Multimix in R

Murray Jorgensen

29 January 2019

About this document

This is a brief explanation of the R version of the Multimix program that is current at the above date. In earlier versions of this program most variables were global. As a step towards making a Multimix package for R I have written a more functional version of the program.

As is traditional we will use the Byar-Green Prostate Cancer data to illustrate the program.

Data Frame

The data should be placed in a data frame in which all continuous variable columns are numeric vectors and all discrete variables are factors.

```
coltype = c("numeric", "numeric", "factor", "factor", "numeric", "numeric", "factor",
"numeric", "numeric", "numeric", "numeric", "factor")
cancer = read.table("CANCER11a.DAT", header=FALSE, colClasses=coltype)
Stage = scan(file="Stage.txt")
Stage = Stage - 2
Init_grp = as.factor(Stage)
names(cancer)  = c("Age", "Wt", "PF" , "HX", "SBP", "DBP",  "EKG", "HG", "SZ", "SG", "AP", "BM" )
levels(cancer[,3])  <- c("Active", "Bed49", "Bed51")
levels(cancer[,4])  <- c("No_hist", "Hist")
levels(cancer[,7])  <- c("Normal", "Benign", "Rythmic", "H_blocks", "H_strain", "Old_MCI", "New_MCI")
levels(cancer[,12]) <- c("No_BM", "BM")
```

The current version of the algorithm works out variances and covariances by appropriate differencing. This may create numerical difficulties if some continuous variables are poorly scaled. To avoid this possibility the data may be transformed as in the following example. (This transformation was not necessary for the Prostate Cancer example and was not done.)

```
# dframe = transform(cancer, Age=Age/100, Wt=Wt/100, SBP=SBP/10, DBP=DBP/10, HG=HG/100, SG=SG/10)
```

To follow some of the code it is useful to be able to refer to the columns by either their name or their number:

```
## Age  Wt  PF  HX SBP DBP EKG  HG  SZ  SG  AP  BM
##   1   2   3   4   5   6   7   8   9  10  11  12
```

Model

The model is a mixture of q components. By default the variables in each component are assumed to be independent. However we can specify associations between continuous variables, and between a single discrete variable and one or more continuous variables. Rather than give a definition here I will just content myself with one particular model for the Prostate Cancer data. We may specify this model to Multimix by

```
cdep = list(c(5,6))
lcdep = list(c(12, 2, 8), c(3,1))
```

This means that within each mixture component

1. the variables SBP (Systolic Blood Pressure) and DBP (Dystolic Blood Pressure) have a joint marginal bivariate normal distribution, independently of all other variables;
2. the factor BM (Bone metastases) with levels No_BM and BM and the variables Wt (Weight) and HG (Haemoglobin) have a joint marginal Olkin-Tate Location distribution, independently of all other variables;
3. the factor PF (Performance) with levels Active, Bed49, and Bed51 (Normal activity, in bed less than half the daylight hours, in bed at least half the daylight hours) and the variable Age have a joint marginal Olkin-Tate Location distribution, independently of all other variables;
4. the factors HX (History of Cardiovascular disease) with levels No_hist and Hist and EKG (Electrocardiogram code) with levels Normal, Benign, Rythmic, H_blocks, H_strain, Old_MCI, and New_MCI have discrete distributions independently of each other and all other variables;
5. the remaining variables (SZ, SG, AP) have marginal univariate normal distributions independently of each other and all other variables.

The function data_organise

This function creates a list, depending on the dataframe and the model, of all the objects that remain constant throughout the fitting process. This list acts as a substitute for the global objects of earlier versions of the program, being supplied as a parameter to most of the other functions. The definition begins

```
data_organise <- function(dframe, Init_grp, cdep=NULL, lcdep=NULL,
                minpstar = 1e-9, nIt = 60)
```

The arguments of the function are

- dframe the dataframe to which the mixture model is to be fitted, for example the frame cancer defined above.
- Init_grp (Initial grouping) for a mixture model with q components a vector having length n = dim(dframe)[1] and taking all values from 1 to q giving an initial assignment of observations to clusters.
- cdep specification of groups of variables to have a multivariate normal distribution. For example cdep = list(c(5,6)) means that the model gives SBP and DBP a bivariate normal distribution in each cluster.
- lcdep specification of groups of variables to have a multivariate normal distribution. For example lcdep = list(c(12, 2, 8), c(3,1)) means that the model gives (BM, Wt, HG) and (PF, Age) Olkin-Tate Location distributions in each cluster.
- minpstar is an experimental parameter that may not be needed. It may be discarded in future versions.
- nIt is the number of iterations of the algorithm to be carried out. After several models have been fitted to a particular dataset a suitable value for this parameter should become clear.

The function mmain

This function runs the Multimix program. The ordinary user should only need to run the functions data_organise() and mmain(). The sole argument to mmain is the list D created by data_organise.

The file test-function.r contains a script written to apply Multimix to the Prostate Cancer data as given in CANCER11a.DAT.

The function `first.Z.to.P`

The pattern of the iteration goes between the $n \times q$ matrix `Z` and the list `P` of parameter estimates and related statistics. The element z_{ij} of `Z` may be thought of as the probability that observation i belongs to cluster j. Before we start the iteration neither `Z` nor `P` exist, so we must from `D`. The outline of the function is

```
first.Z.to.P <- function(D) {
 with(D, {
  Z <- model.matrix(~ 0 + Init_grp)
  ...
  [Set up storage structures and calculate stuff.]
  ...
  P <- list(dstat = dstat,
            ldstat = ldstat,
            ostat = ostat,
            ostat2 = ostat2,
            ovar = ovar,
            pistat = pistat,
            cstat = cstat,
            cstat2 = cstat2,
            cvar = cvar,
            cpstat = cpstat,
            lcstat = lcstat,
            lcstat2 = lcstat2,
            lcpstat = lcpstat,
            MVMV = MVMV,
            LMV = LMV,
            W = W)
  return(P)
  })
}
```

The function `P.to.Z`

This function calculates log-densities for each combination of observation and cluster and thence the log-likelihood for the current parameters and the `Z` matrix. An outline for the function is

```
P.to.Z <- function(P, D) {
    with(c(P, D), {
    ...
    [Calculate fragments of log-likelihoods.]
    ...
    zll <- list(Z = Z, llik = llik)
    return(zll)
    })
}
```

The function `Z.to.P`

This function is similar to `first.Z.to.P` but begins from an existing `Z` matrix. It starts a little differently:

```
Z.to.P <- function(Z, D, P) {
      with( c(D, P), {
           ppi <- colSums(Z)/n
```

```
W <- Z%*%diag(1/{n*ppi}) # Z scaled to have columns sum to 1 for use as weights.
...
```

Note that `P` also needs to be supplied to this function in order that the elements of the list may be overwritten to create the new `P`.