Stat Learning HW3

Yaniv Bronshtein

11/17/2021

Import necessary libraries

```
library(gRbase)
library(gRain)
library(gRim)
library(ggm)
library(bnlearn)
```

```
##
## Attaching package: 'bnlearn'
```

```
## The following objects are masked from 'package:gRbase':
##
##     ancestors, children, parents
```

```
library(glasso)
```

Function to determine if DAG is acyclic

```
is.acyclic <- function(A){
  ## A is an adjacency matrix
  ## i.e. a_ij = 1 if the edge i->j *is* in the graph
  ##      a_ij = 0 if the edge i->j *is not* in the graph
  if (nrow(A) == 1)
    return(TRUE)
  ## compute transitive closure
  H <- A
  diag(H) <- 1
  repeat {
    HH <- sign(H %*% H)
    if (all(HH == H))
      break
    else H <- HH
  }
  diag(H) <- 0
  ## h_ij =1 if there is a directed path from i to j
  l <- H[lower.tri(H)]
  u <- t(H)[lower.tri(t(H))]
  com <- (l & u)
  all(!com)
}
```

Question 2

2. This problem uses a dataset gene.txt containing expression measurements of 9 genes. The names of the genes are included in the dataset. It also uses a R function is.acyclic(). Here is how you will use this function to tell whether a directed graph is acyclic or not. Represent a directed graph G as an adjacency matrix A: if there is an edge from node i to node j, then A[i, j] = 1, otherwise A[i, j] = 0. Note that the adjacency matrix A is not symmetric for a directed graph. The function is.acyclic() takes the adjacency matrix A as the input and returns whether it is acyclic or not. Both the dataset and the R function can be downloaded from the Homework folder. The R file also contains some examples on how to use the is.acyclic() function.

Extract the subset dataframe

```
df <- read.table('../data/gene.txt', header = TRUE)
df_subset <- df[,c(3,6,4,1)]
```

(a) Write your own code try all possible DAGs on the four genes GAL1, GAL2, GAL3, GAL7. There are 6 pairs of nodes, and between each pair of nodes, there can be three options: no edge, or an edge with either direction. Therefore, there are in total $36 = 729$ directed graphs over 4 nodes. For each of them, check whether the graph is acyclic using the is.acyclic() function. If it is indeed a DAG, estimated the model parameters, and report the BIC.

```
bases <- c('1','2','3')
all <- expand.grid(rep(list(bases), 6))

allAMat <- vector(mode='list',length = nrow(all))

BIC_scores <- NULL
dag <- empty.graph(names(df_subset))

genes <- c('GAL1','GAL2','GAL3','GAL7')
for (i in (1:nrow(all))){
  A <- matrix(0, 4, 4)

  A[1,2] <- +(all$Var1[i] == '2')
  A[2,1] <- +(all$Var1[i] == '3')
  A[1,3] <- +(all$Var2[i] == '2')
  A[3,1] <- +(all$Var2[i] == '3')
  A[1,4] <- +(all$Var3[i] == '2')
  A[4,1] <- +(all$Var3[i] == '3')
  A[2,3] <- +(all$Var4[i] == '2')
  A[3,2] <- +(all$Var4[i] == '3')
  A[2,4] <- +(all$Var5[i] == '2')
  A[4,2] <- +(all$Var5[i] == '3')
  A[3,4] <- +(all$Var6[i] == '2')
  A[4,3] <- +(all$Var6[i] == '3')
  rownames(A) <- genes
  colnames(A) <- genes
  allAMat[[i]] <- A
  if (is.acyclic(allAMat[[i]])){
     amat(dag) <- allAMat[[i]]
     BIC_val <- score(dag, df_subset)
     BIC_scores <- c(BIC_scores, BIC_val)
  }

}
```

(b) Report the models with three smallest BIC (there may be multiple models with the same BIC, report all of them). Plot the corresponding DAGs. [You can try to use the R code at the end of the slides to plot the DAGs. Or since there are only 4 nodes, you can draw by hand.]

```
bic_report <- data.frame(cbind(BIC_scores, Graph = seq(length(allAMat))))
```

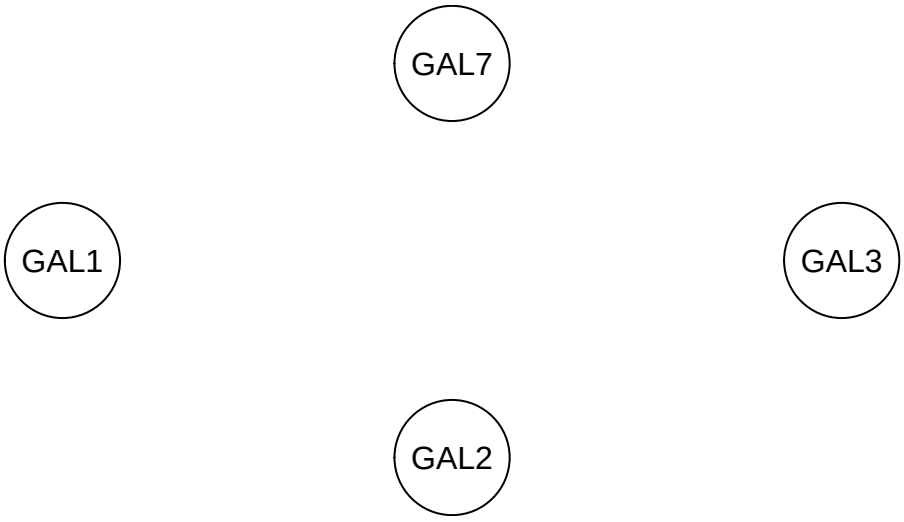
```
## Warning in cbind(BIC_scores, Graph = seq(length(allAMat))): number of rows of
## result is not a multiple of vector length (arg 1)
```

```
bic_report <- bic_report[order(bic_report[,1]),]
head(bic_report)
```

```
##     BIC_scores Graph
## 1    -970.1622     1
## 544  -970.1622   544
## 4    -961.3429     4
## 547  -961.3429   547
## 7    -961.3429     7
## 550  -961.3429   550
```

Note: There is a tie which is why 4 plots were displayed Plot1

```
amat(dag) <- allAMat[[bic_report[1,2]]]
plot(dag)
```

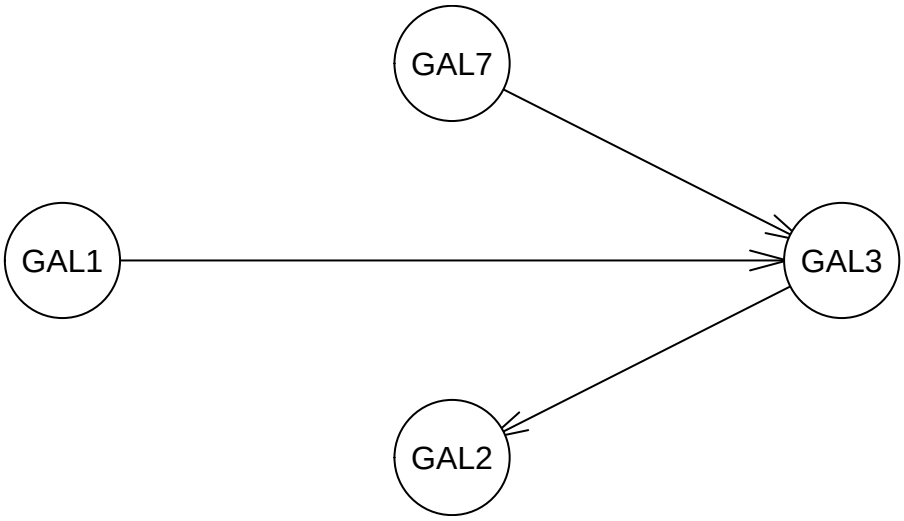



```
cat('BIC:', bic_report[1,1],'\n')
```

```
## BIC: -970.1622
```

Plot2

```
amat(dag) <- allAMat[[bic_report[2,2]]]
plot(dag)
```




```
cat('BIC:', bic_report[2,1],'\n')
```

```
## BIC: -970.1622
```

Plot3

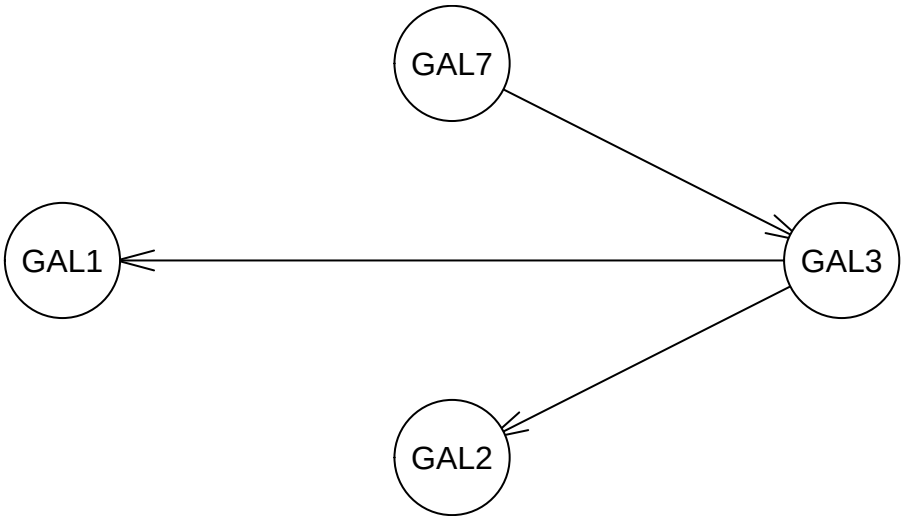
```
amat(dag) <- allAMat[[bic_report[3,2]]]
plot(dag)
```


```
cat('BIC:', bic_report[3,1],'\n')
```

```
## BIC: -961.3429
```

Plot4

```
amat(dag) <- allAMat[[bic_report[4,2]]]
plot(dag)
```




```
cat('BIC:', bic_report[4,1],'\n')
```

```
## BIC: -961.3429
```

Question 3

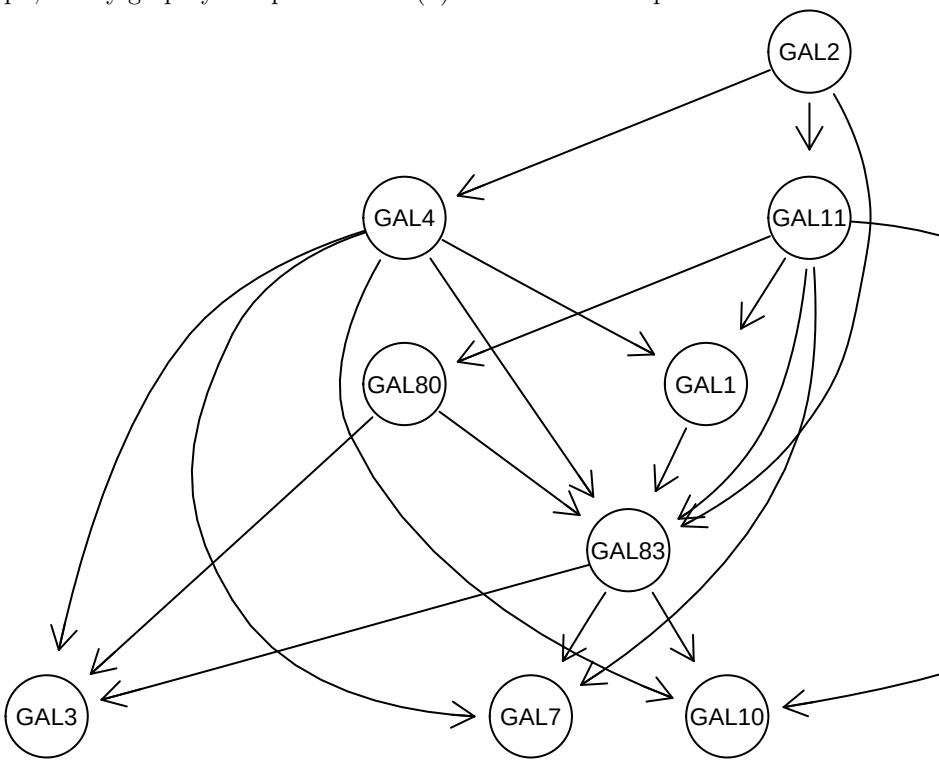
(a) Write you own code to implement the step-wise algorithm for estimating the graph structure. Write a generic code which takes a data matrix of any size as the input (so that it won't take any additional effort if you want to use it for the in-class workshop).

```
# Function to implement stepwise algorithm
stepwise_alg <- function(df) {
  cols <- colnames(df)
  n <- length(cols)
  A <- matrix(0, n, n)
  rownames(A) <- cols
  colnames(A) <- cols

  dag <- empty.graph(cols)
  amat(dag) <- A
  min_bic <- score(dag, df)

  smaller_BIC_exists = TRUE
  while(smaller_BIC_exists) {
    bic_Vals <- list()
    graphs <- list()

    edge_idx <- which(A==1, arr.ind=TRUE)

    len_edge_idx <- length(edge_idx)
    if (len_edge_idx > 0) {
      for (i in 1:(len_edge_idx/2)) {
        currA <- A
        currA[edge_idx[i,1], edge_idx[i,2]] <- 0

          dag <- empty.graph(cols)
          amat(dag) <- currA
```

```
        curr_bic <- score(dag, df)
      bic_Vals <- c(bic_Vals, curr_bic)
      graphs[[length(graphs)+1]] <- currA
    }
  }

  no_edge_idx <- which(A==0, arr.ind=TRUE)
  len_no_edge_idx <- length(no_edge_idx)
  if (len_no_edge_idx > 0) {
    for (i in 1:(len_no_edge_idx/2)) {
      if(no_edge_idx[i,1] != no_edge_idx[i,2]) {
        currA <- A
        currA[no_edge_idx[i,1], no_edge_idx[i,2]] <- 1

        if (is.acyclic(currA)) {
          dag <- empty.graph(names(df))
            amat(dag) <- currA
            curr_bic <- score(dag, df)
          bic_Vals <- c(bic_Vals, curr_bic)
          graphs[[length(graphs)+1]] <- currA
        }
      }
    }
  }

  if (len_edge_idx > 0) {

    for (i in 1:(len_edge_idx/2)) {
      currA <- A
      currA[edge_idx[i,1], edge_idx[i,2]] <- 0
      currA[edge_idx[i,2], edge_idx[i,1]] <- 1

      if (is.acyclic(currA)) {
        dag <- empty.graph(names(df))
          amat(dag) <- currA
          curr_bic <- score(dag, df)
        bic_Vals <- c(bic_Vals, curr_bic)
        graphs[[length(graphs)+1]] <- currA
      }
    }
  }
  unlisted_bic_Vals <- unlist(bic_Vals)
  this_min_bic <- min(unlisted_bic_Vals)
  this_min_graph <- which.min(unlisted_bic_Vals)

  if (this_min_bic < min_bic) {
    min_bic <- this_min_bic
    A <- graphs[[this_min_graph]]
  }
  else {
    smaller_BIC_exists <- FALSE
  }
}
```

```
  graph_and_bic <- list(A, min_bic)
  return(graph_and_bic)
}
```

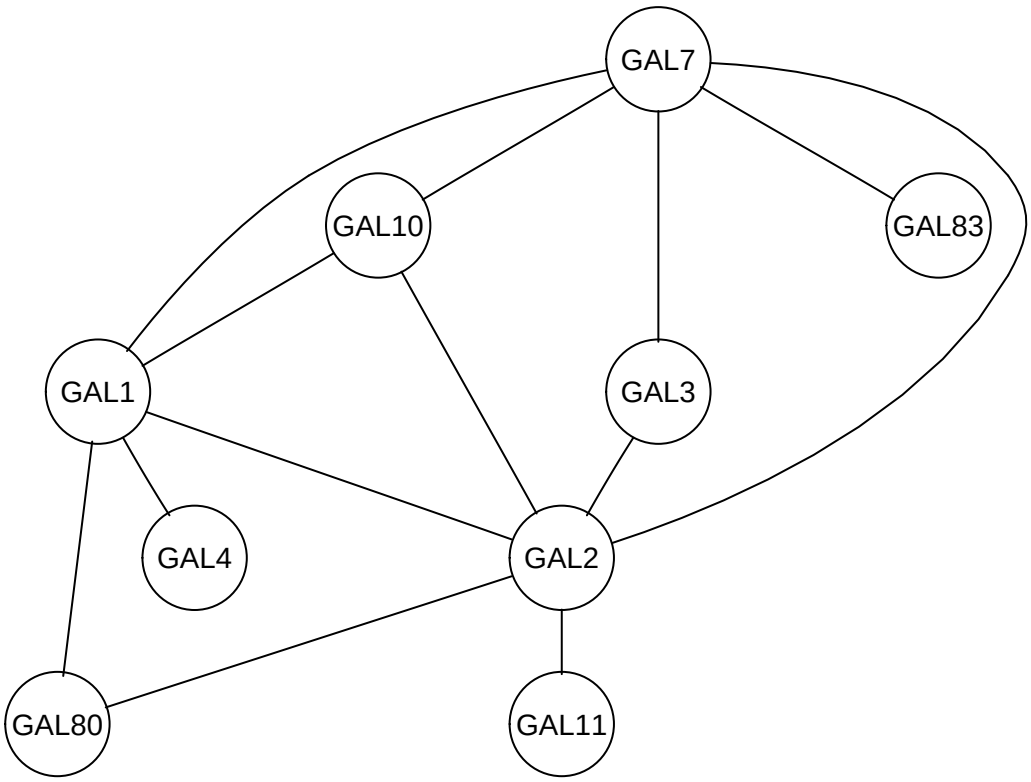
b) Try your program on the full gene.txt dataset. You will need an initial graph to start your algorithm, which can be the empty graph, full graph, or any graph you report in Part (b) of Problem 2. Report the DAG you found with the smallest BIC.



Question 5

(a) Use the graphical lasso to estimate the undirected graph for the gene.txt data. Try different choices of the tuning parameter lambda. Report and plot the undirected graphs you found.

```
lasso_model <- glasso(cov(df),rho=.05)

theta <- lasso_model$wi
cols <- names(df)
colnames(theta) <- cols
rownames(theta) <- cols
adj <- (theta != 0)
adj <- adj*1
diag(adj) <- 0
g1 <- as(adj, "graphNEL")

plot(g1)
```




```
cat('Rho:', .05,'\n')
```

```
## Rho: 0.05
```

Analysis: The greater the value of the constant Rho, the more disconnected the graph becomes