Graphical models with the betaMix package when the number of predictors is large

Haim Bar

June 10, 2021

1 Introduction

This document demonstrates how one can use the *betaMix* package to fit a graphical model when the number of predictors is large. The betaMix method relies on the calculation of all pairwise correlations among P predictors, and the main difficulty in applying any method which requires the calculation of the correlation matrix when P is very large is that storing the $P \times P$ matrix requires a very large memory. For example, with $P \approx 10^5$ and 8-bytes to store a double precision number, we will need about 75GB which is much more than standard computers have.

To be able to handle such cases, we avoid the storage of a large correlation matrix. Instead, we do the following.

1. The input file is reformatted so that are n observation in each of P rows, and we use a fixed-width format. This allows us to use the `fseek` function in C, and read only two rows (predictors) at a time.
2. Create A Sqlite database with 3 tables:
   - metadata contains P, n, and a standard deviation threshold κ so that any predictor with sd$< \kappa$ is considered as a constant vector, and has zero correlation with all other predictors. The table also contains the input file name, the creation date, and a description.
   - predictors contains the predictor number and name.
   - correlations contains first node ID, second node ID, the correlation between the two predictors, and the statistic z_{ij} which is what is used by *betaMix*
3. Read the input file and calculate the pairwise correlations. This is done using Rcpp. In two nested loops we get all pairs of vectors xi, xj where $i < j$ and calculate cor(xi,xj). With the fixed-width format we can access the selected pair very quickly and never keep more than two vectors in memory.
4. Load the metadata, predictors, and correlations to Sqlite3.
5. In R , open the database, select a random set from the correlations table (e.g., 100,000 pairs).
6. Fit the betaMix model using the subset, and get the threshold τ for $z_{ij} = sin^2(\theta_{i,j})$ below which pairs are considered as correlated.
7. Finally, select from the database all rows with $z_{ij} < \tau$ as the edges in the graph.

If the raw input file is too large to handle at once, it can be done in chunks (step 3). If the database becomes too large, it can be distributed across multiple machines (step 4). The random sample can be obtained (step 5) from any machine, or from all of them. The betaMix algorithm will only have to fit the model to this relatively small subset (although large from a CLT point of view). Selecting the pairs given τ which is obtained from betaMix can also be done in a distributed fashion (step 7).

2 Example – the riboflavin data

The riboflavin data [1] contains gene expression values for 4,088 genes, and the logarithm of vitamin B2 production rate for $N = 71$ samples. We want to fit a graphical model, and especially focus on the nodes which are connected to the riboflavin levels. With $P = 4089$ we can do it without using a SQL database.

We start by loading the *betaMix* package and reading the input file.

```
library("betaMix")
dat <- read.csv("riboflavin1.csv", sep="\t")
```

The file contains 71 rows and 4089 columns. Whether the data is arranged as $N \times P$ or $P \times N$ we will create a formatted input file which has a fixed-width format, and in every block (line) there are all the samples for one predictor. In this case the response has already been log-transformed, as can be seen in the following histogram

```
hist(dat[,1],col="grey66",border="white", xlab="log vitamin B2 production rate", main="")
```

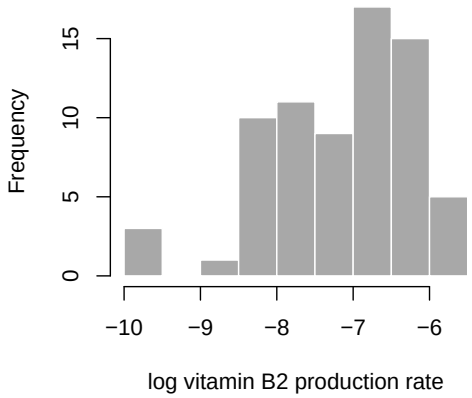

Figure 1: The distribution of the logarithm of vitamin B2 production rate

Notes

1. Any data transformation has to take place before continuing to the next steps.
2. There should be no missing data. Impute any missing values, or drop columns or samples with too many missing entries.

2.1 In-memory method

We will first do the analysis without using SQL. With $P = 4089$ we can store the correlation matrix in memory on typical personal computers. Note that we use a conservative threshold of $1/(P(P-1)/2)$.

```
#### In-memory method
resMem <- betaMix(dat,ind=TRUE, subsamplesize = 100000, delta =
↪  1/choose(ncol(dat),2),ppr=0.001, msg=FALSE)
```

```
plotFittedBetaMix(resMem,yLim=30)
```

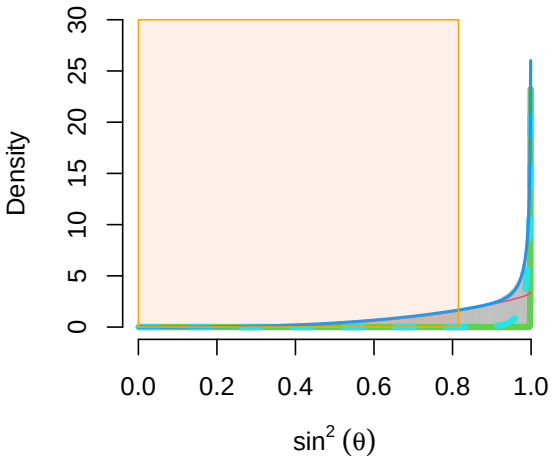

Figure 2: The distribution of z_{ij} and the fitted model.

Figure 2 shows that the mixture model fits the data very well.

We obtain a summary of the model fit and the adjacency matrix, and observe that the graph is not sparse at all. Over 2.8 million edges out of 8,357,916 are determined to be in the graphical model.

```
shortSummary(resMem)
adjMat1 <- getAdjMat(resMem)
```

```
ahat= 3.89055
bhat= 0.9748486
etahat= 35
Post. Pr. threshold= 0.8152351
No. nodes = 4,089
Max no. edges = 8,357,916
No. edges detected = 2,831,686
p0 = 0.222
```

The posterior probability threshold for z_{ij} was found to be = 0.815, and find that the riboflavin production rate variable is correlated with 105 genes. In our paper we show that they are highly-interconnected.

2.2 SQL-based method

To store the correlation information in a SQL database (possibly distributed across multiple machines), we perform the following pre-processing procedure. The correlation calculations are done in C, with the `calcCorr()` function in the *betaMix* package, and it never loads more than two rows (predictors) at a time to the memory. Importantly, in order to get a fast look-up of pairs we first convert the input to a fixed-width format, which allows us to jump from one predictor to another

very quickly with the `fseek` function in C. In the next code, we reformat the input data and save it in a file (the variable `fin` below) which will be the input to the `calcCorr()` function.

In the process, we will also create sqlite batch-load files. One file contains the data for the metadata table. This information includes the formatted file name (the `fin` variable below), database name, maximum predictor name length, record size, the number of predictors, the sample size, a zero-variance threshold, the creation date, and a description.

```
#### SQL-based method
outdir <- "tmp" # where the batch-load file and the SQL database will be saved
P <- ncol(dat)
n <- nrow(dat)
zeroVar <- 1e-5 # The threshold which will be used to decide if the variance of a vector is effectively 0.
dbname <- "riboflavin" # A database name
description <- "The riboflavin data, B{\"u}hlmann, Kalisch, and Meier 2014"
fin <- sprintf("%s/%s_fmt.txt",outdir,dbname) # the re-formatted data file which will be used by calcCorr.
if(file.exists(fin))
  file.remove(fin) # make sure we start a new file
```

In the predictors table we will store the predictor number and name. The `fin` file will have the predictor name, followed by N values, so we have to choose the maximum length of the predictor names (the variable `MaxNameLen`). Here, we use the maximum length plus one, but in other datasets names can be very long, so we may consider using a substring, if the truncated names remain unique.

```
MaxNameLen <- max(nchar(colnames(dat))) + 1
cat(MaxNameLen,"\n")
dat2 <- t(dat)
# Create a fixed-width file:
nchr <- rep(0,nrow(dat2))
for (i in 1:nrow(dat2)) {
  numval <- paste0(formatC(dat2[i,],width=16,digits=10), collapse = "")
  cat(paste0(sprintf("%-13s",rownames(dat2)[i]),numval,collapse = ""),"\n",
      file = fin, append = T, sep = "")
  nchr[i] <- nchar(paste0(sprintf("%-13s",rownames(dat2)[i]),numval))
}
cat(unique(nchr)+1,"\n")
# all lines must have the same length! If not, stop here and check.
if (length(unique(nchr)) == 1)
  recSize <- unique(nchr)+1

cat(fin,P,n,zeroVar,format(Sys.time(), "%Y%M%d%H%M%S"),
    description,sep="|", file=sprintf("%s/%s_meta.txt",outdir,dbname))
cat(paste(1:P,rownames(dat2), sep ="|"),sep="\n", file=sprintf("%s/%s_names.txt",
    outdir,dbname))
```

Note that we got a unique length for all the predictors (`recSize` = 1150) so the creation of the fixed-width file worked as desired. The metadata file looks like this:

```
tmp/riboflavin_fmt.txt|4089|71|1e-05|20211310081345|The riboflavin data, B{"
    u}hlmann, Kalisch, and Meier 2014
```

And the first four lines in the predictors file look like this:

```
1|q_RIBFLV
2|AADK_at
3|AAPA_at
4|ABFA_at
```

Now we can create the sqlite batch-load file for the correlations table. It has four columns: i,j, theta_ij, z_ij where i,j are the predictor numbers, theta_ij is the angle between them, and z_ij is the sine-squared of theta_ij, which is used by *betaMix*. Note that the `calcCorr()` function takes two file names (input and output, the maximum length of predictor names), and `recSize` is the total length of a block in the fixed-width format.

```
        fout <- sprintf("%s/%s_cors.txt",outdir,dbname)
        calcCorr(c(fin,fout), MaxNameLen, recSize, P,  n)
```

The first four lines in the correlations file look like this:

```
1|2|0.148300|0.978007
1|3|-0.111856|0.987488
1|4|0.034063|0.998840
1|5|0.300632|0.909620
```

To load the tables to sqlite, we create a batch file as follows:

```
        cat(sprintf("CREATE TABLE metadata ( inputfile TEXT NOT NULL,  p INTEGER, n INTEGER,
        ↪  zerovar DOUBLE, date TEXT, description TEXT);
.import  %s/%s_meta.txt metadata
CREATE TABLE predictors (num INTEGER, name TEXT NOT NULL);
.import  %s/%s_names.txt predictors
CREATE TABLE correlations (node1 INTEGER, node2 INTEGER,corr DOUBLE,zij DOUBLE);
.import %s/%s_cors.txt correlations\n",outdir,dbname,outdir,dbname,outdir,dbname),
↪  file=sprintf("%s/sqlbatch",outdir))
```

The batch file will contain the following six lines:

```
CREATE TABLE metadata ( inputfile TEXT NOT NULL,  p INTEGER, n INTEGER,
    zerovar DOUBLE, date TEXT, description TEXT);
.import  tmp/riboflavin_meta.txt metadata
CREATE TABLE predictors (num INTEGER, name TEXT NOT NULL);
.import  tmp/riboflavin_names.txt predictors
CREATE TABLE correlations (node1 INTEGER, node2 INTEGER,corr DOUBLE,zij
    DOUBLE);
.import tmp/riboflavin_cors.txt correlations
```

Start the sqlite session by invoking the following from a terminal:
`sqlite3 tmp/riboflavin`
Run the commands in the sqlbatch file. End the sqlite session by typing `.quit`
Except for the sqlite database (`tmp/riboflavin`) all the other files which were created by the script can now be deleted. The riboflavin sqlite database file is available at `https://uconn.sharepoint.com/:u:/r/sites/HaimBar/Shared%20Documents/betaMix/Databases/riboflavin?csf=1&web=1&e=iA6Bnn` (255 MB).

At this point we can do the analysis with *betaMix*. The procedure is nearly identical, except that we must give the database name, instead of a matrix:

```
        dbfile <- sprintf("%s/%s",outdir,dbname)
        resSQL <- betaMix(dbname=dbfile, ind=TRUE, subsamplesize = 100000, delta =
        ↪  1/choose(ncol(dat),2), ppr=0.001, msg=FALSE)
        adjMat2 <- getAdjMat(resSQL,dbname=dbfile)
```

The fitted model plot and the model summary are obtained exactly as in the in-memory example. For example, the model summary is as follows:

```
ahat= 4.129681
bhat= 1.024441
etahat= 35
Post. Pr. threshold= 0.815084
No. nodes = 4,089
Max no. edges = 8,357,916
No. edges detected = 2,829,579
p0 = 0.229
```

Getting the adjacency matrix is similar to the in-memory method, but we must provide the `dbname` argument, as in the example above.

In this case the riboflavin production rate variable is correlated with 105 predictors, which we obtain by using `length(which(adjMat2[1,] > 0))`.

When P is large the adjacency matrix may be too large to load to memory. We can obtain a sparse matrix by choosing which nodes we want to focus on. For example, we can obtain all the neighbors of the response variable, and get an adjacency matrix which contains only edges linking the set which consists of the response variable and its direct neighbors. In the following code we set `nodes = b2nbrs` which means that the returned adjacency matrix will only contain links among the nodes in the `b2nbrs` set. Figure 3 shows that the network around the q_EIBFLV node.

```
b2nbrs <- c(1, which(adjMat2[1,]>0))
adjMat3 <- getAdjMat(resSQL, dbname=dbfile, nodes = b2nbrs)
Msub <- adjMat3[b2nbrs,b2nbrs]
library(igraph)
pdf("tmp/B2nbrs.pdf",width = 6, height = 6)
plot(graph.adjacency(Msub, mode="undirected"),  vertex.label= rownames(Msub),
↪  vertex.label.cex=0.7, vertex.size=0.1, vertex.label.color='blue',
↪  edge.color='grey90',  asp=1)
dev.off()
```

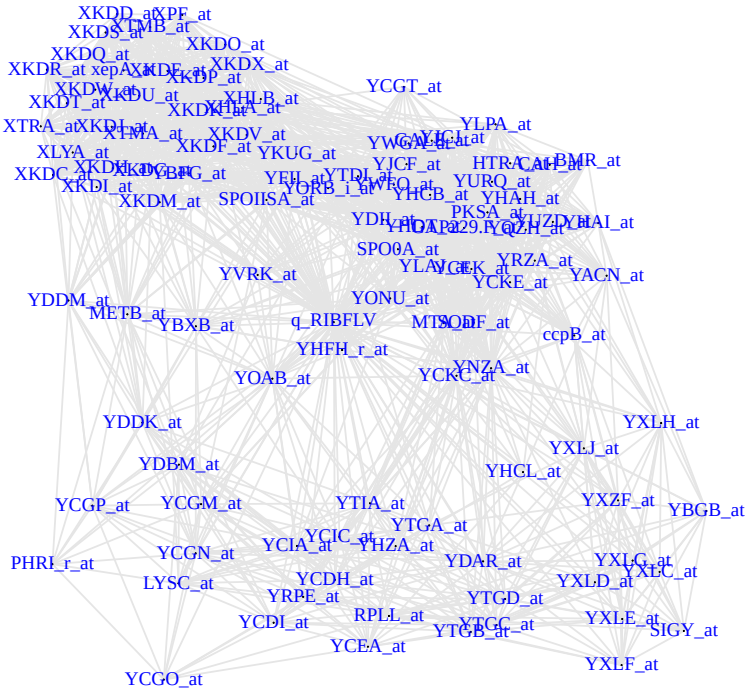

Figure 3: The network for the riboflavin variable.

3 Conclusion

Large-scale network analysis is computationally challenging, and methods which rely on calculating the correlation matrix and storing it in memory can only handle several thousands of nodes. The procedure we have described makes it possible to perform graphical model analysis on very large networks. The data, and especially the large correlation matrix, can be stored and queried across multiple machines thus making it possible to fit the betaMix model to handle essentially any number of nodes.

References

[1] P. Bühlmann, M. Kalisch, and L. Meier. High-dimensional statistics with a view toward applications in biology. *Annual Review of Statistics and Its Application*, 1(1):255–278, 2014.