Package ‘optmatch’

February 20, 2015

Version 0.9-3
Date 2014-08-12
Title Functions for optimal matching
Author Ben B. Hansen <ben.hansen@umich.edu>, Mark Fredrickson
<mark.m.fredrickson@gmail.com>, Josh Buckner, Josh Errickson, and Peter Solenberger, with embedded Fortran code due to Dimitri P. Bertsekas
<dimitrib@mit.edu> and Paul Tseng
Maintainer Mark M. Fredrickson <mark.m.fredrickson@gmail.com>
Depends R (>= 2.15.1), stats, methods, graphics, digest
Suggests boot, Rltools, biglm, testthat, survival, brglm, arm
Description Functions for optimal matching, including full matching
License file LICENSE
URL http://www.r-project.org,
https://github.com/markmfredrickson/optmatch
Collate 'abs.optmatch.dlist.R' 'boxplot-methods.R' 'DenseMatrix.R'
'InfinitySparseMatrix.R' 'caliper.R' 'deprecated.R'
'distUnion.R' 'exactMatch.R' 'feasible.R' 'fill.NAs.R'
'fmatch.R' 'fullmatch.R' 'makedist.R' 'matched.distances.R'
'matched.R' 'matchfailed.R' 'match_on.R' 'max.controls.cap.R'
'mdist.R' 'min.controls.cap.R' 'Ops.optmatch.dlist.R'
'Optmatch.R' 'pairmatch.R' 'print.optmatch.dlist.R'
'print.optmatch.R' 'relaxinfo.R' 'scores.R'
'stratumStructure.R' 'subDivStrat.R' 'summary.optmatch.R'
'utilities.R' 'zzzDistanceSpecification.R' 'zzz.R'

NeedsCompilation yes
Repository CRAN
Date/Publication 2014-08-13 08:49:28
R topics documented:

antiExactMatch .................................................. 3
caliper ............................................................. 4
caliperSize .......................................................... 6
caliperUpperBound ............................................... 7
compare_optmatch ................................................ 7
distUnion ............................................................ 8
dist_digest ........................................................... 9
effectiveSampleSize ............................................... 9
exactMatch-methods .............................................. 10
fill.NAs ............................................................... 11
fmla2treatedblocking ........................................... 14
fullmatch ............................................................ 14
getMaxProblemSize .............................................. 17
makeInfinitySparseMatrix ...................................... 18
makeOptmatch ....................................................... 20
matched ............................................................. 21
matched.distances ............................................... 23
match_on ............................................................. 24
maxCaliper ........................................................... 28
mdist ................................................................. 29
minControlsCap ................................................... 31
minExactMatch ..................................................... 33
nuclearplants ..................................................... 34
num_eligible_matches ........................................... 35
optmatch ............................................................. 36
optmatch-defunct .................................................. 37
optmatch_restrictions .......................................... 38
optmatch_same_distance ........................................ 38
pairmatch ........................................................... 39
plantdist ............................................................. 41
print.optmatch ..................................................... 42
relaxinfo ............................................................. 42
scoreCaliper ........................................................ 43
scores ................................................................. 43
setFeasibilityConstants ........................................ 44
setTryRecovery .................................................... 45
stratumStructure ................................................ 45
subdim ............................................................... 47
subproblemSuccess .............................................. 48
update.optmatch .................................................. 48
validDistanceSpecification .................................... 49

Index 50
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 treatment indicator factor (a numeric vector of 1 and 0, a logical vector, or a 2 level factor).

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)
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

```r
## S4 method for signature 'InfinitySparseMatrix'
caliper(x, width = 1,
    exclude = c(), compare = '<=', values = FALSE)

## S4 method for signature 'matrix'
caliper(x, width = 1, exclude = c(),
    compare = '<=', values = FALSE)

## S4 method for signature 'optmatch.dlist'
caliper(x, width = 1,
    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`)?

```r
aem <- fullmatch(antiExactMatch(nuclearplants$pt, z), data = nuclearplants)
table(nuclearplants$pt, aem)
```
caliper

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.

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


See Also

`exactMatch, match_on, fullmatch, pairmatch`

Examples

data(nuclearplants)

```r
### Caliper of 100 MW\(e\) 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)
```
### caliperSize

*(Internal) Determines how many other units fall within a caliper distance*

**Description**

The matching functions `fullmatch` and `pairmatch` have a maximum problem size, based on the number of comparisons between treated and control units. For a completely dense problem, in which every treated unit is compared to every control unit there are \( \text{length(treated)} \times \text{length(control)} \) comparisons. A caliper restricts which comparisons are valid, disallowing matches of treated and control pairs that are too far apart. A caliper can significantly decrease the size of a matching problem. The `caliperSize` function reports exactly who many valid treated-control comparisons remain after applying a caliper of the given width.

**Usage**

```r
caliperSize(scores, z, width, structure = NULL)
```

**Arguments**

- `scores` A numeric vector of scores providing 1-D position of units
- `z` Treatment indicator vector
- `width` Width of caliper, must be positive
- `structure` Grouping factor to use in computation

**Value**

numeric Total number of pairwise distances remaining after the caliper is placed.
**caliperUpperBound**

*Internal* Returns a reasonable upper bound on the arcs remaining after placing a caliper.

**Description**

(Internal) Returns a reasonable upper bound on the arcs remaining after placing a caliper.

**Usage**

```r
caliperUpperBound(scores, z, width, structure = NULL)
```

**Arguments**

- **scores** A numeric vector of scores providing 1-D position of units
- **z** Treatment indicator vector
- **width** Width of caliper, must be positive.
- **structure** Optional factor variable that groups the scores, as would be used by `exactMatch`. Including structure allows for wider calipers.

**Value**

numeric Total number of pairwise distances remaining after the caliper is placed.

---

**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**

```r
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.

---

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
**dist_digest**

(Internal) Remove the call before digesting a distance so things like omitting caliper and calling caliper=NULL give the same digest

**Description**

(Internal) Remove the call before digesting a distance so things like omitting caliper and calling caliper=NULL give the same digest

**Usage**

```r
dist_digest(dist)
```

**Arguments**

- `dist`: Distance object to hash. Must be one of `InfinitySparseMatrix`, `BlockedInfinitySparseMatrix`, `DenseMatrix`, `matrix`, or `distmatch.dlist`.

**Value**

Hash on the distance object with a null call

**Author(s)**

Josh Errickson

---

**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**

```r
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.
**exactMatch-methods**

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

```r
## 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.
- `treatment`: A vector the same length as `x` that can be coerced to a two level factor (e.g. a vector of 1s and 0s or a logical vector).
- `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 ‘NA’s
- `...`: Additional arguments for methods.

### Details

`exactMatch` creates a block diagonal matrix of 0s and `Inf`s. 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

The equivalent number of pairs in this match.

### See Also

`summary.optmatch`, `stratumStructure`
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)

```r
# 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 factors.

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 “Column-Name.NA” with indicators for each observation that is missing a value for “ColumnName”.

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


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.glm <- glm(fill.NAs(pr ~ t1 * t2, data=np.missing),
family=binomial))

# the previous call is equivalent to:
# glm(pr ~ t1 + t2 + 't1:t2' + t1.NA + t2.NA, fill.NAs(np.missing), family =
# binomial)

### 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))

### fill NAs without using treatment contrasts by making a list of contrasts for
### each factor ## following hints from http://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)],
  function(x){contrasts(x, contrasts = FALSE, drop=FALSE)},
)

### make a data frame filling the missing covariate values, but without
fmla2treatedblocking  (*Internal*) A helper function to turn formulas into treatment and blocking variables

**Description**

Given a function and any of the arguments normally passed to model.frame, this function will return a data.frame with two columns: a treatment indicator and a blocking factor.

**Usage**

```r
fmla2treatedblocking(x, ...)  
```

**Arguments**

- `x` A formula
- `...` Arguments to be passed to model.frame (e.g. data)

**Value**

data.frame containing two columns: `z` is a treatment indicator, `b` is a blocking factor

---

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**

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

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

... Additional arguments, including `within`, which may be passed to `match_on`.

**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").

If the program detects a large problem as been requested that may exceed the computational power of the user's computer, a warning is issued. If you wish to disable this warning, set `options("optmatch_warn_on_big_problem", value = FALSE)`.

**Value**

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

**References**


**Examples**

data(nuclearplants)

```r
### 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))
(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)
```

---

getMaxProblemSize  

*(Internal) What is the maximum allowed problem size?*
makeInfinitySparseMatrix

(Internal) Creating sparse matching problems

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

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

makeInfinitySparseMatrix

(makeInfinitySparseMatrix(data, cols, rows, colnames = NULL, rownames = NULL, dimension = NULL, call = NULL)

asInfinitySparseMatrix(x)

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

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

Arguments

- **data**: A vector of distances for the finite (allowed) treatment-control pairs.
- **cols**: A vector indicating the column number for each entry in data.
- **rows**: A vector indicating the row number for each entry in data.
- **colnames**: A optional character vector with the columns names of the matrix.
- **rownames**: A optional character vector with the row names of the matrix.
- **dimension**: An optional vector giving the dimensions of the matrix, which can be useful for indicating matrices with entirely Inf rows or columns. If supplied with row and columns names, it must match.
- **x**: A matrix to be converted to an InfinitySparseMatrix.
- **call**: Optional call object to store with the distance specification. Allows calling `update` on the distance object at later points.
- **value**: A possible value for `dimnames(x)`: see the ‘Value’ section.

Details

Usually, users will create distance specification using `match_on`, `caliper`, or `exactMatch`, but if you need to generate sparse matching problems directly, use this function. If the data are already in a matrix form, use `as.InfinitySparseMatrix`. If you have the finite entries in a vector format, use `makeInfinitySparseMatrix`.

Value

An object of class `InfinitySparseMatrix`, which will work as a distance argument to `fullmatch` or `pairmatch`.

Author(s)

Mark M. Fredrickson

See Also

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

Examples

```r
example.matrix <- matrix(c(1, 2, Inf, 3, Inf, 4, Inf, Inf, Inf), byrow = TRUE, nrow = 3,
                         dimnames = list(letters[1:3], LETTERS[24:26]))

optmatch::as.InfinitySparseMatrix(example.matrix)

# create the same sparse matrix directly, function will create the appropriate dims
# the data are in a different order, but the indices are correct
(example.ism <-
  optmatch::makeInfinitySparseMatrix(c(1, 2, 3, 4),
    c(1, 2, 1, 3),
    c(1, 1, 2, 2),
    LETTERS[24:26],
)```
makeOptmatch

(Internal) Create optmatch objects, the result of matching.

Description

This internal function is used to create the final output of the matching functions (fullmatch and pairmatch). The optmatch object descends from a factor, but contains additional information relating to the quality of the match.

Usage

makeOptmatch(distance, solutions, call, data = NULL)

Arguments

distance A DistanceSpecificaton object used to create the match.
solutions A list of the results of the matching, one list(cells,maxErr) object per sub-problem.
call The call to fullmatch or pairmatch to be displayed later.
data An object from which names or row.names will provide the order of the items in the match. If no names are attached to this object, the contents will be used as names.

Value

optmatch object

See Also

summary.optmatch
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

```r
matched(x)
unmatched(x)
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. `TRUE`s 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

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

mxpl.fm0 <- fullmatch(plantdist) # A feasible matching problem
c(sum(matched(mxpl.fm0)), sum(unmatched(mxpl.fm0)))
sum(matchfailed(mxpl.fm0))

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

mxpl.si <- factor(c('a', 'a', 'c', rep('d',4), 'b', 'c', 'c', rep('d', 16)))
names(mxpl.si) <- LETTERS[1:26]
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) & !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.

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, `InfinitySparseMatrix`es, so that they can be manipulated and modified in much the same way as dense matrices.

**Usage**

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

## S3 method for class 'formula'
match_on(x, within = NULL,
          caliper = NULL, data = NULL, subset = NULL,
          method = "mahalanobis", ...)

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

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

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

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

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

Arguments

- x: An object defining how to create the distances. All methods require some form of names (e.g. names for vectors or rownames for matrix like objects).
- 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.
- 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.
- data: An optional data frame.
- ...: Other arguments for methods.
- z: A factor, logical, or binary vector indicating treatment (the higher level) and control (the lower level) for each unit in the study.
- 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", "rank_mahalanobis", or pass a user created distance function.
- standardization: scale

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.

The match_on function is similar to the older, but still supplied, mdist function. Future development will concentrate on match_on, but mdist is still supplied for users familiar with the interface. For the most part, the two functions can be used interchangeably by users.

The function method takes as its x argument a function of three 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:

```r
f <- function(index, data, z) { abs(apply(index, 1, function(pair) { data[pair[[1]]] - data[pair[[2]]] }))
```

(Note: This simple case is precisely handled by the numeric method.)

The formula method produces, by default, a Mahalanobis distance specification based on the formula $z \sim x_1 + x_2 + \ldots$, where $z$ is the treatment indicator. 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 $X$s. (The pooling is of the covariance of $X$ within the subset defined by $z=\theta$ and within the complement of that subset. This is similar to a Euclidean distance calculated after reexpressing the $X$s 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". Or, implement your own; for hints as to how, refer to https://github.com/markmfredrickson/optmatch/wiki/How-to-write-your-own-compute-method

The glm method assumes its first argument to be a fitted propensity 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. (Outlier resistance is obtained by the application of mad, as opposed to sd, to linear propensity scores in the treatment; 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.) 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.

The bigglm 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.

The numeric method 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. The matrix and \texttt{InfinitySparseMatrix} 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 \texttt{fullmatch} or \texttt{pairmatch}.

**References**


**See Also**

\texttt{fullmatch}, \texttt{pairmatch}, \texttt{exactMatch}, \texttt{caliper}

**Examples**

```r
data(nuclearplants)
match_on.example <- list()
### Propensity score distances.
### Recommended approach:
(aGlm <- glm(pr~(pr+cost), family=binomial(), data=nuclearplants))
match_on.example$ps1 <- match_on(aGlm)
### A second approach: first extract propensity scores, then separately
### create a distance from them. (Useful when importing propensity
### scores from an external program.)
plantsPS <- predict(aGlm)
match_on.example$ps2 <- match_on(pr=plantsPS, data=nuclearplants)
### Full matching on the propensity score.
fm1 <- fullmatch(match_on.example$ps1, data = nuclearplants)
fm2 <- fullmatch(match_on.example$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.example$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)
```
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.
A generic function, with several supplied methods, for creating matrices of distances between observations to be used in the match process. Using these matrices, pairmatch() or fullmatch() can determine optimal matches.

Usage

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

Arguments

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.

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

(Deprecated, in favor of match_on) Create matching distances

Description

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


**See Also**

`fullmatch`, `pairmatch`, `match_on`

**Examples**

data(nuclearplants)
mdist.examples <- list()
### Propensity score distances.
### Recommended approach:
(aGlm <- glm(pr~-(pr+cost), family=binomial(), data=nuclearplants))
mdist.examples$ps1 <- mdist(aGlm)
### A second approach: first extract propensity scores, then separately
### create a distance from them. (Useful when importing propensity
### scores from an external program.)
plantsPS <- predict(aGlm)
mdist.examples$ps2 <- mdist(pr~plantsPS, data=nuclearplants)^{(1/2)}
### Full matching on the propensity score.
fullmatch(mdist.examples$ps1)
fullmatch(mdist.examples$ps2)
### Because `mdist.glm` uses robust estimates of spread,
### the results differ in detail -- but they are close enough
### to yield similar optimal matches.
all(fullmatch(mdist.examples$ps1)==fullmatch(mdist.examples$ps2)) # The same

### Mahalanobis distance:
mdist.examples$mhl <- mdist(pr - t1 + t2, data = nuclearplants)

### Absolute differences on a scalar:
sdiffs <- function(treatments, controls) {

minControlsCap

Set thinning and thickening caps for full matching

Description

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

Usage

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

Arguments

distance Either a matrix of non-negative, numeric discrepancies, or a list of such matrices. (See fullmatch for details.)

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

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

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

- `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


See Also

- `fullmatch`

Examples

```r
### TEMPORARY COMMENT OUT BY MMF
### #---- Should be DIRECTLY executable !! ----
### #-- => Define data, use random,
### #-- or do help(data=index) for the standard data sets.
### plandist <- matrix(nrow=7, ncol=19, byrow=TRUE, data=c(
```
### minExactMatch

#### 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

```r
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 $Z \sim X1 + X2$, where $Z$ 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 `nuclearplants` data frame has 32 rows and 11 columns. The data relate to the construction of 32 light water reactor (LWR) plants constructed in the U.S.A in the late 1960’s and early 1970’s. The data was collected with the aim of predicting the cost of construction of further LWR plants. 6 of the power plants had partial turnkey guarantees and it is possible that, for these plants, some manufacturers’ subsidies may be hidden in the quoted capital costs.

Usage

`nuclearplants`

Format

This data frame contains the following columns:

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

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

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
```r
## 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**

```r
## 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 this object is passed and the Rlt tools package is loaded, 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 Rlt tools package for more details.
- `...` Additional arguments to pass to xBalance when also passing a propensity model.
- `min.controls` To minimize 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`

---

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.

All functionality of the `pscore.dist` function has been moved into the `mdist` function. Additionally, this function will also act on other objects, such as formulas. The `match_on` function also provides similar functionality, though with a different syntax.

All functionality of the `mahal.dist` function has been moved into the `mdist` function. Additionally, this function will also act on other objects, such as `glm` objects. The `match_on` function also provides similar functionality, though with a different syntax.

**Usage**

```r
pscore.dist(...)  
mahal.dist(...)```

**Arguments**

...  
All arguments ignored.

**See Also**

`mdist.match_on`
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.

Author(s)

Josh Errickson

---

optmatch_same_distance

Description

Checks if the distance newdist is identical to the distance used to generate the optmatch object obj.

Usage

optmatch_same_distance(obj, newdist)
Arguments

obj An optmatch object.
newdist A distance

Details

Note that the distance is hashed with its call set to NULL. (This avoids issues where, for example, \texttt{match_on(Z~X, data=d, caliper=NULL)} and \texttt{match_on(Z~x, data=d)} produce identical matches (since the default argument to \texttt{caliper} is NULL) but distinct calls.)

Value

Boolean whether the two distance specifications are identical.

Author(s)

Josh Errickson

\begin{description}
\item[pairmatch] \emph{Optimal 1:1 and 1:k matching}
\end{description}

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

\begin{verbatim}
pairmatch(x, controls = 1, data = NULL,
            remove.unmatchables = FALSE, ...)
\end{verbatim}

\begin{verbatim}
pair(x, controls = 1, data = NULL,
     remove.unmatchables = FALSE, ...)
\end{verbatim}

Arguments

\begin{itemize}
\item[x] Any valid input to \texttt{match_on}. If \texttt{x} is a numeric vector, there must also be passed a vector \texttt{z} indicating grouping. Both vectors must be named.

\item[controls] The number of controls to be matched to each treatment

\item[data] Optional data set.

\item[remove.unmatchables] Should treatment group members for which there are no eligible controls be removed prior to matching?

\item[...]
\end{itemize}

Additional arguments to pass to \texttt{match_on} or \texttt{fullmatch}. It is an error to pass \texttt{min.controls}, \texttt{max.controls}, \texttt{mean.controls} or \texttt{omit.fraction} as \texttt{pairmatch} must set these values.
pairmatch

Details

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

If \texttt{remove.unmatchables} is \texttt{FALSE}, then if there are unmatchable treated units then the matching as a whole will fail and no units will be matched. If \texttt{TRUE}, then this unit will be removed and the function will attempt to match each of the other treatment units. (In this case matching can still fail, 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 \texttt{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 \texttt{distance} (with possible additions from the optional \texttt{data} argument). Each element of the vector is the concatenation of: (i) a character abbreviation of \texttt{subclass.indices}, if that argument was given, or the string 'n' if it was not; (ii) the string '.'; and (iii) a non-negative integer. Unmatched units have \texttt{NA} entries. Secondarily, \texttt{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 \texttt{exceedances} attribute gives upper bounds, not necessarily sharp, for the amount by which the sum of distances between matched units in the result of \texttt{fullmatch} exceeds the least possible sum of distances between matched units in a feasible solution to the matching problem given to \texttt{fullmatch}. (Such a bound is also printed by \texttt{print.optmatch} and by \texttt{summary.optmatch}.)

Value

A \texttt{optmatch} object (factor) indicating matched groups.

References


See Also

\texttt{matched}, \texttt{caliper}, \texttt{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 \texttt{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)])

---

plantdist                  Dissimilarities of Some U.S. Nuclear Plants

### Description

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

### Usage

data(plantdist)

### Format

A 7 by 19 numeric matrix.

### Source


### References

print.optmatch  Printing optmatch objects.

Description

Printing optmatch objects.

Usage

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

Arguments

- `x`: The optmatch object, as returned by `fullmatch` or `pairmatch`.
- `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.
- `quote`: A boolean indicating if the matched group names should be quoted or not (default is not to quote).
- `...`: Arguments passed to `print.default`.

See Also

`fullmatch`, `pairmatch`, `print`, `summary.optmatch`.

Examples

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

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

relaxinfo  Display license information about embedded code

Description

Function to display license information regarding code embedded in `optmatch`.

Usage

`relaxinfo()`
Value

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

Author(s)

Ben B. Hansen

---

**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**

*Wrapper for predict to cleanly look for new data to predict on.*

---

**Description**

When called without a `newdata` argument, it will attempt to determine the correct new data to predict on; e.g. in a `lm` or `glm` model, will use the data in that model.

**Usage**

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

- `object`: a model object from which prediction is desired.
- `newdata`: optionally, specifies a data frame in which to look for variables to predict with. When omitted, attempts to intelligently use the correct data frame as opposed to `predict` using the data originally used to create `object`.
- `...`: additional arguments passed to `predict`.

Details

If `newdata` (either the explicit argument, or the implicit data generated from `object`) has NA values, imputation will be performed on the missing data via the `fill.NAs` function and `object` will be refit using the imputed data frame, before calling `predict`.

If `newdata` is specified and contains no missing data, this is identical to calling `predict`.

If the call to create `object` is involved, particularly if it includes optional arguments such as `subset` or `weights` whose values reference the data, this function may fail or otherwise have undesirable results if the `newdata` argument is not given. It is therefore strongly recommended to include the `newdata` argument in these sort of situations.

Value

See individual `predict` functions.

Author(s)

Josh Errickson

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)

---

`setFeasibilityConstants`

*(Internal)* Sets up the default values for maximum feasible problems

Description

*(Internal)* Sets up the default values for maximum feasible problems
setTryRecovery

Usage

\begin{verbatim}
setFeasibilityConstants()
\end{verbatim}

---

\textbf{setTryRecovery} \hspace{1cm} \textit{(Internal) Sets up option to try recovery in fullmatch.}

---

\textbf{Descripion}

\textit{(Internal) Sets up option to try recovery in fullmatch.}

\textbf{Usage}

\begin{verbatim}
setTryRecovery()
\end{verbatim}

---

\textbf{stratumStructure} \hspace{1cm} \textit{Return structure of matched sets}

---

\textbf{Description}

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

\textbf{Usage}

\begin{verbatim}
stratumStructure(stratum,trtgrp=NULL,min.controls=0,max.controls=Inf)
\end{verbatim}

\textbf{Arguments}

\begin{verbatim}
stratum \hspace{1cm} \text{Matched strata, as returned by fullmatch or pairmatch}
trtgrp \hspace{1cm} \text{Dummy variable for treatment group membership. (Not required if stratum is an optmatch object, as returned by fullmatch or pairmatch.)}
min.controls \hspace{1cm} \text{For display, the number of treatment group members per stratum will be truncated at the reciprocal of min.controls.}
max.controls \hspace{1cm} \text{For display, the number of control group members will be truncated at max.controls.}
\end{verbatim}

\textbf{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, \( \sigma \), 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, \( h_s = \left( \frac{n_{ts}^{-1} + n_{cs}^{-1}}{2} \right)^{-1} \) (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. Since for a matched pair \( s, h_s = 1, \sum_s h_s \) can be thought of as the number of matched pairs needed to attain comparable precision. (Alternately, the stratification might be taken into account when comparing treatment and control groups using fixed effects in an ordinary least-squares regression, as in Hansen (2004). This leads to the same result. A still different formulation, in which outcomes are not modeled as random variables but assignment to treatment or control is, again suggests the same weighting across strata, and a measure of precision featuring \( \sum_s h_s \) in a similar role; see Hansen and Bowers (2008).)

Author(s)

Ben Hansen

References


See Also

`matched, fullmatch`

Examples

data(plantdist)

plantsfm <- fullmatch(plantdist) # A full match with unrestricted
subdim

# 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

## 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 list of 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.
subproblemSuccess  *(Internal)* Report successful subproblems.

**Description**

`fullmatch` can break up a large matching problem into smaller subproblems (for example, using strata defined by `exactMatch`). This function lists the subproblems in a match and list whether at least one treated unit was matched in subproblem. Subproblems that have no matched treated units are said to have "failed."

**Usage**

subproblemSuccess(x)

**Arguments**

- x The result of `fullmatch` or `pairmatch`.

**Value**

A named logical vector indicating either success or failure for each subproblem.

**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**

update.optmatch(optmatch, ..., evaluate = TRUE)

**Arguments**

- optmatch Optmatch object to update.
- ... Additional arguments to the call, or arguments with changed values.
- evaluate If true evaluate the new call else return the call.

**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.
validDistanceSpecification

Value
An updated optmatch object.

Author(s)
Josh Errickson

validDistanceSpecification
(Internal) Validate that objects are valid distance specifications.

Description
The functions fullmatch and pairmatch create optimal matchings of treated and control units given a matrix (or similar representation) of distances between treated and control units. These distance specifications must implement certain generic functions. This function checks that all necessary methods exist and the object can be used to specify distances in a way that the matching functions can use.

Usage
validDistanceSpecification(distance,
   stopOnProblem = TRUE)

Arguments
distance The object to test.
stopOnProblem If TRUE (default) the function will raise an error for invalid objects. Otherwise, it returns a logical.

Value
logical
Index

*Topic **datasets**
  - nuclearplants, 34
  - plantdist, 41

*Topic **manip**
  - fill.NAs, 11
  - matched, 21

*Topic **misc**
  - stratumStructure, 45

*Topic **nonparametric**
  - caliper, 4
  - exactMatch-methods, 10
  - fullmatch, 14
  - matched.distances, 23
  - mdist, 29
  - pairmatch, 39

*Topic **optimize**
  - fullmatch, 14
  - minControlsCap, 31
  - pairmatch, 39
  - relaxinfo, 42

antiExactMatch, 3, 11
asInfinitySparseMatrix
  (makeInfinitySparseMatrix), 18

BlockedInfinitySparseMatrix-class
  (exactMatch-methods), 10

caliper, 3, 4, 11, 16, 19, 25–27, 40

caliper,InfinitySparseMatrix-method
  (caliper), 4

caliper,matrix-method(caliper), 4

caliper,optmatch.dlist-method
  (caliper), 4

caliper-methods (caliper), 4

caliperSize, 6

caliperUpperBound, 7

cbind, 8

compare_optmatch, 7

contrasts, 12

dimnames,InfinitySparseMatrix-method
  (makeInfinitySparseMatrix), 18

dimnames<-,InfinitySparseMatrix-method
  (makeInfinitySparseMatrix), 18

dist_digest, 9

distUnion, 8

effectiveSampleSize, 9

exactMatch, 3, 5, 7, 8, 15, 16, 19, 25, 27, 28, 33, 34, 36, 48

exactMatch (exactMatch-methods), 10

exactMatch.formula-method
  (exactMatch-methods), 10

exactMatch.vector-method
  (exactMatch-methods), 10

exactMatch-methods, 10

factor, 12

fill.NAs, 11, 44

fmla2treatedblocking, 14

full (fullmatch), 14

fullmatch, 3, 5, 6, 8, 9, 11, 14, 19, 20, 22, 24, 27, 28, 30–32, 36, 39, 40, 42, 45, 46, 48, 49

getMaxProblemSize, 17

InfinitySparseMatrix-class (match_on), 24

lm, 10, 12, 13

mahal.dist (optmatch-defunct), 37

makeInfinitySparseMatrix, 18

makeOptmatch, 20

match_on, 3–5, 8, 11, 13, 15, 16, 19, 24, 29, 30, 37, 39, 43

matched, 21, 36, 40, 46

matched.distances, 23

matchfailed, 36

matchfailed (matched), 21
maxCaliper, 28
maxControlsCap (minControlsCap), 31
mdist, 25, 29, 37
minControlsCap, 31
minExactMatch, 33
model.frame, 34

nuclearplants, 34
num_eligible_matches, 35

options, 18
optmatch, 16, 36, 40
optmatch-class (optmatch), 36
optmatch-defunct, 37
optmatch_restrictions, 38
optmatch_same_distance, 38

pair (pairmatch), 39
pairmatch, 3, 5, 6, 8, 9, 11, 19, 20, 24, 27, 28, 30, 36, 39, 42, 45, 48, 49
plantdist, 41
predict, 44
print, 42
print.default, 42
print.optmatch, 37, 42
pscore.dist (optmatch-defunct), 37

rbind, 8
relaxinfo, 42

scoreCaliper, 43
scores, 43
setFeasibilityConstants, 44
setTryRecovery, 45
stratumStructure, 10, 45
subdim, 47
subproblemSuccess, 48
summary.optmatch, 10, 20, 42
summary.optmatch (optmatch), 36

unmatched, 36
unmatched (matched), 21
update, 19
update.optmatch, 48

validDistanceSpecification, 49