can significantly speed up computation for sparse matching problems. Specify this filter either via within or via strata elements of a formula; mixing these methods will fail.
caliper	The width of a caliper to use to exclude treated-control pairs with values greater than the width. For some methods, there may be a speed advantage to passing a width rather than using the caliper function on an existing distance specification.
exclude	A list of units (treated or control) to exclude from the caliper argument (if supplied).
data	An optional data frame.
...	Other arguments for methods.
standardization.scale	Function for rescaling of scores(x), or NULL; defaults to mad. (See Details.)
subset	A subset of the data to use in creating the distance specification.
method	A string indicating which method to use in computing the distances from the data. The current possibilities are "mahalanobis", "euclidean" or "rank_mahalanobis".

z A logical or binary vector indicating treatment and control for each unit in the study. TRUE or 1 represents a treatment unit, FALSE of 0 represents a control unit. Any unit with NA treatment status will be excluded from the distance matrix.

Details

match_on is generic. There are several supplied methods, all providing the same basic output: a matrix (or similar) object with treated units on the rows and control units on the columns. Each cell [i,j] then indicates the distance from a treated unit i to control unit j. Entries that are Inf are said to be unmatchable. Such units are guaranteed to never be in a matched set. For problems with many Inf entries, so called sparse matching problems, match_on uses a special data type that is more space efficient than a standard R matrix. When problems are not sparse (i.e. dense), match_on uses the standard matrix type.

match_on methods differ on the types of arguments they take, making the function a one-stop location of many different ways of specifying matches: using functions, formulas, models, and even simple scores. Many of the methods require additional arguments, detailed below. All methods take a within argument, a distance specification made using exactMatch or caliper (or some additive combination of these or other distance creating functions). All match_on methods will use the finite entries in the within argument as a guide for producing the new distance. Any entry that is Inf in within will be Inf in the distance matrix returned by match_on. This argument can reduce the processing time needed to compute sparse distance matrices.

Details for each particular first type of argument follow:

First argument (x): glm. The model is assumed to be a fitted propensity score model. From this it extracts distances on the *linear* propensity score: fitted values of the linear predictor, the link function applied to the estimated conditional probabilities, as opposed to the estimated conditional probabilities themselves (Rosenbaum & Rubin, 1985). For example, a logistic model (glm with family=binomial()) has the logit function as its link, so from such models match_on computes distances in terms of logits of the estimated conditional probabilities, i.e. the estimated log odds.

Optionally these distances are also rescaled. The default is to rescale, by the reciprocal of an outlier-resistant variant of the pooled s.d. of propensity scores; see standardization_scale. (The standardization.scale argument of this function can be used to change how this dispersion is calculated, e.g. to calculate an ordinary not an outlier-resistant s.d.; it will be passed down to standardization_scale as its standardizer argument.) To skip rescaling, set argument standardization.scale to 1. The overall result records absolute differences between treated and control units on linear, possibly rescaled, propensity scores.

In addition, one can impose a caliper in terms of these distances by providing a scalar as a caliper argument, forbidding matches between treatment and control units differing in the calculated propensity score by more than the specified caliper. For example, Rosenbaum and Rubin's (1985) caliper of one-fifth of a pooled propensity score s.d. would be imposed by specifying caliper=.2, in tandem either with the default rescaling or, to follow their example even more closely, with the additional specification standardization.scale=sd. Propensity calipers are beneficial computationally as well as statistically, for reasons indicated in the below discussion of the numeric method.

One can also specify exactMatching criteria by using strata(foo) inside the formula to build the glm. For example, passing glm(y ~ x + strata(s)) to match_on is equivalent to passing within=exactMatch(y ~ strata(s)). Note that when combining with the caliper argument, the standard deviation used for the caliper will be computed across all strata, not within each strata.

If data used to fit the glm have missing values in the left-hand side (dependent) variable, these observations are omitted from the output of match_on. If there are observations with missing values in right hand side (independent) variables, then a re-fit of the model after imputing these variables using a simple scheme and adding indicator variables of missingness will be attempted, via the scores function.

First argument (x)**:** bigglm**.** This method works analogously to the glm method, but with bigglm objects, created by the bigglm function from package 'biglm', which can handle bigger data sets than the ordinary glm function can.

First argument (x)**:** formula**.** The formula must have Z, the treatment indicator (Z=0 indicates control group, Z=1 indicates treatment group), on the left hand side, and any variables to compute a distance on on the right hand side. E.g. Z ~ X1 + X2. The Mahalanobis distance is calculated as the square root of d'Cd, where d is the vector of X-differences on a pair of observations and C is an inverse (generalized inverse) of the pooled covariance of Xes. (The pooling is of the covariance of X within the subset defined by Z==0 and within the complement of that subset. This is similar to a Euclidean distance calculated after reexpressing the Xes in standard units, such that the reexpressed variables all have pooled SDs of 1; except that it addresses redundancies among the variables by scaling down variables contributions in proportion to their correlations with other included variables.)

Euclidean distance is also available, via method="euclidean", and ranked, Mahalanobis distance, via method="rank_mahalanobis".

The treatment indicator Z as noted above must either be numeric (1 representing treated units and 0 control units) or logical (TRUE for treated, FALSE for controls). (Earlier versions of the software accepted factor variables and other types of numeric variable; you may have to update existing scripts to get them to run.)

As an alternative to specifying a within argument, when x is a formula, the strata command can be used inside the formula to specify exact matching. For example, rather than using within=exactMatch(y ~ z, data=data), you may update your formula as y ~ x + strata(z). Do not use both methods (within and strata simultaneously. Note that when combining with the caliper argument, the standard deviation used for the caliper will be computed across all strata, not separately by stratum.

A unit with NA treatment status (Z) is ignored and will not be included in the distance output. Missing values in variables on the right hand side of the formula are handled as follows. By default match_on will (1) create a matrix of distances between observations which have only valid values for **all** covariates and then (2) append matrices of Inf values for distances between observations either of which has a missing values on any of the right-hand-side variables. (I.e., observations with missing values are retained in the output, but matches involving them are forbidden.)

First argument (x)**:** function**.** The passed function must take arguments: index, data, and z. The data and z arguments will be the same as those passed directly to match_on. The index argument is a matrix of two columns, representing the pairs of treated and control units that are valid comparisons (given any within arguments). The first column is the row name or id of the treated unit in the data object. The second column is the id for the control unit, again in the data object. For each of these pairs, the function should return the distance between the treated unit and control unit. This may sound complicated, but is simple to use. For example, a function that returned the absolute difference between two units using a vector of data would be f <- function(index, data, z) { abs(data[index[,1]] - data[index[,2]]) } . (Note: This simple case is precisely handled by the numeric method.)

First argument (x)**:** numeric**.** This returns absolute differences between treated and control units'

values of x. If a caliper is specified, pairings with x-differences greater than it are forbidden. Conceptually, those distances are set to Inf; computationally, if either of caliper and within has been specified then only information about permissible pairings will be stored, so the forbidden pairings are simply omitted. Providing a caliper argument here, as opposed to omitting it and afterward applying the caliper function, reduces storage requirements and may otherwise improve performance, particularly in larger problems.

For the numeric method, x must have names. If z is named it must have the same names as x, though it allows for a different ordering of names. x's name ordering is considered canonical.

First argument (x): matrix or InfinitySparseMatrix. These just return their arguments as these objects are already valid distance specifications.

Value

A distance specification (a matrix or similar object) which is suitable to be given as the distance argument to fullmatch or pairmatch.

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

fullmatch, pairmatch, exactMatch, caliper

scores

Examples

```
data(nuclearplants)
match_on.examples <- list()
### Propensity score distances.
### Recommended approach:
(aGlm <- glm(pr~.-(pr+cost), family=binomial(), data=nuclearplants))
match_on.examples$ps1 <- match_on(aGlm)
### A second approach: first extract propensity scores, then separately
### create a distance from them.  (Useful when importing propensity
### scores from an external program.)
plantsPS <- predict(aGlm)
match_on.examples$ps2 <- match_on(pr~plantsPS, data=nuclearplants)
### Full matching on the propensity score.
fm1 <- fullmatch(match_on.examples$ps1, data = nuclearplants)
fm2 <- fullmatch(match_on.examples$ps2, data = nuclearplants)
### Because match_on.glm uses robust estimates of spread,
### the results differ in detail -- but they are close enough
### to yield similar optimal matches.
all(fm1 == fm2) # The same

### Mahalanobis distance:
match_on.examples$mh1 <- match_on(pr ~ t1 + t2, data = nuclearplants)
```

```
### Absolute differences on a scalar:
tmp <- nuclearplants$t1
names(tmp) <- rownames(nuclearplants)

(absdist <- match_on(tmp, z = nuclearplants$pr,
                 within = exactMatch(pr ~ pt, nuclearplants)))

### Pair matching on the variable `t1`:
pairmatch(absdist, data = nuclearplants)


### Propensity score matching within subgroups:
match_on.examples$ps3 <- match_on(aGlm, exactMatch(pr ~ pt, nuclearplants))
fullmatch(match_on.examples$ps3, data = nuclearplants)

### Propensity score matching with a propensity score caliper:
match_on.examples$pscal <- match_on.examples$ps1 + caliper(match_on.examples$ps1, 1)
fullmatch(match_on.examples$pscal, data = nuclearplants) # Note that the caliper excludes some units

### A Mahalanobis distance for matching within subgroups:
match_on.examples$mh2 <- match_on(pr ~ t1 + t2 , data = nuclearplants,
                           within = exactMatch(pr ~ pt, nuclearplants))

### Mahalanobis matching within subgroups, with a propensity score
### caliper:
fullmatch(match_on.examples$mh2 + caliper(match_on.examples$ps3, 1), data = nuclearplants)

### Alternative methods to matching without groups (exact matching)
m1 <- match_on(pr ~ t1 + t2, data=nuclearplants, within=exactMatch(pr ~ pt, nuclearplants))
m2 <- match_on(pr ~ t1 + t2 + strata(pt), data=nuclearplants)
# m1 and m2 are identical

m3 <- match_on(glm(pr ~ t1 + t2 + cost, data=nuclearplants,
                   family=binomial),
               data=nuclearplants,
               within=exactMatch(pr ~ pt, data=nuclearplants))
m4 <- match_on(glm(pr ~ t1 + t2 + cost + pt, data=nuclearplants,
                   family=binomial),
               data=nuclearplants,
               within=exactMatch(pr ~ pt, data=nuclearplants))
m5 <- match_on(glm(pr ~ t1 + t2 + cost + strata(pt), data=nuclearplants,
                   family=binomial), data=nuclearplants)
# Including `strata(foo)` inside a glm uses `foo` in the model as
# well, so here m4 and m5 are equivalent. m3 differs in that it does
# not include `pt` in the glm.
```

`maxCaliper` *Find the maximum caliper width that will create a feasible problem.*

Description

Larger calipers permit more possible matches between treated and control groups, which can be better for creating matches with larger effective sample sizes. The downside is that wide calipers may make the matching problem too big for processor or memory constraints. `maxCaliper` attempts to find a caliper value, for a given vector of scores and a treatment indicator, that will be possible given the maximum problem size constraints imposed by `fullmatch` and `pairmatch`.

Usage

```
maxCaliper(scores, z, widths, structure = NULL, exact = TRUE)
```

Arguments

`scores`	A numeric vector of scores providing 1-D position of units
`z`	Treatment indicator vector
`widths`	A vector of caliper widths to try, will be sorted largest to smallest.
`structure`	Optional factor variable that groups the scores, as would be used by `exactMatch`. Including structure allows for wider calipers.
`exact`	A logical indicating if the exact problem size should be computed (`exact = TRUE`) or if a more computationally efficient upper bound should be used instead (`exact = FALSE`). The upper bound may lead to narrower calipers, even if wider calipers would have sufficed using the exact method.

Value

numeric The value of the largest caliper that creates a feasible problem. If no such caliper exists in `widths`, an error will be generated.

`maxControlsCap` *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

```
maxControlsCap(distance, min.controls = NULL, solver = "")

minControlsCap(distance, max.controls = NULL, solver = "")
```

Arguments

distance	Either a matrix of non-negative, numeric discrepancies, or a list of such matrices. (See fullmatch for details.)
min.controls	Optionally, set limits on the minimum number of controls per matched set. (Only makes sense for maxControlsCap.)
solver	Choose which solver to use. See help(fullmatch) for details.
max.controls	Optionally, set limits on the maximum number of controls per matched set. (Only makes sense for minControlsCap.)

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

---

mdist *(Deprecated, in favor of* match_on*) Create matching distances*

Description

Deprecated in favor of match_on

Usage

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

## S3 method for class 'optmatch.dlist'
mdist(x, structure.fmla = NULL, ...)

## S3 method for class '`function`'
mdist(x, structure.fmla = NULL, data = NULL, ...)

## S3 method for class 'formula'
mdist(x, structure.fmla = NULL, data = NULL, subset = NULL, ...)

## S3 method for class 'glm'
mdist(x, structure.fmla = NULL, standardization.scale = mad, ...)

## S3 method for class 'bigglm'
mdist(x, structure.fmla = NULL, data = NULL, standardization.scale = mad, ...)

## S3 method for class 'numeric'
mdist(x, structure.fmla = NULL, trtgrp = NULL, ...)
```

Arguments

x	The object to use as the basis for forming the mdist. Methods exist for formulas, functions, and generalized linear models.
structure.fmla	A formula denoting the treatment variable on the left hand side and an optional grouping expression on the right hand side. For example, z ~ 1 indicates no grouping. z ~ s subsets the data only computing distances within the subsets formed by s. See method notes, below, for additional formula options.
...	Additional method arguments. Most methods require a 'data' argument.
data	Data where the variables references in x live.
subset	If non-NULL, the subset of data to be used.
standardization.scale	A function to scale the distances; by default uses mad.
trtgrp	Dummy variable for treatment group membership.

Details

The `mdist` method provides three ways to construct a matching distance (i.e., a distance matrix or suitably organized list of such matrices): guided by a function, by a fitted model, or by a formula. The class of the first argument given to `mdist` determines which of these methods is invoked.

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` method computes the squared Mahalanobis distance between observations, with the right-hand side of the formula determining which variables contribute to the Mahalanobis distance. If matching is to be done within strata, the stratification can be communicated using either the `structure.fmla` argument (e.g. `~ grp`) or as part of the main formula (e.g. `z ~ x1 + x2 | grp`).

An `mdist.glm` method takes an argument of class `glm` as first argument. It assumes that this object is a fitted propensity model, extracting distances on the linear propensity score (logits of the estimated conditional probabilities) and, by default, rescaling the distances by the reciprocal of the pooled s.d. of treatment- and control-group propensity scores. (The scaling uses `mad`, for resistance to outliers, by default; this can be changed to the actual s.d., or rescaling can be skipped entirely, by setting argument `standardization.scale` to `sd` or `NULL`, respectively.) A `mdist.bigglm` method works analogously with `bigglm` objects, created by the `bigglm` function from package 'biglm', which can handle bigger data sets than the ordinary glm function can. In contrast with `mdist.glm` it requires additional `data` and `structure.fmla` arguments. (If you have enough data to have to use `bigglm`, then you'll probably have to subgroup before matching to avoid memory problems. So you'll have to use the `structure.fmla` argument anyway.)

Value

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

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

`fullmatch`, `pairmatch`, `match_on`

minExactMatch *Find the minimal exact match factors that will be feasible for a given maximum problem size.*

Description

The exactMatch function creates a smaller matching problem by stratifying observations into smaller groups. For a problem that is larger than maximum allowed size, minExactMatch provides a way to find the smallest exact matching problem that will allow for matching.

Usage

```
minExactMatch(x, scores = NULL, width = NULL, maxarcs = 1e+07, ...)
```

Arguments

x	The object for dispatching.
scores	Optional vector of scores that will be checked against a caliper width.
width	Optional width of a caliper to place on the scores.
maxarcs	The maximum problem size to attempt to fit.
...	Additional arguments for methods.

Details

x is a formula of the form Z ~ X1 + X2, where Z is indicates treatment or control status, and X1 and X2 are variables can be converted to factors. Any additional arguments are passed to model.frame (e.g., a data argument containing Z, X1, and X2).

The the arguments scores and width must be passed together. The function will apply the caliper implied by the scores and the width while also adding in blocking factors.

Value

A factor grouping units, suitable for exactMatch.

nuclearplants *Nuclear Power Station Construction Data*

Description

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

A data frame with 32 rows and 11 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.

`num_eligible_matches` *Returns the number of eligible matches for the distance.*

Description

This will return a list of the number of finite entries in a distance matrix. If the distance has no subgroups, it will be a list of length 1. If the distance has subgroups (i.e. `x` is an `BlockedInfinitySparseMatrix`, it will be a named list.)

Usage

```
num_eligible_matches(x)

## S3 method for class 'optmatch.dlist'
num_eligible_matches(x)

## S3 method for class 'matrix'
num_eligible_matches(x)

## S3 method for class 'InfinitySparseMatrix'
num_eligible_matches(x)

## S3 method for class 'BlockedInfinitySparseMatrix'
num_eligible_matches(x)
```

Arguments

x	Any distance object.

Value

A list counting the number of eligible matches in the distance.

`optmatch` *Optmatch Class*

Description

The `optmatch` class describes the results of an optimal full matching (using either `fullmatch` or `pairmatch`). For the most part, these objects can be treated as `factors`.

The summary function quantifies `optmatch` objects on the effective sample size, the distribution of distances between matched units, and how well the match reduces average differences.

Usage

```
## S3 method for class 'optmatch'
summary(
  object,
  propensity.model = NULL,
  ...,
  min.controls = 0.2,
  max.controls = 5,
  quantiles = c(0, 0.5, 0.95, 1)
)
```

Arguments

object	The optmatch object to summarize.
propensity.model	An optional propensity model (the result of a call to glm) to use when summarizing the match. If the **RItools** package is installed, an additional chi-squared test will be performed on the average differences between treated and control units on each variable used in the model. See the xBalance function in the **RItools** package for more details.
...	Additional arguments to pass to xBalance when also passing a propensity model.
min.controls	To minimize the the display of a groups with many treated and few controls, all groups with more than 5 treated units will be summarized as "5+". This is the reciprocal of the default value (1/5 = 0.2). Lower this value to see more groups.
max.controls	Like min.controls sets maximum group sized displayed with respect to the number of controls. Raise this value to see more groups.
quantiles	A points in the ECDF at which the distances between units will be displayed.

Details

optmatch objects descend from 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 (as encoded using exactMatch) (ii) the string .; and (iii) a non-negative integer. 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.)

Value

optmatch.summary

See Also

print.optmatch

optmatch-defunct *Functions deprecated or removed from optmatch*

Description

Over the course of time, several functions in optmatch have been removed in favor of new interfaces and functions.

Usage

```
pscore.dist(...)

mahal.dist(...)
```

Arguments

...	All arguments ignored.

See Also

mdist, match_on

optmatch_restrictions *optmatch_restrictions*

Description

Returns the restrictions which were used to generate the match.

Usage

```
optmatch_restrictions(obj)
```

Arguments

obj	An optmatch object

Details

If `mean.controls` was explicitly specified in the creation of the optmatch object, it is returned; otherwise `omit.fraction` is given.

Note that if the matching algorithm attempted to recover from initial infeasible restrictions, the output from this function may not be the same as the original function call.

Value

A list of `min.controls`, `max.controls` and either `omit.fraction` or `mean.controls`.

optmatch_same_distance

Checks if two distances are equivalent. `x` *and* `y` *can be distances (*`InfinitySparseMatrix`*,* `BlockedInfinitySparseMatrix`*, or* `DenseMatrix`*), or they can be* `optmatch` *objects.*

Description

To save space, `optmatch` objects merely store a hash of the distance matrix instead of the original object. Any distance objects are hashed before comparison.

Usage

```
optmatch_same_distance(x, y)
```

Arguments

`x`	A distances (`InfinitySparseMatrix`, `BlockedInfinitySparseMatrix`, or `DenseMatrix`), or `optmatch` object.
`y`	A distances (`InfinitySparseMatrix`, `BlockedInfinitySparseMatrix`, or `DenseMatrix`), or `optmatch` object.

Details

Note that the distance is hashed with its `call` set to `NULL`. (This avoids issues where, for example, `match_on(Z~X, data=d,caliper=NULL)` and `match_on(Z~X, data=d)` produce identical matches but have differing `call`s.)

Value

Boolean whether the two distance specifications are identical.

pairmatch *Optimal 1:1 and 1:k matching*

Description

Given a treatment group, a larger control reservoir, and a method for creating discrepancies between each treatment and control unit (or optionally an already created such discrepancy matrix), finds a pairing of treatment units to controls that minimizes the sum of discrepancies.

Usage

```
pairmatch(x, controls = 1, data = NULL, remove.unmatchables = FALSE, ...)

pair(x, controls = 1, data = NULL, remove.unmatchables = FALSE, ...)
```

Arguments

x	Any valid input to match_on. If x is a numeric vector, there must also be passed a vector z indicating grouping. Both vectors must be named. Alternatively, a precomputed distance may be entered.
controls	The number of controls to be matched to each treatment
data	Optional data set.
remove.unmatchables	Should treatment group members for which there are no eligible controls be removed prior to matching?
...	Additional arguments to pass to match_on (e.g. within)) or to fullmatch (e.g. tol). It is an error to pass min.controls, max.controls, mean.controls or omit.fraction as pairmatch must set these values.

Details

This is a wrapper to fullmatch; see its documentation for more information, especially on additional arguments to pass, additional discussion of valid input for parameter x, and feasibility recovery.

If remove.unmatchables is FALSE, then if there are unmatchable treated units then the matching as a whole will fail and no units will be matched. If TRUE, then this unit will be removed and the function will attempt to match each of the other treatment units. As of version 0.9-8, if there are fewer matchable treated units than matchable controls then pairmatch will attempt to place each into a matched pair each of the matchable controls and a strict subset of the matchable treated units. (Previously matching would have failed for subclasses of this structure.)

Matching can still fail, even with remove.unmatchables set to TRUE, if there is too much competition for certain controls; if you find yourself in that situation you should consider full matching, which necessarily finds a match for everyone with an eligible match somewhere.

The units of the optmatch object returned 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 (with possible additions from the optional data argument). 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 non-negative integer. Unmatched units have NA entries. 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 by summary.optmatch.)

Value

A optmatch object (factor) indicating matched groups.

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

Examples

```
data(nuclearplants)

### Pair matching on a Mahalanobis distance
( pm1 <- pairmatch(pr ~ t1 + t2, data = nuclearplants) )
summary(pm1)

### Pair matching within a propensity score caliper.
ppty <- glm(pr ~ . - (pr + cost), family = binomial(), data = nuclearplants)
### For more complicated models, create a distance matrix and pass it to fullmatch.
mhd <- match_on(pr ~ t1 + t2, data = nuclearplants) + caliper(match_on(ppty), 2)
( pm2 <- pairmatch(mhd, data = nuclearplants) )
summary(pm2)

### Propensity balance assessment. Requires RItools package.
if(require(RItools)) summary(pm2, ppty)

### 1:2 matched triples
( tm <- pairmatch(pr ~ t1 + t2, controls = 2, data = nuclearplants) )
summary(tm)

### Creating a data frame with the matched sets attached.
### match_on(), 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)])


### Match in subgroups only. There are a few ways to specify this.
m1 <- pairmatch(pr ~ t1 + t2, data=nuclearplants,
                within=exactMatch(pr ~ pt, data=nuclearplants))
m2 <- pairmatch(pr ~ t1 + t2 + strata(pt), data=nuclearplants)
### Matching on propensity scores within matching in subgroups only:
m3 <- pairmatch(glm(pr ~ t1 + t2, data=nuclearplants, family=binomial),
                data=nuclearplants,
                within=exactMatch(pr ~ pt, data=nuclearplants))
m4 <- pairmatch(glm(pr ~ t1 + t2 + pt, data=nuclearplants,
```

```
                    family=binomial),
                data=nuclearplants,
                within=exactMatch(pr ~ pt, data=nuclearplants))
m5 <- pairmatch(glm(pr ~ t1 + t2 + strata(pt), data=nuclearplants,
                    family=binomial), data=nuclearplants)
# Including `strata(foo)` inside a glm uses `foo` in the model as
# well, so here m4 and m5 are equivalent. m3 differs in that it does
# not include `pt` in the glm.
```

`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

```
plantdist
```

Format

A matrix with 7 rows and 19 columns

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).

`predict.CBPS` *(Internal) Predict for CBPS objects*

Description

The CBPS package fits 'covariate balancing propensity score' for use in propensity score weighting. In the binary treatment case they can also be used for matching. This method helps to implement that process in a manner consistent with use of propensity scores elsewhere in optmatch; see `scores` documentation.

Usage

```
## S3 method for class 'CBPS'
predict(object, newdata = NULL, type = c("link", "response"), ...)
```

Arguments

object	A CBPS object
newdata	Unused.
type	Return inverse logits of fitted values (the default) or fitted values themselves
...	Unused.

Value

Inverse logit of the fitted values.

---

print.optmatch *Printing* optmatch *objects.*

Description

Printing optmatch objects.

Usage

```
## S3 method for class 'optmatch'
print(x, quote = FALSE, grouped = FALSE, ...)
```

Arguments

x	The optmatch object, as returned by fullmatch or pairmatch.
quote	A boolean indicating if the matched group names should be quoted or not (default is not to quote).
grouped	A logical indicating if the object should printed in the style of a named factor object (grouped = TRUE) or as a table of group names and members.
...	Arguments passed to print.default.

See Also

fullmatch, pairmatch, print, summary.optmatch

Examples

```
data(nuclearplants)
fm <- fullmatch(pr ~ t1 + t2, data = nuclearplants)

print(fm)
print(fm, grouped = TRUE)
```

scoreCaliper — *(Internal) Helper function to create an InfinitySparseMatrix from a set of scores, a treatment indicator, and a caliper width.*

Description

(Internal) Helper function to create an InfinitySparseMatrix from a set of scores, a treatment indicator, and a caliper width.

Usage

```
scoreCaliper(x, z, caliper)
```

Arguments

x	The scores, a vector indicating the 1-D location of each unit.
z	The treatment assignment vector (same length as x)
caliper	The width of the caliper with respect to the scores x.

Value

An InfinitySparseMatrix object, suitable to be passed to match_on as an within argument.

scores — *Extract scores (propensity, prognostic,...) from a fitted model*

Description

This is a wrapper for predict, adapted for use in matching. Given a fitted model but no explicit newdata to 'predict' from, it constructs its own newdata in a manner that's generally better suited for matching.

Usage

```
scores(object, newdata = NULL, ...)
```

Arguments

object	fitted model object determining scores to be generated.
newdata	(optional) data frame containing variables with which scores are produced.
...	additional arguments passed to predict.

Details

Like predict, its default predictions from a glm are on the scale of the linear predictor, not the scale of the response; see Rosenbaum \ Rubin (1985). (This default can be overridden by specifying type="response".) In contrast to predict, if scores isn't given an explicit newdata argument then it attempts to reconstruct one from the context in which it is called, rather than from its first argument. For example, if it's called within the formula argument of a call to glm, its newdata is the same data frame that glm evaluates that formula in, as opposed to the model frame associated with object. See Examples.

The handling of missing independent variables also differs from that of predict in two ways. First, if the data used to generate object has NA values, they're mean-imputed using fill.NAs. Secondly, if newdata (either the explicit argument, or the implicit data generated from object) has NA values, they're likewise mean-imputed using fill.NAs. Also, missingness flags are added to the formula of object, which is then re-fit, using fill.NAs, prior to calling predict.

If newdata is specified and contains no missing data, scores returns the same value as predict.

Value

See individual predict functions.

Author(s)

Josh Errickson

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

predict

Examples

```
data(nuclearplants)
pg <- lm(cost~., data=nuclearplants, subset=(pr==0))
# The following two lines produce identical results.
ps1 <- glm(pr~cap+date+t1+bw+predict(pg, newdata=nuclearplants),
           data=nuclearplants)
ps2 <- glm(pr~cap+date+t1+bw+scores(pg), data=nuclearplants)
```

`setMaxProblemSize` *Set the maximum problem size*

Description

Helper function to ease setting the largest problem size to be accepted by `pairmatch` or `fullmatch`.

Usage

```
setMaxProblemSize(size = Inf)
```

Arguments

`size`	Positive integer, or `Inf`

Details

The function sets the optmatch_max_problem_size global option. The option ships with the option pre-set to a value that's relatively small, smaller than what most modern computers can handle. Invoking this function with no argument re-sets the optmatch_max_problem_size option to `Inf`, effectively disabling checks on problem size. Unless you're working with an older computer, it probably makes sense for most users to do this, at least until they determine what problem sizes are too large for their machines. (You'll know that when your R crashes, or simply takes too long for your taste.)

To determine the size of a problem without subproblems, i.e. exact matching categories, use `match_on` to set up and store the problem distance, then apply `length` to the result. If there were exact matching constraints imposed during the creation of the distance, then you'll want to look at the largest size of a subproblem. Apply `findSubproblems` to your distance, creating a list, say `dlist`, of your distances; then do `sapply(dlist, length)` to determine the sizes of the subproblems.

Author(s)

Ben B. Hansen

See Also

`getMaxProblemSize`

show,BlockedInfinitySparseMatrix-method *Displays a BlockedInfinitySparseMatrix*

Description

Displays each block of the BlockedInfinitySparseMatrix separately.

Usage

```
## S4 method for signature 'BlockedInfinitySparseMatrix'
show(object)
```

Arguments

object	An BlockedInfinitySparseMatrix to print.

show,InfinitySparseMatrix-method *Displays an InfinitySparseMatrix*

Description

Specifically, displays an InfinitySparseMatrix by converting it to a matrix first.

Usage

```
## S4 method for signature 'InfinitySparseMatrix'
show(object)
```

Arguments

object	An InfinitySparseMatrix to print.

sort.InfinitySparseMatrix *Sort the internal structure of an InfinitySparseMatrix.*

Description

Internally, an `InfinitySparseMatrix` (Blocked or non) comprises of vectors of values, row positions, and column positions. The ordering of these vectors is not enforced. This function sorts the internal structure, leaving the external structure unchanged (e.g. `as.matrix(ism)` and `as.matrix(sort(ism))` will look identical despite sorting.)

Usage

```
## S3 method for class 'InfinitySparseMatrix'
sort(x, decreasing = FALSE, ..., byCol = FALSE)

## S3 method for class 'BlockedInfinitySparseMatrix'
sort(x, decreasing = FALSE, ..., byCol = FALSE)
```

Arguments

`x`	An `InfinitySparseMatrix` or `BlockedInfinitySparseMatrix`.
`decreasing`	Logical. Should the sort be increasing or decreasing? Default `FALSE`.
`...`	Additional arguments ignored.
`byCol`	Logical. Defaults to `FALSE`, so the returned ISM is row-dominant. `TRUE` returns a column-dominant ISM.

Details

By default, the `InfinitySparseMatrix` is row-dominant, meaning the row positions are sorted first, then column positions are sorted within each row. Use argument `byCol` to change this.

Value

An object of the same class as `x` which is sorted according to `byCol`.

strata *Identify Stratafication Variables*

Description

This is a special function used only in identifying the strata variables when defining an `exactMatch` during a call to `fullmatch`, `pairmatch`, or `match_on`. It should not be called externally.

Usage

```
strata(...)
```

Arguments

...	any number of variables of the same length.

Value

the variables with appropriate labels

Examples

```
data(nuclearplants)
fullmatch(pr ~ cost + strata(pt), data = nuclearplants)
```

---

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)

## S3 method for class 'optmatch'
stratumStructure(stratum, trtgrp, min.controls = 0, max.controls = Inf)

## Default S3 method:
stratumStructure(stratum, trtgrp, min.controls = 0, max.controls = Inf)

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

Arguments

stratum	Matched strata, as returned by fullmatch or pairmatch
trtgrp	Dummy variable for treatment group membership. (Not required if stratum is an optmatch object, as returned by fullmatch or pairmatch.)
min.controls	For display, the number of treatment group members per stratum will be truncated at the reciprocal of min.controls.
max.controls	For display, the number of control group members will be truncated at max.controls.
x	stratumStructure object to be printed.
...	Additional arguments to print.

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 $2\sigma/\sum_s h_s$. This motivates the use of $\sum_s h_s$ as a measure of effective sample size (Hansen, 2011). Somewhat different motivations of the same calculation appear in Hansen (2004) and in Hansen and Bowers (2008). 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.

Author(s)

Ben B. 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** (2), 219–236.

Hansen, B.B. (2011), 'Propensity score matching to extract latent experiments from nonexperimental data: A case study'. Ch. 9 of *Looking Backwards: Proceedings from a Conference in Honor of Paul W. Holland*, Springer.

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)
```

---

subdim *Returns the dimension of each valid subproblem*

Description

Returns a list containing the dimensions of all valid subproblems.

Usage

```
subdim(x)

## S3 method for class 'InfinitySparseMatrix'
subdim(x)

## S3 method for class 'matrix'
subdim(x)

## S3 method for class 'BlockedInfinitySparseMatrix'
subdim(x)

## S3 method for class 'optmatch.dlist'
subdim(x)
```

Arguments

x	A distance specification to get the sub-dimensions of.

Value

A data frame listing the dimensions of each valid subproblem. Any subproblems with 0 controls or 0 treatments will be ignored. The names of the entries in the list will be the names of the subproblems, if they exist. There will be two rows, named "treatments" and "controls".

Examples

```
em <- exactMatch(pr ~ pt, data=nuclearplants)
m1 <- fullmatch(pr ~ t1 + t2, within=em, data=nuclearplants)
stratumStructure(m1)
(subdims_em <- subdim(em))
m2 <- fullmatch(pr ~ t1 + t2, within=em, data=nuclearplants,
            mean.controls=pmin(1.5, subdims_em["controls",] / subdims_em["treatments",])
                )
stratumStructure(m2)
```

`subset.InfinitySparseMatrix`
Subsetting for InfinitySparseMatrices

Description

This matches the syntax and semantics of subset for matrices.

Usage

```
## S3 method for class 'InfinitySparseMatrix'
subset(x, subset, select, ...)

## S4 method for signature 'InfinitySparseMatrix'
x[i, j = NULL, ..., drop = TRUE]

## S4 replacement method for signature 'InfinitySparseMatrix'
x[i, j] <- value
```

Arguments

`x`	InfinitySparseMatrix to be subset or bound.
`subset`	Logical expression indicating rows to keep.
`select`	Logical expression indicating columns to keep.
`...`	Other arguments are ignored.
`i`	Row indices.
`j`	Col indices.
`drop`	Ignored.
`value`	replacement values

Value

An InfinitySparseMatrix with only the selected elements.

Author(s)

Mark Fredrickson

summary.ism *Summarize a distance matrix*

Description

Given a distance matrix, return information above it, including dimension, sparsity information, unmatchable members, summary of finite distances, and, in the case of `BlockedInfinitySparseMatrix`, block structure.

Usage

```
## S3 method for class 'InfinitySparseMatrix'
summary(object, ..., distanceSummary = TRUE)

## S3 method for class 'BlockedInfinitySparseMatrix'
summary(
  object,
  ...,
  distanceSummary = TRUE,
  printAllBlocks = FALSE,
  blockStructure = TRUE
)

## S3 method for class 'DenseMatrix'
summary(object, ..., distanceSummary = TRUE)
```

Arguments

`object`	A `InfinitySparseMatrix`, `BlockedInfinitySparseMatrix` or `DenseMatrix`.
`...`	Ignored.
`distanceSummary`	Default `TRUE`. Should a summary of minimum distance per treatment member be calculated? May be slow on larger data sets.
`printAllBlocks`	If `object` is a `BlockedInfinitySparseMatrix`, should summaries of all blocks be printed alongside the overall summary? Default `FALSE`.
`blockStructure`	If `object` is a `BlockedInfinitySparseMatrix` and `printAllBlocks` is false, print a quick summary of each individual block. Default `TRUE`. If the number of blocks is high, consider suppressing this.

Details

The output consists of several pieces.

- Membership: Indicates the dimension of the distance.

- Total (in)eligible potential matches: A measure of the sparsity of the distance. Eligible matches have a finite distance between treatment and control members; they could be matched. Ineligible matches have `Inf` distance and can not be matched. A higher number of ineligible matches can speed up matching, but runs the risk of less optimal overall matching results.
- Unmatchable treatment/control members: If any observations have no eligible matches (e.g. their distance to every potential match is `Inf`) they are listed here. See Value below for details of how to access lists of matchable and unmatchable treatment and control members.
- Summary of minimum matchable distance per treatment member: To assist with choosing a caliper, this is a numeric summary of the smallest distance per matchable treatment member. If you provide a caliper that is less than the maximum value, at least one treatment member will become unmatchable.
- Block structure: For `BlockedInfinitySparseMatrix`, a quick summary of the structure of each individual block. (The above will all be across all blocks.) This may indicate which blocks, if any, are problematic.

Value

A named `list`. The summary for an `InfinitySparseMatrix` or `DenseMatrix` contains the following:

`total:`	Contains the total number of treatment and control members, as well as eligible and ineligible matches.
`matchable:`	The names of all treatment and control members with at least one eligible match.
`unmatchable:`	The names of all treatment and control members with no eligible matches.
`distances:`	The summary of minimum matchable distances, if `distanceSummary` is `TRUE`.

For `BlockedInfinitySparseMatrix`, the named `list` instead of contains one entry per block, named after each block (i.e. the value of the blocking variable) as well as a block named 'overall' which contains the summary ignoring blocks. Each of these entries contains a `list` with entries 'total', 'matchable', 'unmatchable' and 'distances', as described above.

`update.optmatch` *Performs an update on an* `optmatch` *object.*

Description

NB: THIS CODE IS CURRENTLY VERY MUCH ALPHA AND SOMEWHAT UNTESTED, ESPECIALLY CALLING `update` ON AN OPTMATCH OBJECT CREATED WITHOUT AN EXPLICIT DISTANCE ARGUMENT.

Usage

```
## S3 method for class 'optmatch'
update(object, ...)
```

Arguments

object	Optmatch object to update.
...	Additional arguments to the call, or arguments with changed values.

Details

Note that passing data again is strongly recommended. A warning will be printed if the hash of the data used to generate the optmatch object differs from the hash of the new data.

To obtain an updated call without performing the actual update, pass an additional evaluate = FALSE argument.

Value

An updated optmatch object.

Index