Appendix part 2

47602, 35261, 38844, 41781

12/12/2019

```r
# libraries used
library(MASS)
library(matrixcalc)
library(reshape2)
library(DescTools)
library(tidyverse)
library(glmnet)
library(glasso)
```

Node-wise lasso

```r
# Generate simulation matrix, with different p's and n's

simulation <- function(p, n){
  #a generate the lower triangle part of the pxp matrix with 10% to be
  #0.5 and 90% to be 0.
  a <- rbinom(n = p * (p - 1) / 2 , size = 1, prob = 0.1)
  a[a == 1] <- 0.5
  #B become the B in the sheet. Diagnal is all 0.
  B <- matrix(0, p, p)
  B[lower.tri(B, diag = FALSE)] <- a
  B[upper.tri(B)] <- t(B)[upper.tri(B)]
  #Identity matrix
  I <- diag(x = 1, p, p)
  #In this version, delta will start from 6 and choose as minimal
  #as possible, usually delta = 6 will be chosen.
  delta <- 6
  while (is.positive.definite(B+delta*I, tol=0)==FALSE){delta <- delta + 1}
  #theta
  theta = B + delta*I
  #standardize the theta
  standard_theta <- cov2cor(theta)
  #calculate the inverse of theta
  covMatrix <- solve(standard_theta)
  #generate n random samples from a multivariate gaussian distribution
  #with zero mean and the covariance matrix sigma = theta^-1.
  testdata <- mvrnorm(n = n, mu = numeric(p), Sigma = covMatrix, tol = 0, empirical = FALSE, EISPACK = 
  ls1 <-  list("data" = testdata, "standardtheta" = standard_theta, "theta" = theta)

  return(ls1)
}
```

```r
# Generate true edge
# Argument: theta.
# Return: a dataframe. Each element in the frame means if there is
#an edge between row index and column index (eg: element at (1,3)
#is FALSE, then X_1 and X_3 don't have edge inside.).

true_edge <- function(theta){
  numOfDims <- ncol(theta)
  theta <- data.frame(theta)
  tf <- data.frame(lapply(theta, function(x) {x!=0}))
  colnames(tf) <- seq(numOfDims)
  return(tf)
}
```

```r
# This is the code for choosing best lambda based using cross
#validation (select lambda corresponds to lowest MSE)
#
# Arguments:1. data: In this case we input the simulation data
#           2. numOfFolds: 5,10 or loocv. need to type in the
#           exact integer.
#           3. MIN_1SE : Input "MIN" or "1SE".
#
# Retutn: 1.final: lambda. a number.
cv_best_lambda <- function(data,numOfFolds, MIN_1SE){

  numOfRows <- nrow(data)
  numOfDims <- ncol(data)

  lambda.best_list = rep(0,numOfDims)
  for (i in seq(numOfDims)){
    y <- (data[,i])
    x <- (data[,-c(i)])
    lasso.train <-cv.glmnet(x, y,type.measure = "mse",nfolds = numOfFolds)
    if (MIN_1SE == 'MIN'){
      lambda.best_list[i] <-lasso.train$lambda.min
    } else if (MIN_1SE == '1SE'){
      lambda.best_list[i] <-lasso.train$lambda.1se
    }
  }
  final <- mean(lambda.best_list)
  return(final)
}
```

```r
# This is the code for choosing best lambda based on the simulation data
#
# Arguments:1. data: In this case we input the simulation data
#
# Return: 1.final: which is the best lambda,in each case of
# making X_i as response, there will be a lambda that minimised the test error.
#                   In this case, the best lambda 'final' is the
# mean of these lambda. Further discussion needed here.
rmse_best_lambda <- function(data){

  numOfRows <- nrow(data)
  numOfDims <- ncol(data)

  lambda.best_list = rep(0,numOfDims)
  for (i in seq(numOfDims)){
    y <- (data[,i])
    x <- (data[,-c(i)])
    train <-sample(seq(numOfRows), 0.7*numOfRows, replace=FALSE)
    lasso.train <-glmnet(x[train,], y[train])
    pred.test <-predict(lasso.train, x[-train,])
    rmse <-sqrt(apply((y[-train]-pred.test)^2,2,mean))
    lambda.best <-lasso.train$lambda[order(rmse)[1]]
    lambda.best_list[i] <- lambda.best
  }
  final <- mean(lambda.best_list)
  return(final)
}
```

```
## Compute estimated edge table

# Test part:(Please import function: Simulation and
# choose_best_lambda first.)
# This is the code for creating the Edge table based on the node-wise lasso
# With some adjustment it is expected to be applied into graphical lasso.
#
# Arguments:1. data: In this case we input the
#            simulation data
#            2. lambda_choice: this is supposed to be the
#            best data we choose
#
# Retutn: 1.tf: A dataframe, col and row are the number
#                of predictors(eg. row 2 means X_2)
#                each element in the frame means if there
#                is an edge between row index and column index
#                (eg: element at (1,3) is FALSE, then X_1 and X_3
#                don't have edge inside.)
#
#This function is supposed to used after the choose_best_lambda function.
#This function is supposed to used for 1.2.1 NodeWise lasso approach.
#However it only needs a few modification before we can
# apply it to the graphical lasso approach.

edge_table <- function(data_set, lambda_choice){

  numOfRows <- nrow(data_set)
  numOfDims <- ncol(data_set)

  edge <- data.frame(matrix(ncol = numOfDims, nrow = numOfDims))
  for (i in seq(numOfDims)){
    y <- (data_set[,i])
    x <- (data_set[,-c(i)])
    coeff <- coef(glmnet(x,y, lambda=lambda_choice))
    coeff <- as.matrix(coeff)[-c(1)]
    coeff <- append(coeff, 1, after= i-1)
    edge[i,] <- coeff
  }
  tf <- data.frame(lapply(edge, function(x) {x!=0}))
  colnames(tf) <- seq(numOfDims)
  return(tf)
}
```

```
# Generate confusion matrix/ROC curve point based on
# given edtimated edge table and true edge table.

# Arguments:1. estimate_edge: the TRUE/FALSE table from edgetable.R
#           2. trueEdge: the TRUE/FALSE table from trueEdge.R
#           3. estimate_way: input 'both' or 'either' here.
#           'both' stands for E_1 estimator. 'either' stands
#           for E_2 estimator.
#
# Retutn: 1.confusion: a list. Use
# list$TP/FP/FN/TN/TP_rate/FP_rate to get each
# confusion element or ROC curve point
#
confusion_matrix <- function(estimate_edge, trueEdge, estimate_way){
  numOfDims <- nrow(estimate_edge)
  confusion <- list(TP = 0,FP = 0,FN = 0,TN = 0,TP_rate = 0,FP_rate = 0)

  for (i in seq(numOfDims-1)){
    for (j in seq(numOfDims)[seq(numOfDims)>i]){
      if (estimate_way == 'either'){
        if (trueEdge[i,j] == TRUE){
          if (estimate_edge[i,j] == FALSE & estimate_edge[j,i] == FALSE){
            confusion$FN <- confusion$FN + 1
          } else {
            confusion$TP <- confusion$TP + 1
          }
        } else {
          if (estimate_edge[i,j] == FALSE & estimate_edge[j,i] == FALSE){
            confusion$TN <- confusion$TN + 1
          } else {
            confusion$FP <- confusion$FP + 1
          }
        }
      } else if (estimate_way == 'both') {
        if (trueEdge[i,j] == TRUE){
          if (estimate_edge[i,j] == TRUE & estimate_edge[j,i] == TRUE){
            confusion$TP <- confusion$TP + 1
          } else {
            confusion$FN <- confusion$FN + 1
          }
        } else {
          if (estimate_edge[i,j] == TRUE & estimate_edge[j,i] == TRUE){
            confusion$FP <- confusion$FP + 1
          } else {
            confusion$TN <- confusion$TN + 1
          }
        }
      }
    }
  }
  confusion$TP_rate <- confusion$TP/(confusion$TP + confusion$FN)
  confusion$FP_rate <- confusion$FP/(confusion$FP + confusion$TN)
```

```r
  return(confusion)
}
```

```r
# Plot ROC curve for E1 and E2
#
# This is the code for preparing the points that
# should be plotted on a ROC curve.
# Since we know for different lambda there will be
# different lasso coef->different estimated table->different confusion
# matrix->different FPR TPR points
# This function will first calculate the lambda range
# then use evenly spaced lambdas to generate points(TPR and FPR).
#
# Arguments:1. data_set: our simulated data set.
#           2. theta: the standard theta/general theta for
#           simulation. we use it to run true_edge/confusion_matrix functions.
#           3. estimate_way: input 'both' or 'either' here. '
#           both' stands for E_1 estimator. 'either' stands for E_2 estimator.
#           4. how_many_lambda_in_roc : default is 200
#           This instance how many points there will be shown on the ROC Curve.
#           Since the ROC curve was plotted by (FPR, TPR)
#           for different lambda,
#              so after we calculate the range of lambda,
#           we evenly space the range into 'how_many_lambda_in_roc' pieces.
#
# Retutn: 1.points:a (how_many_lambda_in_roc,3) sized
#             data frame. Storing three data: FPR,TPR,Lambda as columns.
#
# Usage: after generating ROC Points, we should
#               manually use ggplots to plot these point, sample code
#             are provided after the function.
ROC_curve <- function(data_set, theta, estimate_way, how_many_lambda_in_roc = 200){

  numOfDims <- ncol(data_set)
  points <- data.frame(matrix(ncol = 3, nrow = how_many_lambda_in_roc))

  max_lambda = 1
  min_lambda = 0
  for (i in seq(numOfDims)){
    y <- (data_set[,i])
    x <- (data_set[,-c(i)])
    lasso.train <-glmnet(x, y)
    if (lasso.train$lambda[1]>max_lambda){
      max_lambda <- lasso.train$lambda[1]
    }
    if (lasso.train$lambda[length(lasso.train$lambda)]<min_lambda){
      min_lambda <- lasso.train$lambda[length(lasso.train$lambda)]
    }
  }
  #Lambdas that I am going to use.
  lambda_seq <- seq(from = min_lambda, to = max_lambda, length.out= how_many_lambda_in_roc)
  #Put it into `points` which is a dataframe we just built
  points[,3] <- lambda_seq
  true_edge <- true_edge(theta)
  for (i in seq(length(lambda_seq))){
    print(c('Calculating points:',i))
    estimate_edge <- edge_table(data_set, lambda_seq[i])
    confusion <- confusion_matrix(estimate_edge, true_edge, estimate_way)
    TPR <- confusion$TP_rate
    FPR <- confusion$FP_rate
    # put them into `points` as well, they will be the coordinates.
    points[i,1] <- TPR
    points[i,2] <- FPR
  }
  colnames(points) <- c("TPR","FPR","Lambda")
  return(points)
}
```

```r
# Plot the ROC Curve for both methods under p=20,n=1000 and p=100,n=100.
# And calculate their AUC.
testsample <- simulation(20,1000)
testdata <- testsample$data
testtheta <-testsample$standardtheta
E1_roc <- ROC_curve(testdata, testtheta,"both",100)
E2_roc <- ROC_curve(testdata, testtheta,"either",100)
ggplot()+geom_step(data=E2_roc,mapping = aes(FPR, TPR,colour = 'E_2: either'))+
  geom_step(data=E1_roc, mapping = aes(FPR, TPR,colour = 'E_1: both'))
E1_auc <- AUC(E1_roc$FPR,E1_roc$TPR,method="step")
E2_auc <- AUC(E2_roc$FPR,E2_roc$TPR,method="step")

testsample <- simulation(100,100)
testdata <- testsample$data
testtheta <-testsample$standardtheta
E1_roc <- ROC_curve(testdata, testtheta,"both",100)
E2_roc <- ROC_curve(testdata, testtheta,"either",100)
ggplot()+geom_step(data=E2_roc,mapping = aes(FPR, TPR,colour = 'E_2: either'))+
  geom_step(data=E1_roc, mapping = aes(FPR, TPR,colour = 'E_1: both'))
E1_auc <- AUC(E1_roc$FPR,E1_roc$TPR,method="step")
E2_auc <- AUC(E2_roc$FPR,E2_roc$TPR,method="step")
```

```r
# Under 50 simulations running, under varying n's and p's,
# Calculate the mean and standard error of AUC for methods E1 and E2.
auc_table <- data.frame(matrix(ncol = 2, nrow = 50))
colnames(auc_table)<- c('E1_auc','E2_auc')
for (i in seq(50)){
  testsample <- simulation(20,100)
  testdata <- testsample$data
  testtheta <-testsample$standardtheta
  E1_roc <- ROC_curve(testdata, testtheta,"both",100)
  E2_roc <- ROC_curve(testdata, testtheta,"either",100)
  E1_auc <- AUC(E1_roc$FPR,E1_roc$TPR,method="step")
  E2_auc <- AUC(E2_roc$FPR,E2_roc$TPR,method="step")
  auc_table[i,1] <- E1_auc
  auc_table[i,2] <- E2_auc
}
E1E2_auc_mean <- sapply(auc_table, mean, na.rm = TRUE)
E1E2_auc_mean <- sapply(auc_table, sd, na.rm = TRUE)
```

```r
# Use below codes to generate 50 simulations run, with varying n's and p's
# in each generated file, it contains the performance of 3 different lambdas
# In example below, n is set at 10,000 and p at 20, and the file is saved
# under "20p10000n_data.csv")

# the same code is run for different combinations of n's and p's for E1 and E2.
n = 10000
p = 20
test_df <- data.frame(matrix(ncol=27, nrow=50))
colnames(test_df) <- c("lambda_rmse", "RMSE_either_FP", "RMSE_either_FN", "RMSE_either_TPR", "RMSE_eith
                       "RMSE_both_FP", "RMSE_both_FN", "RMSE_both_TPR", "RMSE_both_TFR",
                       "lambda_cv_min", "cv_min_either_FP", "cv_min_either_FN", "cv_min_either_TPR", "c
                       "cv_min_both_FP", "cv_min_both_FN", "cv_min_both_TPR", "cv_min_both_TFR",
                       "lambda_cv1se", "cv1se_either_FP", "cv1se_either_FN", "cv1se_either_TPR", "cv1se
                       "cv1se_both_FP", "cv1se_both_FN", "cv1se_both_TPR", "cv1se_both_TFR")

for (i in seq(50)){
  testsample <- simulation(p,n)
  testdata <- testsample$data
  testtheta <-testsample$standardtheta
  trueedge <- true_edge(testtheta)

  lambda_rmse <- rmse_best_lambda(testdata)
  estimate_edge_rmse <- edge_table(testdata, lambda_rmse)
  table_rmse_either <- confusion_matrix(estimate_edge_rmse, trueedge, "either")
  table_rmse_both <- confusion_matrix(estimate_edge_rmse, trueedge, "both")

  lambda_cv_MIN <- cv_best_lambda(testdata,10,"MIN")
  estimate_edge_cv <- edge_table(testdata, lambda_cv_MIN)
  table_cv_either <- confusion_matrix(estimate_edge_cv, trueedge, "either")
  table_cv_both <- confusion_matrix(estimate_edge_cv, trueedge, "both")

  lambda_CV_1SE <- cv_best_lambda(testdata,10,"1SE")
  estimate_edge_cv1se <- edge_table(testdata, lambda_CV_1SE)
  table_cv1se_either <- confusion_matrix(estimate_edge_cv1se, trueedge, "either")
  table_cv1se_both <- confusion_matrix(estimate_edge_cv1se, trueedge, "both")

  test_df[i,1] <- lambda_rmse
  test_df[i,2] <- table_rmse_either[2]
  test_df[i,3] <- table_rmse_either[3]
  test_df[i,4] <- table_rmse_either[5]
  test_df[i,5] <- table_rmse_either[6]
  test_df[i,6] <- table_rmse_both[2]
  test_df[i,7] <- table_rmse_both[3]
  test_df[i,8] <- table_rmse_both[5]
  test_df[i,9] <- table_rmse_both[6]
  test_df[i,10] <- lambda_cv_MIN
  test_df[i,11] <- table_cv_either[2]
  test_df[i,12] <- table_cv_either[3]
  test_df[i,13] <- table_cv_either[5]
  test_df[i,14] <- table_cv_either[6]
  test_df[i,15] <- table_cv_both[2]
  test_df[i,16] <- table_cv_both[3]
  test_df[i,17] <- table_cv_both[5]
  test_df[i,18] <- table_cv_both[6]
  test_df[i,19] <- lambda_CV_1SE
  test_df[i,20] <- table_cv1se_either[2]
  test_df[i,21] <- table_cv1se_either[3]
  test_df[i,22] <- table_cv1se_either[5]
  test_df[i,23] <- table_cv1se_either[6]
  test_df[i,24] <- table_cv1se_both[2]
  test_df[i,25] <- table_cv1se_both[3]
  test_df[i,26] <- table_cv1se_both[5]
  test_df[i,27] <- table_cv1se_both[6]
}


write.csv(test_df, "20p10000n_data")
```

```r
# Objective - To combine outputs for different n's and p's together.
# Methods: Use function to create 3 output files - df_10000n,
# df_1000n, df_100n, df_10n
# These files contain outputs for different p's which correspond
# to n=10000, 1000, 100, 10 r
# To be used for plotting of graphs

groupdf_tpr <- function(df) {
  df$rmse <- "rmse"
  df$cv_min <- "cv_min"
  df$cv1se <- "cv1se"
  df$e1 <- "E_1"
  df$e2 <- "E_2"

  df_rmse_e1 <- df[,c("RMSE_both_TPR","RMSE_both_FPR","rmse","e1")]
  colnames(df_rmse_e1) = c("TPR", "FPR", "method", "Estimated_E")
  df_cv_min_e1 <- df[,c("cv_min_both_TPR","cv_min_both_FPR","cv_min","e1")]
  colnames(df_cv_min_e1) = c("TPR", "FPR", "method", "Estimated_E")
  df_cv1se_e1 <- df[,c("cv1se_both_TPR","cv1se_both_FPR","cv1se","e1")]
  colnames(df_cv1se_e1) = c("TPR", "FPR", "method", "Estimated_E")

  df_rmse_e2 <- df[,c("RMSE_either_TPR","RMSE_either_FPR","rmse","e2")]
  colnames(df_rmse_e2) = c("TPR", "FPR", "method", "Estimated_E")
  df_cv_min_e2 <- df[,c("cv_min_either_TPR","cv_min_either_FPR","cv_min","e2")]
  colnames(df_cv_min_e2) = c("TPR", "FPR", "method", "Estimated_E")
  df_cv1se_e2 <- df[,c("cv1se_either_TPR","cv1se_either_FPR","cv1se","e2")]
  colnames(df_cv1se_e2) = c("TPR", "FPR", "method", "Estimated_E")

  df_all <- rbind(df_rmse_e1,df_cv_min_e1, df_cv1se_e1, df_rmse_e2,df_cv_min_e2, df_cv1se_e2)
}


#df5p100n <- read.csv("5p100n_data")
df20p100n <- read.csv("20p100n_data")
df50p100n <- read.csv("50p100n_data")
df100p100n <- read.csv("100p100n_data")

#df5p1000n <- read.csv("5p1000n_data")
```

```r
df20p1000n <- read.csv("20p1000n_data")
df50p1000n <- read.csv("50p1000n_data")
df100p1000n <- read.csv("100p1000n_data")

#df5p10000n <- read.csv("5p10000n_data")
df20p10000n <- read.csv("20p10000n_data")
df50p10000n <- read.csv("50p10000n_data")
df100p10000n <- read.csv("100p10000n_data")

#df5p10n <- read.csv("5p10n_data")
df20p10n <- read.csv("20p10n_data")
df50p10n <- read.csv("50p10n_data")
df100p10n <- read.csv("100p10n_data")

#for n =10000
#df_5p10000n <- groupdf_tpr(df5p10000n)
df_20p10000n <- groupdf_tpr(df20p10000n)
df_50p10000n <- groupdf_tpr(df50p10000n)
df_100p10000n <- groupdf_tpr(df100p10000n)

#df_5p10000n$parameters <- 5
#df_5p10000n$count <- 10000
df_20p10000n$parameters <- 20
df_20p10000n$count <- 10000
df_50p10000n$parameters <- 50
df_50p10000n$count <- 10000
df_100p10000n$parameters <- 100
df_100p10000n$count <- 10000

#for n =1000
#df_5p1000n <- groupdf_tpr(df5p1000n)
df_20p1000n <- groupdf_tpr(df20p1000n)
df_50p1000n <- groupdf_tpr(df50p1000n)
df_100p1000n <- groupdf_tpr(df100p1000n)

#df_5p1000n$parameters <- 5
#df_5p1000n$count <- 1000
df_20p1000n$parameters <- 20
df_20p1000n$count <- 1000
df_50p1000n$parameters <- 50
df_50p1000n$count <- 1000
df_100p1000n$parameters <- 100
df_100p1000n$count <- 1000

#for n =100
#df_5p100n <- groupdf_tpr(df5p100n)
df_20p100n <- groupdf_tpr(df20p100n)
df_50p100n <- groupdf_tpr(df50p100n)
df_100p100n <- groupdf_tpr(df100p100n)

#df_5p100n$parameters <- 5
#df_5p100n$count <- 100
df_20p100n$parameters <- 20
```

```
df_20p100n$count <- 100
df_50p100n$parameters <- 50
df_50p100n$count <- 100
df_100p100n$parameters <- 100
df_100p100n$count <- 100

#for n =10
#df_5p10n <- groupdf_tpr(df5p10n)
df_20p10n <- groupdf_tpr(df20p10n)
df_50p10n <- groupdf_tpr(df50p10n)
df_100p10n <- groupdf_tpr(df100p10n)

#df_5p10n$parameters <- 5
#df_5p10n$count <- 10
df_20p10n$parameters <- 20
df_20p10n$count <- 10
df_50p10n$parameters <- 50
df_50p10n$count <- 10
df_100p10n$parameters <- 100
df_100p10n$count <- 10

df_10000n <- rbind(df_20p10000n, df_50p10000n, df_100p10000n)
df_1000n <- rbind(df_20p1000n, df_50p1000n, df_100p1000n)
df_100n <- rbind(df_20p100n, df_50p100n, df_100p100n)
df_10n <- rbind(df_20p10n, df_50p10n, df_100p10n)
```

```
# boxplot for the performance measurement TPR
# Use each output file and generate boxplot for varied n.
# only attach code to draw plot for E_1. E_2 is generated by changing df_10000n$Estimated_E=="E_2")

ggplot(df_10000n[c(df_10000n$Estimated_E=="E_1"),], aes(x=method, y=TPR)) +
  geom_boxplot() +
  ggtitle("n=10000, p = 20, 50, 100") +
  facet_wrap(~parameters, scales="free")

ggplot(df_1000n[df_1000n$Estimated_E=="E_1",], aes(x=method, y=TPR)) +
  geom_boxplot() +
  ggtitle("n=1000, p = 20, 50, 100") +
  facet_wrap(~parameters, scales="free")

ggplot(df_100n[df_100n$Estimated_E=="E_1",], aes(x=method, y=TPR)) +
  geom_boxplot() +
  ggtitle("n = 100, p = 20, 50, 100") +
  facet_wrap(~parameters, scales="free")

ggplot(df_10n[df_10n$Estimated_E=="E_1",], aes(x=method, y=TPR)) +
  geom_boxplot() +
  ggtitle("n = 10, p = 20, 50, 100") +
  facet_wrap(~parameters, scales="free")
```

```
# boxplot for the performance measurement FPR
# Use each output file and generate boxplot for varied n.
# only attach code to draw plot for E_1. E_2 is generated by changing df_10000n$Estimated_E=="E_2")
```

```r
ggplot(df_10000n[df_10000n$Estimated_E=="E_1",], aes(x=method, y=FPR)) +
  geom_boxplot() +
  ggtitle("n=10000, p=5, 20, 50, 100") +
  facet_wrap(~parameters, scales="free")

ggplot(df_1000n[df_1000n$Estimated_E=="E_1",], aes(x=method, y=FPR)) +
  geom_boxplot() +
  ggtitle("n=1000, p=5, 20, 50, 100") +
  facet_wrap(~parameters, scales="free")

ggplot(df_100n[df_100n$Estimated_E=="E_1",], aes(x=method, y=FPR)) +
  geom_boxplot() +
  ggtitle("n=100, p=5, 20, 50, 100") +
  facet_wrap(~parameters, scales="free")

ggplot(df_10n[df_10n$Estimated_E=="E_1",], aes(x=method, y=FPR)) +
  geom_boxplot() +
  ggtitle("n=10, p=5, 20, 50, 100") +
  facet_wrap(~parameters, scales="free")
```

```r
# Combine data for E1, E2, E3.
# for comparison of performance measures of TPR, FPR, MCE
glassomodel <- read.csv("50repeatsKfoldwithMCE.csv")
glassomodel <- glassomodel[,-1]
glassomodel1 <- glassomodel1
install.packages("reshape2")
library(reshape2)
glassomodel1$optimalrates
glassomodel2 <- recast(glassomodel1, p + n + optRho + variable ~ optimalrates,
               id.var = c("optimalrates", "p", "n", "optRho"))
glassomodel2$Estimated_E <- "E_3"
glassomodel2$method <- "loglikhood"

groupdf1 <- function(df) {
  df$rmse <- "rmse"
  df$cv_min <- "cv_min"
  df$cv1se <- "cv1se"
  df$e1 <- "E_1"
  df$e2 <- "E_2"

  df_rmse_e1 <- df[,c("RMSE_both_TP", "RMSE_both_FP","RMSE_both_FN", "RMSE_both_TN", "RMSE_both_TPR", "R
  colnames(df_rmse_e1) = c("TP", "FP", "FN", "TN", "TPR", "FPR", "method", "Estimated_E")
  df_cv_min_e1 <- df[,c("cv_min_both_TP", "cv_min_both_FP","cv_min_both_FN", "cv_min_both_TN", "cv_min_
  colnames(df_cv_min_e1) = c("TP", "FP", "FN", "TN", "TPR", "FPR", "method", "Estimated_E")
  df_cv1se_e1 <- df[,c("cv1se_both_TP", "cv1se_both_FP","cv1se_both_FN", "cv1se_both_TN", "cv1se_both_TP
  colnames(df_cv1se_e1) = c("TP", "FP", "FN", "TN", "TPR", "FPR", "method", "Estimated_E")

  df_rmse_e2 <- df[,c("RMSE_either_TP", "RMSE_either_FP","RMSE_either_FN", "RMSE_either_TN", "RMSE_eithe
  colnames(df_rmse_e2) = c("TP", "FP", "FN", "TN", "TPR", "FPR", "method", "Estimated_E")
  df_cv_min_e2 <- df[,c("cv_min_either_TP", "cv_min_either_FP","cv_min_either_FN", "cv_min_either_TN",
  colnames(df_cv_min_e2) = c("TP", "FP", "FN", "TN", "TPR", "FPR", "method", "Estimated_E")
  df_cv1se_e2 <- df[,c("cv1se_either_TP", "cv1se_either_FP","cv1se_either_FN", "cv1se_either_TN", "cv1se
  colnames(df_cv1se_e2) = c("TP", "FP", "FN", "TN", "TPR", "FPR", "method", "Estimated_E")
  df_all <- rbind(df_rmse_e1,df_cv_min_e1, df_cv1se_e1, df_rmse_e2,df_cv_min_e2, df_cv1se_e2)
}
colnames(test_df) <- c("lambda_rmse", "RMSE_either_TP", "RMSE_either_FP","RMSE_either_FN", "RMSE_either
                       "RMSE_both_TP", "RMSE_both_FP","RMSE_both_FN", "RMSE_both_TN", "RMSE_both_TPR",
                       "lambda_cv_min", "cv_min_either_TP", "cv_min_either_FP","cv_min_either_FN", "cv_r
                       "cv_min_both_TP", "cv_min_both_FP","cv_min_both_FN", "cv_min_both_TN", "cv_min_bo
                       "lambda_cv1se", "cv1se_either_TP", "cv1se_either_FP","cv1se_either_FN", "cv1se_ei
                       "cv1se_both_TP", "cv1se_both_FP","cv1se_both_FN", "cv1se_both_TN", "cv1se_both_TI



fm_20p100n <- groupdf1(df20p100n)
fm_50p100n <- groupdf1(df50p100n)
fm_20p1000n <- groupdf1(df20p1000n)
fm_50p1000n <- groupdf1(df50p1000n)
fm_100p1000n <- groupdf1(df100p1000n)
fm_20p10000n <- groupdf1(df20p10000n)
fm_50p10000n <- groupdf1(df50p10000n)
fm_100p10000n <- groupdf1(df100p10000n)

fm_20p100n$parameters <- 20
fm_20p100n$count <- 100
fm_50p100n$parameters <- 50
fm_50p100n$count <- 100
fm_20p1000n$parameters <- 20
fm_20p1000n$count <- 1000
fm_50p1000n$parameters <- 50
fm_50p1000n$count <- 1000
fm_100p1000n$parameters <- 100
fm_100p1000n$count <- 1000
fm_20p10000n$parameters <- 20
fm_20p10000n$count <- 10000
fm_50p10000n$parameters <- 50
fm_50p10000n$count <- 10000
fm_100p10000n$parameters <- 100
fm_100p10000n$count <- 10000

finaldf <- rbind(fm_100p10000n, fm_100p1000n, fm_20p10000n, fm_20p1000n, fm_20p100n, fm_50p10000n,
                 fm_50p1000n, fm_50p100n)

finaldf$MCE <- (finaldf$FP+finaldf$FN) / (finaldf$TP+finaldf$TN+finaldf$FN+finaldf$FP)

findaldf1 <-finaldf[,c("TPR", "FPR", "MCE", "method", "Estimated_E", "parameters", "count")]
colnames(findaldf1) = c("TPR", "FPR", "MCE", "method", "Estimated_E", "p", "n")
glassomodel3 <-glassomodel2[,c("p", "n", "Estimated_E", "MCE", "FPR", "TPR", "method")]
colnames(glassomodel3) = c("p", "n", "Estimated_E", "MCE", "FPR", "TPR", "method")

alldf <- rbind(findaldf1, lau3)
write.csv(alldf, "groupdf")

aggregate(MCE ~ method + Estimated_E + p +n, data=alldf, mean)
```

Graphical Lasso

```r
# Codefor true edge and confusion matrix
# almost exactly the same to determine true edge set and confusion matrix
# Tiny alterations to ensure it works on glasso
true_edge <- function(theta){
  numOfDims <- ncol(theta)
  theta <- data.frame(theta)
  tf <- data.frame(lapply(theta, function(x) {x!=0}))
  colnames(tf) <- seq(numOfDims)
  return(tf)
}
confusion_matrix <- function(estimate_edge, trueEdge){
  numOfDims <- nrow(estimate_edge)
  confusion <- list(TP = 0,FP = 0,FN = 0,TN = 0,TP_rate = 0,FP_rate = 0)
  for (i in seq(numOfDims-1)){
    for (j in seq(numOfDims)[seq(numOfDims)>i]){
      if (trueEdge[i,j] == TRUE){
        if (estimate_edge[i,j] == TRUE & estimate_edge[j,i] == TRUE){
          confusion$TP <- confusion$TP + 1
        } else {
          confusion$FN <- confusion$FN + 1
        }
      } else {
        if (estimate_edge[i,j] == TRUE & estimate_edge[j,i] == TRUE){
          confusion$FP <- confusion$FP + 1
        } else {
          confusion$TN <- confusion$TN + 1
        }
      }
    }
  }
  confusion$TP_rate <- confusion$TP/(confusion$TP + confusion$FN)
  confusion$FP_rate <- confusion$FP/(confusion$FP + confusion$TN)
  return(confusion)
}
```

```r
# Below is the code for the ROC curve for E_3
# Very similar to that of node-wise but two extra params included.
ROC_curve <- function(data_set, theta, lambdamin = 0, lambdamax = 1, lambdacount = 200){

  numOfDims <- ncol(data_set)

  max_lambda = lambdamax
  min_lambda = lambdamin
  points <- data.frame(matrix(ncol = 3, nrow = lambdacount))

  lambda_seq <- seq(from = min_lambda, to = max_lambda, length.out= lambdacount)
  #print(lambda_seq)
  points[,3] <- lambda_seq
  trueedge <- true_edge(theta)

  for (i in seq(length(lambda_seq))){
    print(c('Calculating points:',i))
    glassotry <- glasso(s, rho = lambda_seq[i])
    estimate_edge <- true_edge(glassotry$wi)
    confusion <- confusion_matrix(estimate_edge, trueedge)
    #print(confusion)
    TPR <- confusion$TP_rate
    FPR <- confusion$FP_rate
    #print(TPR)
    #print(FPR)
    points[i,1] <- TPR
    points[i,2] <- FPR
  }
  colnames(points) <- c("TPR","FPR","Lambda")
  return(points)
}
```

```r
#take a range of values of n(10,100,1000,10000) and p(5,20,50,100),
# 12 combinations in total manually change the value of p and n
testsample <- simulation(5,1000)

testdata <- testsample$data

testtheta <-testsample$standardtheta

trueedge <- true_edge(testtheta)

s <- cov(testdata)

roc <- ROC_curve(testdata, testtheta, lambdamin = 0, lambdamax = 0.2, 200)

plot1=ggplot(roc, aes(FPR, TPR)) + geom_step() + xlim(0,1) + ylim(0,1)  + geom_point(alpha = 0.3)
plot2=ggplot(roc, aes(FPR, TPR)) + geom_step() + xlim(0,1) + ylim(0,1)  + geom_point(alpha = 0.3)
plot3=ggplot(roc, aes(FPR, TPR)) + geom_step() + xlim(0,1) + ylim(0,1)  + geom_point(alpha = 0.3)
plot4=ggplot(roc, aes(FPR, TPR)) + geom_step() + xlim(0,1) + ylim(0,1)  + geom_point(alpha = 0.3)

plot5=ggplot(roc, aes(FPR, TPR)) + geom_step() + xlim(0,1) + ylim(0,1)  + geom_point(alpha = 0.3)
plot6=ggplot(roc, aes(FPR, TPR)) + geom_step() + xlim(0,1) + ylim(0,1)  + geom_point(alpha = 0.3)
plot7=ggplot(roc, aes(FPR, TPR)) + geom_step() + xlim(0,1) + ylim(0,1)  + geom_point(alpha = 0.3)
plot8=ggplot(roc, aes(FPR, TPR)) + geom_step() + xlim(0,1) + ylim(0,1)  + geom_point(alpha = 0.3)

plot9=ggplot(roc, aes(FPR, TPR)) + geom_step() + xlim(0,1) + ylim(0,1)  + geom_point(alpha = 0.3)
plot10=ggplot(roc, aes(FPR, TPR)) + geom_step() + xlim(0,1) + ylim(0,1)  + geom_point(alpha = 0.3)
plot11=ggplot(roc, aes(FPR, TPR)) + geom_step() + xlim(0,1) + ylim(0,1)  + geom_point(alpha = 0.3)
plot12=ggplot(roc, aes(FPR, TPR)) + geom_step() + xlim(0,1) + ylim(0,1)  + geom_point(alpha = 0.3)
library(gridExtra)
grid.arrange(plot5,plot6,plot7,plot8,ncol=2,nrow=2)
#Calculating the areas under ROC curves by taking the mean of 50 simulations
Result1 = vector()
Result2 = vector()
Result3 = vector()
Result4 = vector()
Result5 = vector()
Result6 = vector()
Result7 = vector()
Result8 = vector()
```

```r
Result9 = vector()
Result10 = vector()
Result11 = vector()
Result12 = vector()
for(i in seq(50)){
  testsample <- simulation(20,10)

  testdata <- testsample$data

  testtheta <-testsample$standardtheta

  trueedge <- true_edge(testtheta)

  s <- cov(testdata)

  roc <- ROC_curve(testdata, testtheta, lambdamin = 0, lambdamax = 0.2, 200)

  a = AUC(roc$FPR,roc$TPR,from = min(roc$FPR),to = max(roc$FPR))
  Result1[i] = a
}
#mean1 takes too long
mean2 = mean(Result2)
mean3 = mean(Result3)
mean4 = mean(Result4)
#mean5 takes too long
mean6 = mean(Result6)
mean7 = mean(Result7)
mean8 = mean(Result8)
#mean9 takes too long
mean10 = mean(Result10)
mean11 = mean(Result11)
mean12 = mean(Result12)
```

```r
# Below Rho can be determined for different p and n using 3-fold cross validation.
# Works for n > p, otherwise fails or yields incosistent results.
determineRhoK <- function(p,n){
  sampleData <- simulation(p = p, n = (3/2)*n)
  # note the multiplication by 3/2 to ensure glasso is trained on n, instead of (2/3)*n
  dat <- sampleData$data
  ll <- data.frame(rho = vector(length=300), logl1 = vector(length = 300),
                   logl2 = vector(length = 300), logl3 = vector(length = 300))
  k <- 3
  folds <- sample(rep(1:k, length=n))

  for(ki in 1:3){
    sTrain <- cov(dat[which(folds!=ki),])
    sTest <- cov(dat[which(folds==ki),])
    count <- 0
    for(rhos in seq(0.001,0.3,0.001)){
      count <- count + 1
      glassoOutput <- glasso(sTrain, rho = rhos)
      modelPrecMat <- glassoOutput$wi
      ll[count,ki + 1] <- log(det(modelPrecMat)) - sum(diag(sTest %*% modelPrecMat))
      ll[count, 1] <- rhos
    }
  }
  ll$mean <- rowMeans(ll[,c(2:4)])
  index <- which.max(ll$mean)
  rhoBest <- ll$rho[index]
  return(rhoBest)
}
```

```r
# Below EBICglasso is implemented, only for n = 10000.
# Manually rerun for all p.
library(qgraph)

tprfpr100 <- data.frame(tpr = vector(length = 50), fpr = vector(length = 50))
for(i in 1:50){
  sampleDataEbic <- simulation(p = 100, n = 10000)
  dat <- sampleDataEbic$data
  s <- cov(dat)
  ebic <- EBICglasso(s, nrow(dat), returnAllResults = TRUE,
                     nlambda = 100, lambda.min.ratio = 0.001, gamma = 0)
  optLambdaEBIC <- ebic$lambda[which.min(ebic$ebic)]
  optMatrixEBIC <- ebic$optnet
  estimated <- true_edge(optMatrixEBIC)
  trueTet <- true_edge(sampleDataEbic$standardtheta)
  cm <- confusion_matrix(estimated,trueTet)
  #EBICgraph <- qgraph(optMatrixEBIC, layout = "spring", title = "EBIC")
  tprfpr100[i,1] <- cm$TP_rate
  tprfpr100[i,2] <- cm$FP_rate
}
ebictf <- data.frame('p20' = tprfpr20, 'p50' = tprfpr50, 'p100' = tprfpr100)
write.csv(ebictf, file = 'ebictprfpr.csv')

mean(ebictf$p100.fpr)
sd(ebictf$p50.fpr)
```

```r
# Finding the TPR, FPR and MCE for all p and n.
p <- c(rep(20,9), rep(50,9), rep(100,6))
n <- c(rep(c(rep(100,3),rep(1000,3), rep(10000,3)),2),rep(1000,3),rep(10000,3))
rates <- rep(c("TPR", "FPR","MCE"), 8)
optRho <- c(rep(0.1692,3),rep(0.0415,3),rep(0.0115,3), rep(0.2054,3),rep(0.0465,3),rep(0.0121,3),rep(0.0
ratedf <- data.frame(p = p, n = n, optimalrates = rates, optRho = optRho)
for(i in 5:54){
  ratedf[,i] <- NA
}

for(i in 1:8){
  pi <- ratedf$p[3*i]
  ni <- ratedf$n[3*i]
  rhoi <- ratedf$optRho[3*i]
  for(j in 1:50){
    sampleData <- simulation(p = pi, n = ni)
    trueTheta <- sampleData$standardtheta
    sAll <- cov(sampleData$data)
    glassoBest <- glasso(sAll, rhoi)
    true <- true_edge(trueTheta)
    modelfit <- true_edge(glassoBest$wi)
    cm <- confusion_matrix(modelfit,true)
    MCE <- (cm$FP + cm$FN)/(cm$FP + cm$FN + cm$TP + cm$TN)
    ratedf[3*i-2,j+4] <- cm$TP_rate
    ratedf[3*i-1,j+4] <- cm$FP_rate
    ratedf[3*i,j+4] <- MCE
  }
}

#write.csv(ratedf,'50repeatsKfoldwithMCE.csv')
# Printing relevant measures

library(readr)
kfold <- read.csv('50repeatsKfoldwithMCE.csv')
kfold <- kfold[,-1]
kfold <- ratedf
for(i in 1:24){
  cat('For p = ' ,kfold[i,1],' and n = ' ,kfold[i,2],' ')
  cat('mean = ' ,mean(t(kfold[i,c(5:54)])),' ')
  cat('sd = ' ,sd(t(kfold[i,c(5:54)])))
  print('')
}
```

```
# Boxplots for E_3,
# transformWen comes from transformation code from node-wise part
dfBP <- read.csv('transformWen.csv')
dfBP <- dfBP[,-1]
dfBP <- dfBP[,-4]
dfBP$p <- as.factor(dfBP$p)
dfBPn100 <- filter(dfBP, n == 100)
dfBPn1000 <- filter(dfBP, n == 1000)
dfBPn10000 <- filter(dfBP, n == 10000)


#TPR
pn100TPR <- ggplot(dfBPn100, aes(x=method, y=TPR)) +
  geom_boxplot(color = 'blue') +
  ggtitle("n= 100") + ylim(0,1)
pn1000TPR <- ggplot(dfBPn1000, aes(x=method, y=TPR)) +
  geom_boxplot(color = 'blue') +
  ggtitle("n= 1000") + ylim(0,1)
pn10000TPR <- ggplot(dfBPn10000, aes(x=method, y=TPR)) +
  geom_boxplot(color = 'blue') +
  ggtitle("n= 10000") + ylim(0,1)


#FPR
pn100FPR <- ggplot(dfBPn100, aes(x=method, y=FPR)) +
  geom_boxplot(color = 'blue') +
  ggtitle("n= 100") + ylim(0,0.5)
pn1000FPR <- ggplot(dfBPn1000, aes(x=method, y=FPR)) +
  geom_boxplot(color = 'blue') +
  ggtitle("n= 1000") + ylim(0,0.5)
pn10000FPR <- ggplot(dfBPn10000, aes(x=method, y=FPR)) +
  geom_boxplot(color = 'blue') +
  ggtitle("n= 10000") + ylim(0,0.5)

#MCE
pn100MCE <- ggplot(dfBPn100, aes(x=p, y=MCE)) +
  geom_boxplot(color = 'blue') +
  ggtitle("n= 100") + ylim(0,0.5)
pn1000MCE <- ggplot(dfBPn1000, aes(x=p, y=MCE)) +
  geom_boxplot(color = 'blue') +
  ggtitle("n= 1000") + ylim(0,0.5)
pn10000MCE <- ggplot(dfBPn10000, aes(x=p, y=MCE)) +
  geom_boxplot(color = 'blue') +
  ggtitle("n= 10000") + ylim(0,0.5)

library(gridExtra)
grid.arrange(pn100TPR, pn1000TPR, pn10000TPR, ncol = 3)
gTPR <- arrangeGrob(pn100TPR, pn1000TPR, pn10000TPR, ncol = 3) #generates g
ggsave(file="TPRbp.pdf", gTPR)

grid.arrange(pn100FPR, pn1000FPR, pn10000FPR, ncol = 3)
gFPR <- arrangeGrob(pn100FPR, pn1000FPR, pn10000FPR, ncol = 3) #generates g
ggsave(file="FPRbp.pdf", gFPR)

grid.arrange(pn100MCE, pn1000MCE, pn10000MCE, ncol = 3)
gMCE <- arrangeGrob(pn100MCE, pn1000MCE, pn10000MCE, ncol = 3) #generates g
ggsave(file="MCEbp.pdf", gMCE)
```

```
# Boxplots comparing E1 E2 E3
dfAll <- read.csv('E1E2E3withMCE.csv')
dfAll <- dfAll[,-1]
dfAll$p <- as.factor(dfAll$p)
dfAllTPR <- filter(dfAll, method == 'cv_min' | method == 'loglikhood')
dfAllFPR <- filter(dfAll, method == 'cv1se' | method == 'loglikhood')

dfAllTPRn100 <- filter(dfAllTPR, n == 100)
dfAllTPRn1000 <- filter(dfAllTPR, n == 1000)
dfAllTPRn10000 <- filter(dfAllTPR, n == 10000)

dfAllFPRn100 <- filter(dfAllFPR, n == 100)
dfAllFPRn1000 <- filter(dfAllFPR, n == 1000)
dfAllFPRn10000 <- filter(dfAllFPR, n == 10000)

#TPR
pn100TPR <- ggplot(dfAllTPRn100, aes(x=Estimated_E, y=TPR)) +
  geom_boxplot(aes(fill = p)) +
  ggtitle("n = 100") + ylim(0,1)
pn1000TPR <- ggplot(dfAllTPRn1000, aes(x=Estimated_E, y=TPR)) +
  geom_boxplot(aes(fill = p)) +
  ggtitle("n = 1000") + ylim(0,1)
pn10000TPR <- ggplot(dfAllTPRn10000, aes(x=Estimated_E, y=TPR)) +
  geom_boxplot(aes(fill = p)) +
  ggtitle("n = 10000") + ylim(0,1)
```

```r
#FPR
pn100FPR <- ggplot(dfAllFPRn100, aes(x=Estimated_E, y=FPR)) +
  geom_boxplot(aes(fill = p)) +
  ggtitle("n = 100") + ylim(0,0.5)
pn1000FPR <- ggplot(dfAllFPRn1000, aes(x=Estimated_E, y=FPR)) +
  geom_boxplot(aes(fill = p)) +
  ggtitle("n = 1000") + ylim(0,0.5)
pn10000FPR <- ggplot(dfAllFPRn10000, aes(x=Estimated_E, y=FPR)) +
  geom_boxplot(aes(fill = p)) +
  ggtitle("n = 10000") + ylim(0,0.5)

#MCE
pn100MCE <- ggplot(dfAllFPRn100, aes(x=Estimated_E, y=MCE)) +
  geom_boxplot(aes(fill = p)) +
  ggtitle("n = 100") + ylim(0,0.5)
pn1000MCE <- ggplot(dfAllFPRn1000, aes(x=Estimated_E, y=MCE)) +
  geom_boxplot(aes(fill = p)) +
  ggtitle("n = 1000") + ylim(0,0.5)
pn10000MCE <- ggplot(dfAllFPRn10000, aes(x=Estimated_E, y=MCE)) +
  geom_boxplot(aes(fill = p)) +
  ggtitle("n = 10000") + ylim(0,0.5)

library(gridExtra)
grid.arrange(pn100TPR, pn1000TPR, pn10000TPR, ncol = 3)
gTPR <- arrangeGrob(pn100TPR, pn1000TPR, pn10000TPR, ncol = 3) #generates g
ggsave(file="TPRbpALL.pdf", gTPR)

grid.arrange(pn100FPR, pn1000FPR, pn10000FPR, ncol = 3)
gFPR <- arrangeGrob(pn100FPR, pn1000FPR, pn10000FPR, ncol = 3) #generates g
ggsave(file="FPRbpALL.pdf", gFPR)

grid.arrange(pn100MCE, pn1000MCE, pn10000MCE, ncol = 3)
gMCE <- arrangeGrob(pn100MCE, pn1000MCE, pn10000MCE, ncol = 3) #generates g
ggsave(file="MCEbpALL.pdf", gMCE)
```