How to create lists of distance matrices for large matching problems

Ben Hansen

November 4, 2005

Optimal matching of treatment (case) to control subjects requires a treatment-by-control ($n_t \times n_c$) matrix of discrepancies, ratings of the desirability of each potential match. In propensity-score matching, for example, the discrepancy matrix gives magnitudes of differences on the propensity score for each possible pairing of a treatment to a control. Often the data arrive in a data frame containing a row for each observation and a column for each of k variables, in which case the discrepancies are usually calculated from information in this frame. As observations are added, a distance matrix derived from a data frame grows quite a bit more quickly than the size of the frame, since the distance matrix contains $n_t \times n_c$ entries whereas the data frame contains only $(n_t + n_c)k$ entries. As the number of observations moves into the thousands, an $n_t \times n_c$ numeric matrix quickly exceeds the storage capacity of most desktop computers, even as the $(n_t + n_c)k$ data frame is readily accommodated.

For these situations, it becomes necessary to split the matching problem, and with it the data frame, into parts. A data set with 2000 treatment subjects and 4000 controls would seem to call for a 2000×4000 distance matrix, containing 8 million cells; but by separately matching men to men and women to women, one might split it into two matching problems, each requiring just 2 million cells. The total storage requirements would be halved; furthermore, it is easier for R to store two matrices of 2 million cells each than it would be for it to store a single 4 million cell matrix. It is up to the data analyst to decide which variable or variables to split along, but once this decision is made, `fullmatch` has some features to ease separation of the constituent matching problems and the joining of their ultimate results.

For simplicity, assume that matching is to be done along a single variable,

such as a propensity score, and assume that the variable appears in a data frame that also contains an indicator of treatment/control group membership. For concreteness, let these variables be named `Pscore` and `TrtDummy`. (If the variable is an estimated propensity score or other variable synthesized from given data, then it may have to be added to the original data frame in order to achieve this configuration.) In this case the discrepancy matrix is simply a matrix of absolute differences between treatment and control propensity scores.

```
myDist <- function(names,data)
          {
          stopifnot(all(names %in% row.names(data)))
          trtnames <-
  names[as.logical(data[names,'TrtDummy'])]
          ctlnames <-
          names[!data[names,'TrtDummy']]
          ans <-
  outer(X=data[trtnames,'Pscore'],
                Y=data[ctlnames,'Pscore'],
                FUN=function(X,Y){abs(X-Y)}
                )
          dimnames(ans) <- list(trtnames,ctlnames)
          ans
          }
```

`myDist` forms a distance matrix for only those treatments and controls belonging to a specified subset of the data frame. (The `outer` command is quite useful in this regard; it tends to be much faster than, say, coding the same operations in a `for` loop.) It will be important for later uses that the matrix produced by `myDist()` record names of the treatment and control subjects as well as distances between them. Since `myDist` forms a distance matrix only for those names it is given in its first argument, it's easy to test it on a small subset of your data frame.

To split a matching problem along lines of, say, gender and race, proceed as follows.

```
distlist <- tapply(row.names(myDataFrame),
                   myDataFrame[,c('gender', 'race')],
                   myDist,
                   data=myDataFrame)
fm1 <- fullmatch(distlist)
fm2 <- fullmatch(distlist,
                 min.controls=
                           tapply(myDataFrame$TrtDummy,
                                  myDataFrame[,c('gender', 'race')],
                                  function(x){
                                    pmax(floor(sum(!x)/sum(x)),
                                         1/ceiling(sum(x)/sum(!x)))} )
                 )
```

The method being described can be adapted to more complicated discrepancies, such as Mahalanobis discrepancies, combinations of Mahalanobis and propensity calipers, or variations on the Mahalanobis distance such as those produced by Sekhon's Matching package; the changes required to effect such an adaptation will be made to `myDist()`.

The code above is illustrative. Below is a small working example, using the nuclear plants data set from Cox and Snell's *Applied Statistics* (1981, p.82). It requires the "boot" package, which seems to be part of the base installation. Excluding the six "partial turnkey" plants (`pt==1`), I stratify on whether the plant was build in the northeast (`ne`) or not, then match on the plant capacities (`cap`) within a caliper on date of construction (`date`). Besides the commands to do this, displayed below are the two distance matrices produced, one each for `ne` equal to 0 and 1, and the matched sets that result from full matching separately within the two strata.

```
> library(optmatch)

The optmatch package makes essential use of D. P. Bertsekas
 and P. Tseng's RELAX-IV algorithm and code, as well as
 Bertsekas' AUCTION algorithm and code:

 Bersekas, D. P. and  Tseng, P., "Relaxation Methods for
 Minimum Cost ..." Operations Research, vol. 26, 1988, 93-114;
 Bertsekas, D. P., "An Auction/Sequential Shortest Path
 Algorithm for the Minimum Cost Flow Problem", LIDS Report
 P-2146, MIT, Nov. 1992;
 <http://web.mit.edu/dimitrib/www/noc.htm>.

 Bertsekas and Tseng freely permit their software to be used for
 research purposes, but non-research uses, including the use of it
 to 'satisfy in any part commercial delivery requirements to
 government or industry,' require a special agreement with them.
 By extension, this requirement applies to most any use of R
 functions in the optmatch package.

 To request permissions not here relayed, contact Professor Bertsekas at
 Laboratory for Information and Decision Systems
 Massachusetts Institute of Technology
 Cambridge, MA 02139
 (617) 253-7267 <dimitrib@mit.edu>

> library(boot)
> data(nuclear)
> myDataFrame <- nuclear[nuclear$pt == 0, ]
> row.names(myDataFrame)[as.logical(myDataFrame$pr)] <- LETTERS[1:7]
> row.names(myDataFrame)[!(myDataFrame$pr)] <- LETTERS[8:26]
> myDist <- function(names, data) {
+     stopifnot(all(names %in% row.names(data)))
+     trtnames <- names[as.logical(data[names, "pr"])]
+     ctlnames <- names[!data[names, "pr"]]
+     ans <- outer(data[trtnames, "cap"], data[ctlnames, "cap"],
+         FUN = function(X, Y) {
+             abs(X - Y)
```

```
+         })
+     ans <- ans/outer(data[trtnames, "date"], data[ctlnames, "date"],
+         FUN = function(X, Y) {
+             (abs(X - Y) < 1.5)
+         })
+     dimnames(ans) <- list(trtnames, ctlnames)
+     ans
+ }
```

```
> (distlist <- tapply(row.names(myDataFrame), myDataFrame[, "ne"],
+     myDist, data = myDataFrame))
$"0"
    I   L   M   O   Q   R   S   T   V   W   X   Y   Z
A   0 243 608 505 535 Inf Inf 287 Inf 152 Inf Inf Inf
C 243   0 365 262 292 228  28  44 268  91  36 Inf Inf
D 535 292  73  30   0 520 320 248 560 383 256 Inf Inf
E Inf 228 593 490 520   0 200 272  40 137 264 Inf Inf
G Inf Inf Inf Inf Inf 229  29 Inf 269  92  35 283 309

$"1"
    H   J   K   N   P   U
B 378   0 551 273 275 Inf
F Inf Inf Inf Inf Inf  17
> fm1 <- fullmatch(distlist)
> split(names(fm1), fm1)
$"0.1"
[1] "A" "I"

$"0.2"
[1] "C" "L" "S" "T" "W"

$"0.3"
[1] "D" "M" "O" "Q"

$"0.4"
[1] "E" "R" "V"

$"0.5"
[1] "G" "X" "Y" "Z"

$"1.1"
[1] "B" "H" "J" "K" "N" "P"

$"1.2"
[1] "F" "U"
```