Basic uses of the `optmatch` package

Yevgeniya Kleyman and Ben Hansen

30th October 2006

For case-control studies, prospective studies with a treatment and a control group, and studies of a few other types, a statistician may wish to match individuals from one predesignated group, the cases or the treatment group, to similar members of another group of control subjects in order to reduce treatment selection bias and estimate treatment effect. This is bipartite matching.

Matching algorithms fall into two categories: greedy and optimal. In each case, there is a need for a criterion of discrimination between better and worse matches. Here, it takes form of a "distance". In greedy matching, treated units are considered in sequence, with each paired to the closest (as measured by distance) control that has not already been matched to another treated unit. This approach, also called "nearest available" matching, is easily implemented, but can result in matches that are much poorer than they need be. "Optimal" matching, on the other hand, assigns controls to treated units so as to minimize the average of distances among matched units. (The algorithms needed for optimal matching are not simply described, but they are functionally similar to sequential pairing procedures that reconsider previously made matchings at each new step.) When the number of controls is large, greedy and optimal matching often lead to very similar or same sets of matches; however optimal matching generally results in smaller distances within each pair (Gu and Rosenbaum, 1993).

To see the difference between greedy and optimal matching, consider the following example. Below is a table indicating the quality of potential pairings among four objects.

	y	z
w	A	A-
x	A-	F

A greedy algorithm for pair matching creates pairs sequentially, at each step choosing the pairing that then seems best, without attending to its effect on future pairings. A greedy match might begin with w, in which case w would be paired to y, as y is w's nearest available counterpart. In and of itself this is a good pairing, as the given distance matrix rates it a "A"; but it is bad in that it leaves only z as a possible match to x. The average of its two grades is "C". A luckier implementation would begin with x, matching it to y and then w to z, for an average of A-. Optimal matching removes luck from the picture, immediately matching x to y and w `to` ***`z`***.

Full matching is an especially general form of optimal matching. In a full match, some matched sets may contain one treated subject (or case) alongside one of more controls, while other matched sets may contain multiple treated units (cases) alongside one control. This distinguishes full matching from both pair matching, matching with $k \geq 2$ controls, and matching with a variable number of controls, although full matchings sometimes coincide with matches produced by these simpler methods (Hansen and Klopfer, 2005). Among all methods of matching for two groups, full matching alone is demonstrably optimal as a method of producing similarity with matched sets; this was shown by Rosenbaum (1991), who introduced the method. It shares these advantages with full matching with restrictions (Hansen and Klopfer, 2005), a procedure that can also mimic pair matching and matching with a fixed or variable number of controls. The workhorse of "optmatch" is a function, `fullmatch()`, that performs full matching and full matching with restrictions.

The following are the functions contained in "optmatch" and their respective purposes:

`fullmatch` Optimal full matching

`matched` Identification of units placed into matched sets

`matchfailed` Identification of units not placed into matched sets

`maxControlsCap` Set thinning cap for full matching

`minControlsCap` Set thickening cap for full matching

`unmatched` Identification of units not placed into matched sets

Optimal Full Matching

To follow this example:

1. Install the `optmatch` package

2. Install the `boot` package
3. Use command `help(nuclear, package=boot)`
4. Use command `data(nuclear, package=boot)`
5. Use command `attach(nuclear)`
6. Use command `library(optmatch)`
7. Use command `help(fullmatch, package=optmatch)`

`The main function is of the following form:`

```
 fullmatch(distance, subclass.indices = NULL, min.controls = 0, max.controls
= Inf,omit.fraction = NULL, tol = 0.01)
```

The only mandatory argument is `distance`.

```
distance: A matrix of nonnegative discrepancies, each indicating the per-
     missibility and desirability of matching the unit corresponding to
     its row (a 'treatment') to the unit corresponding to its column (a
     'control'); or a list of such matrices.  Finite discrepancies in-
     dicate permissible matches, with smaller discrepancies indicating more
     desirable matches.  Matrix 'distance', or the matrix elements of 'dis-
     tance', must have row and column names.
```

Example. The following data relate to the 26 light water reactor (LWR) plants constructed in the U.S.A. in the late 1960's and early 1970's. The data were collected with the aim of predicting the cost of construction of further LWR plants. To illustrate the use of `distance,` consider the problem of matching new to refurbished nuclear plants, for the purpose of comparing their construction costs. In the nuclear plans example, seven existing-site plans are to be matched to 19 new-site plants, in order to allow an adjusted assessment of the cost of building on existing versus new sights. Variables available for use in the analysis are cost and date of issue of construction permit. For the illustration, we matched only upon cap and date, but an earnest analysis might match on other variables as well. Also, in this example, we will use only the first 26 data points in the dataset, for which the variable `pt` (a binary variable where '1' indicates those plants withpartial turnkey guarantees) is set to zero.

Existing site

	date	capacity
A	2.3	660
B	3.0	660
C	3.4	420
D	3.4	130
E	3.9	650
F	5.9	430
G	5.1	420

date is date of construction, in years after 1965; capacity is net capacity of the power plant, in MWe above 400.

New site

	date	capacity
H	3.6	290
I	2.3	660
J	3.0	660
K	2.9	110
L	3.2	420
M	3.4	60
N	3.3	390
O	3.6	160
P	3.8	390
Q	3.4	130
R	3.9	650
S	3.9	450
T	3.4	380
U	4.5	440
V	4.2	690
W	3.8	510
X	4.7	390
Y	5.4	140
Z	6.1	730

The discrepancy matrix should record plants' differences on the covariates, here date of construction and capacity, but the manner in which it combines these differences is at the discretion of the analyst. To illustrate, we calculate a distance matrix from the *ranks* of the date and the capacity variables (following section 10.3.4 of (Rosenbaum, 2002)). This transforms the covariates as follows:

Existing site		
	date	capacity
A	1.5	22.5
B	4.5	22.5
C	10.0	13.5
D	10.0	3.5
E	18.0	19.5
F	25.0	15
G	23.0	12

Here, date is *rank of* date of construction, in years after 1965, and capacity is *rank of* net capacity of the power plant, in MWe above 400.

New site		
	date	capacity
H	13.5	7.0
I	1.5	22.5
J	4.5	22.5
K	3.0	2.0
L	6.0	13.5
M	10.0	1.0
N	7.0	11.0
O	13.5	6.0
P	15.5	10.0
Q	10.0	3.5
R	18.0	19.5
S	18.0	17.0
T	10.0	8.0
U	21.0	16.0
V	20.0	25.0
W	15.5	18.0
X	22.0	9.0
Y	24.0	5.0
Z	26.0	26.0

Now we create a matrix whose entries are the total differences between the ranks of dates and capacities for each pair. For example, the difference for the pair AH is (13.5-1.5)+(22.5-7.0) = 27.5 (which we round to 28, for simplicity).

This matrix, which we name `plantdist`, is formed as follows:

```
> attach(nuclear.nopt)
> plantdist <- round(outer(rank(cap)[as.logical(pr)], rank(cap)[!as.logical(pr)],
+     FUN = function(X, Y) {
+         abs(X - Y)
+     }) + outer(rank(date)[as.logical(pr)], rank(date)[!as.logical(pr)],
+     FUN = function(X, Y) {
+         abs(X - Y)
+     }))
> dimnames(plantdist) <- list(LETTERS[1:7], LETTERS[8:26])
> plantdist

   H  I  J  K  L  M  N  O  P  Q  R  S  T  U  V  W  X  Y  Z
A 28  0  3 22 14 30 17 28 26 28 20 22 23 26 21 18 34 40 28
B 24  3  0 22 10 27 14 26 24 24 16 19 20 23 18 16 31 37 25
```

```
C 10 18 14 18  4 12  6 11  9 10 14 12  6 14 22 10 16 22 28
D  7 28 24  8 14  2 10  6 12  0 24 22  4 24 32 20 18 16 38
E 17 20 16 32 18 26 20 18 12 24  0  2 20  6  8  4 14 20 14
F 20 31 28 35 20 29 22 20 14 26 12  9 22  5 15 12  9 11 12
G 14 32 29 30 18 24 17 16 10 22 12 10 17  6 16 14  4  8 17
```

Having created the distance matrix, we are in a position to match new to refurbished plants. Enter

```
> plantsfm <- fullmatch(plantdist)
```

This results in the following match:

```
  A   B   C   D   E   F   G   H   I   J   K   L   M   N   O   P   Q   R   S   T
m.1 m.2 m.3 m.4 m.5 m.6 m.7 m.4 m.1 m.2 m.4 m.3 m.4 m.3 m.4 m.3 m.4 m.5 m.5 m.4
  U   V   W   X   Y   Z
m.6 m.5 m.5 m.7 m.7 m.6
```

where m.1, m.2, etc. are the indices for matches. Before adjustments are made, the fullmatch command matches A to I, B- to J, etc.

Matching within Calipers

The statistician may wish to forbid the matching of certain treatment-control pairs, perhaps those that are quite dissimilar. On the other hand, it may be too restrictive to insist that matched units have the same covariates: for instance, in the nuclear plants example only four exactly matched pairs are possible, namely AI, BJ, DQ, and ER. For continuous covariates, matching within calipers enforces similarity of matches (if not that they be exactly the same) (Hansen and Klopfer, 2005).

Continuing the nuclear plants illustration, we impose a caliper of 3 years in the date of construction. This forbids matches between plants whose date of construction differs by more than three years. Note that this may not minimize total distance, since total distance is a combination of date and capacity.

Coding a caliper when using optmatch and R:

To incorporate such a caliper, invoke `fullmatch()` after modifying the distance matrix as follows,

```
> (plantdist <- plantdist/outer(date[pr == 1], date[pr == 0], function(x,
+     y) {
+     abs(x - y) < 3
+ }))
```

```
   H   I  J   K  L  M  N  O  P  Q  R  S  T  U  V  W  X   Y   Z
A 28   0  3  22 14 30 17 28 26 28 20 22 23 26 21 18 34 Inf Inf
B 24   3  0  22 10 27 14 26 24 24 16 19 20 23 18 16 31  37 Inf
C 10  18 14  18  4 12  6 11  9 10 14 12  6 14 22 10 16  22  28
D  7  28 24   8 14  2 10  6 12  0 24 22  4 24 32 20 18  16  38
E 17  20 16  32 18 26 20 18 12 24  0  2 20  6  8  4 14  20  14
F 20 Inf 28 Inf 20 29 22 20 14 26 12  9 22  5 15 12  9  11  12
G 14  32 29  30 18 24 17 16 10 22 12 10 17  6 16 14  4   8  17
```

The infinite entries indicate unwanted matches.

Maintaining balance between treatment and control using min.controls and max.controls

The statistician may wish to place restrictions on the relative proportions with which treated and control subjects are combined into matched sets, perhaps to control the variability of an estimate based on the matching (See Hansen 2004, §3). The following are two arguments of the `fullmatch()` function that allow that adjustment.

`min.controls`: The minimum ratio of controls to treatments that is to be permitted within a matched set: should be nonnegative and finite. If min.controls is not a whole number, the reciprocal of a whole number, or zero, then it is rounded down to the nearest whole number or reciprocal of a whole number.

`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.

Treatment-control balance by matched set

```
> table(plantsfm, ifelse(pr, "treated", "control"))

plantsfm control treated
     m.1       1       1
     m.2       1       1
     m.3       3       1
     m.4       6       1
     m.5       4       1
     m.6       2       1
     m.7       2       1
```

In this unrestricted match, the number of controls per treatment subject varies widely. Such imbalance can be prevented as follows.

```
> plantsfm1 <- fullmatch(plantdist, min.controls = 2, max.controls = 3)
> table(plantsfm1, ifelse(pr, "treated", "control"))

plantsfm1 control treated
      m.1       2       1
      m.2       2       1
      m.3       3       1
      m.4       3       1
      m.5       3       1
      m.6       3       1
      m.7       3       1
```

Specifying `min.controls` and/or `max.controls` amounts to *full matching with restrictions*, which improves matched sets' treatment-control balance.

To determine the largest value of `min.controls` with which the matching problem is possible to solve, use `minControlsCap`. To get the smalles feasible value of `max.controls`, use `maxControlsCap`. (Note: these function calls `fullmatch` repeatedly, in a line search of the positive half-line. They can be time-consuming in large matching problems.)

```
> (mincc <- minControlsCap(plantdist))

$strictest.feasible.min.controls
m
2

$given.max.controls
  m
Inf

> maxControlsCap(plantdist, min.controls = mincc$strictest.feasible)

$given.min.controls
m
2

$strictest.feasible.max.controls
m
3
```

Largest treatment-control distances, by matched set

```
> tapply(names(plantsfm), plantsfm, FUN = function(x, dmat) {
+     max(dmat[match(x, dimnames(dmat)[[1]]), match(x, dimnames(dmat)[[2]])],
+         na.rm = TRUE)
+ }, dmat = plantdist)

m.1 m.2 m.3 m.4 m.5 m.6 m.7
  0   0   9   8   8  12   8
```

This function gives the largest difference in each of the matched sets.

A similar application of the `tapply()` function gives the actual ranges of the `date` variable within each match, i.e. the biggest difference on the variable `date` in each match:

```
> tapply(date, plantsfm, FUN = function(x) {
+     diff(range(x))
+ })

 m.1  m.2  m.3  m.4  m.5  m.6  m.7
0.00 0.00 0.58 0.66 0.42 1.58 0.75
```

For example, in m.6, we see that the biggest difference in the dates of F,U, and Z is the difference between U and Z: 71.08 - 69.50 = 1.58.

Omitting a fraction of controls

The statistician may wish to discard a portion of the control data. This may be done if a cost issue arises or if the statistician would like to discard outliers on the distance.

`omit.fraction:` Optionally, specify what fraction of controls or treated subjects are to be rejected. If '`omit.fraction`' is a positive fraction less than one, then '`fullmatch`' leaves up to that fraction of the control reservoir unmatched. If `omit.fraction` is a negative number greater than -1, then `fullmatch` leaves up to |`omit.fraction`| of the treated group unmatched. Positive values are only accepted if `max.controls` >= 1; negative values, only if `min.controls` <= 1. If `omit.fraction` is not specified, then only those treated and control subjects without permissible matches among the control and treated subjects, respectively, are omitted.

Suppose we wish to omit 50% of the control subjects:

```
> (plantsfm2 <- fullmatch(plantdist, omit.fraction = 0.5))

   A    B    C    D    E    F    G    H    I    J    K    L    M    N    O    P
 m.1  m.2  m.3  m.4  m.5  m.6  m.7 m.01  m.1  m.2 m.02  m.3  m.4 m.03 m.04 m.05
   Q    R    S    T    U    V    W    X    Y    Z
 m.4  m.5  m.5 m.06  m.6 m.07  m.5  m.7 m.08 m.09

> table(plantsfm2, ifelse(pr, "treated", "control"))

plantsfm2 control treated
     m.01       1       0
     m.02       1       0
     m.03       1       0
     m.04       1       0
     m.05       1       0
     m.06       1       0
     m.07       1       0
     m.08       1       0
     m.09       1       0
     m.1        1       1
     m.2        1       1
     m.3        1       1
     m.4        2       1
     m.5        3       1
     m.6        1       1
     m.7        1       1
```

If before we had a total of 19 controls, of them all were matched, now we have matched only 10 of them. The first nine rows of the above table are not matched sets, but rather indicators for unmatched controls. Controls H, K, N, O, P, T, V, Y, and Z are now not matched to a treated subject.

Rounding

If the matching is taking too long, increasing the `tol` argument may speed up the process.

`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` 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.

Matched, Unmatched, and Matchfailed

An easy way to check which plants were matched and unmatched is to use commands `matched()` and `unmatched()`. In our original matching attempt, full matching was performed so we see:

```
> matched(plantsfm)

 [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[16] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE

> table(unmatched(plantsfm))

FALSE
   26
```

After we omitted half of our controls, however, the output is

```
> table(matched(plantsfm2))

FALSE  TRUE
    9    17

> table(unmatched(plantsfm2))

FALSE  TRUE
   17     9
```

Names of the controls that were not matched can be retrieved as follows:

```
> names(plantsfm2)[unmatched(plantsfm2)]

[1] "H" "K" "N" "O" "P" "T" "V" "Y" "Z"
```

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.

```
> table(matchfailed(fullmatch(plantdist, min.controls = 5)))

TRUE
  26
```

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`.

In this case there were no inconsistencies in the constraints, so the command returns:

```
> table(matchfailed(plantsfm))

FALSE
   26

> detach()
```

References

Gu, X. and Rosenbaum, P. R. (1993), "Comparison of Multivariate Matching Methods: Structures, Distances, and Algorithms," *Journal of Computational and Graphical Statistics*, 2, 405–420.

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 Klopfer, S. O. (2005), "Optimal full matching and related designs via network flows," Tech. Rep. 416, Statistics Department, University of Michigan.

Rosenbaum, P. R. (1991), "A Characterization of Optimal Designs for Observational Studies," *Journal of the Royal Statistical Society*, 53, 597– 610.

— (2002), *Observational Studies*, Springer-Verlag, 2nd ed.