Estimation of graphical models using lasso-related approach

group 9

4/12/2017

Packages

```
library(MASS)
library(glmnet)
library(glasso)
library(zoo)
```

Function to generate theta

We need to generate a pxp symmetric positive definite matrix B. We sample each elements in the upper triangular matrix from a bernoulli with $p = 0.1$. We sum B with its transpose and now we have a symmetric matrix with the diagonal elements equal to 0.
If a matrix is symmetrical and strictly diagonally dominant then it is positive definite. Hence we put the elements of the diagonal equals to the maximum sum of the row plus 1. To standardize the matrix we divide by its diagonal elements.

```
rtheta=function(p)
{
  B=matrix(0,p,p)
  B[upper.tri(B, diag = FALSE)]=rbinom((p^2-p)/2,1,0.1)/2
  B=B+t(B)
  k=max(apply(B,1,sum))+1
  diag(B)=rep(k,p)
  B=B/B[1,1]
  return(B)
}
```

Node-wise lasso approach

This function takes a matrix x and a vector lambda. x are the fitting data and lambda is a vector of penalization term. It returns a matrix which a vector of prediction for each lambda.
To estimate this matrix it use the package "glmnet" to implement the node-wise lasso regression.

```
nodewise.lasso=function(x,lambda)
{
  #par.glmnet is a 3 dimensional array
  #its indexes are the variable x[i]
  #the estimated values for beta[i,j]
  #the levels if lambda

  p=ncol(x)
```

```
  par.glmnet=array(0,dim=c(p,p,length(lambda)))

  #We regress each X[i] on the remaining variables
  #we save the coefficients beta[i,j]
  #in the 3-dimensional array par.glmnet

  for(i in 1:p)
  {
    fit=glmnet(x[,-i],x[,i],lambda = lambda,intercept = FALSE)
    for(j in 1:p)
    {
      if(j<i) par.glmnet[i,j,]=coef(fit)[j+1,]
      if(j>i) par.glmnet[i,j,]=coef(fit)[j,]
    }
  }

  #We estimate 1 if b[i,j]!=0
  par.glmnet[par.glmnet!=0]=1

  #We arrange the 3-dimensional array into a matrix
  #For each lambda we save the estimated edges in a column of pred
  pred=matrix(0,p*(p-1)/2,length(lambda))

  #For each lambda we sum the pxp matrix
  #which contains 1 if b[i,j]!=0 with its transpose.
  #The new matrix cointains
  #0 if b[i,j]=b[j,i]=0
  #1 if b[i,j]=0 or b[j,i]=0 (not both of them)
  #2 if b[i,j]!=0 and b[j,i]!=0
  for(i in 1:length(lambda))
  {
    mat=par.glmnet[,,(length(lambda)-i+1)]+t(par.glmnet[,,(length(lambda)-i+1)])
    pred[,i]=mat[upper.tri(mat)]
  }
  return(pred)
}
```

Graphical LASSO approach

This function takes a matrix x and a vector lambda. x are the fitting data and lambda is a vector of penalization term. It use the function "glassopath" from the package "glasso". This function allow to implement the graphical lasso regression for a vector of penalization term. It's output it is a matrix containing a $p(p-1)/2$ vector of prediction for each lambda.

```
graphical.lasso=function(x,lambda)
{
  p=ncol(x)
  var.x=var(x)
  fit.glasso=glassopath(var.x,lambda,penalize.diagonal=FALSE,trace=0)

  #For each lambda we save the estimation for the matrix theta in the column of pred
  pred=matrix(0,p*(p-1)/2,length(lambda))
```

```
  for (i in 1:length(lambda))
  {
    fit=fit.glasso$wi[,,i]
    pred[,i]=as.vector(fit[upper.tri(fit)])
  }

  #we predict 1 if the element is != 0
  pred[pred!=0]=1
  return(pred)
}
```

TPR and FPR calculation

This function takes as input the estimated vertices, the real vertices and the vector of lambda and it evaluate the true and false positive rate.
Since our predictions are vector of 0 and 1, we can use some trick to evaluate the TPR and the FPR. In particular $TPR=#{E'=1 && E=1}/#{E=1}. We have that {E'=1 && E=1} if and only if {E'E=1}. Hence, to evaluate the TPR it is possible to count the 1 in the vector E'E and divide it by the number of 1 in E. The FPR it has been calculated in a similar way.
The multiplication between vectors are elements by elements (according to the R definition).

```
error.rate=function(pred,edges,lambda)
{
  tpr=rep(0,length(lambda))
  fpr=rep(0,length(lambda))
  tpr=apply(pred*edges,2,sum)/sum(edges)
  fpr=apply(pred*(1-edges),2,sum)/sum(1-edges)
  return(cbind(tpr,fpr))
}
```

AUROC evaluation

This function evaluate the AUROC with the trapezium approximation.

```
aur=function(tpr,fpr)
{
  o=order(fpr)
  a=sum(rollmean(tpr[o],2)*diff(fpr[o]))
  return(a)
}
```

Main function

This function recall all the previous function: it generate the set of vertices, it predicts the vertices with the 3 estimation methods, it plots the miss-classification rate, it evaluates TPR, FPR, ROC and AUROC and it returns them as a list.

```r
one.time.run=function(p,n,lambda)
{
  theta=rtheta(p)
  #sigma is the inverse of theta
  sigma=solve(theta,sparse=TRUE)

  #Edges is the true response variable
  edges=as.vector(theta[upper.tri(sigma)])
  edges[edges!=0]=1

  #We generate a n random sample from a multivariate gaussian distribution with mean 0 and variance Sig
  x=mvrnorm(n,rep(0,p),sigma)

  #GLMNET package for Node-wise LASSO approach
  #Determining prediction for method 1 and 2
  #Method 1 predicts a vertex if pred=1 or pred=2
  #Method 2 predicts a vertex if pred=2

  pred=nodewise.lasso(x,lambda)
  #Method 1 predicts and edge if pred is equal to 1 or 2
  pred.1=matrix(0,p*(p-1)/2,length(lambda))
  pred.1[pred!=0]=1
  #Method 2 predicts and edge if pred is equal to 2
  pred.2=matrix(0,p*(p-1)/2,length(lambda))
  pred.2[pred==2]=1

  #GLASSO package for Graphical Lasso approach
  pred.3=graphical.lasso(x,lambda)

  #OUTPUT - Mis-classification rate
  #Graph
  if(p==100 && n==500)
  {
  par(mfrow=c(1,3))
  plot(lambda,2*apply(abs(pred.1-edges)/(p*(p-1)),2,sum),type="l",ylim=c(0,1),ylab = "Method 1",xlab="La
  mtext("Mis-classification rate", side = 3, line = 2, font=2)
  legend("topright",legend=c(p,n))
  plot(lambda,2*apply(abs(pred.2-edges)/(p*(p-1)),2,sum),type="l",ylim=c(0,1),ylab = "Method 2",xlab="La
  plot(lambda,2*apply(abs(pred.3-edges)/(p*(p-1)),2,sum),type="l",ylim=c(0,1),ylab = "Method 3",xlab="La
  par(mfrow=c(1,1))
  }

  #ROC and AUROC
  #positive.rate is a matrix cointaining the vector of true and false positive rate for each lambda

  positive.rate.1=error.rate(pred.1,edges,lambda)
  positive.rate.2=error.rate(pred.2,edges,lambda)
  positive.rate.3=error.rate(pred.3,edges,lambda)

  auroc.1=aur(positive.rate.1[,1],positive.rate.1[,2])
  auroc.2=aur(positive.rate.2[,1],positive.rate.2[,2])
  auroc.3=aur(positive.rate.3[,1],positive.rate.3[,2])
```

```
  li=list(list(pred.1,pred.2,pred.3),list(positive.rate.1,positive.rate.2,positive.rate.3),list(auroc.1
  return(li)
}
```

Main Function 2

This function is the same as the latter, but it doesn't plot any graph and it returns only the AUROC values.

```
no.graph.run=function(p,n,lambda)
{
  theta=rtheta(p)
  sigma=solve(theta,sparse=TRUE)

  edges=as.vector(theta[upper.tri(sigma)])
  edges[edges!=0]=1

  x=mvrnorm(n,rep(0,p),sigma)

  pred=nodewise.lasso(x,lambda)
  pred.1=matrix(0,p*(p-1)/2,length(lambda))
  pred.1[pred!=0]=1
  pred.2=matrix(0,p*(p-1)/2,length(lambda))
  pred.2[pred==2]=1

  pred.3=graphical.lasso(x,lambda)

  positive.rate.1=error.rate(pred.1,edges,lambda)
  positive.rate.2=error.rate(pred.2,edges,lambda)
  positive.rate.3=error.rate(pred.3,edges,lambda)

  auroc.1=aur(positive.rate.1[,1],positive.rate.1[,2])
  auroc.2=aur(positive.rate.2[,1],positive.rate.2[,2])
  auroc.3=aur(positive.rate.3[,1],positive.rate.3[,2])

  return(cbind(auroc.1,auroc.2,auroc.3))
}
```

Main Program

Parameters settings

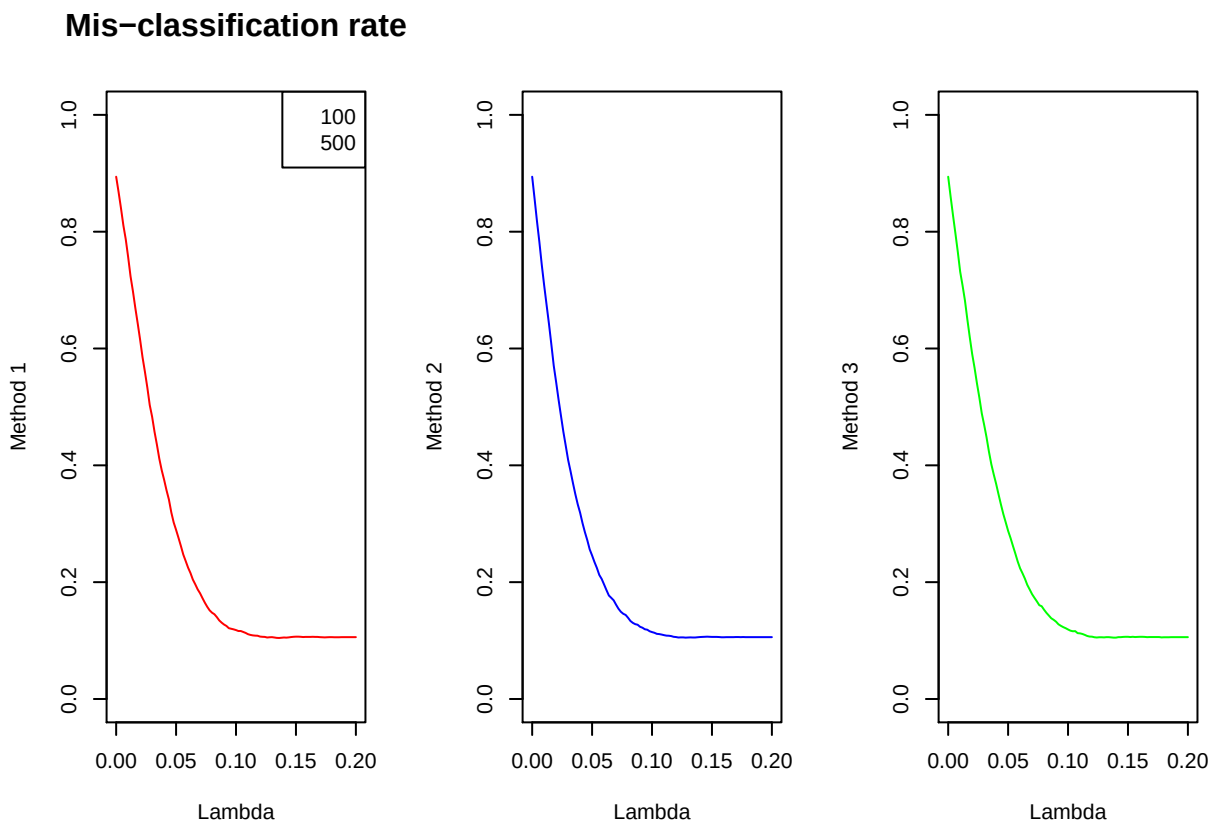
```
#The seed is the delivery date
set.seed(06122017)
#We define the vector lambda for the penality term
lambda=seq(0.0,0.2,by=0.002)
```

Estimation with different p and n

Result is a list. The following instruction explain how to recall the informations:

result.p.n[[*i*]][[*j*]]
j=1,2,3 is the method used
i=1 => prediction
i=2 => true positive rate and false negative rate i=3 => auroc

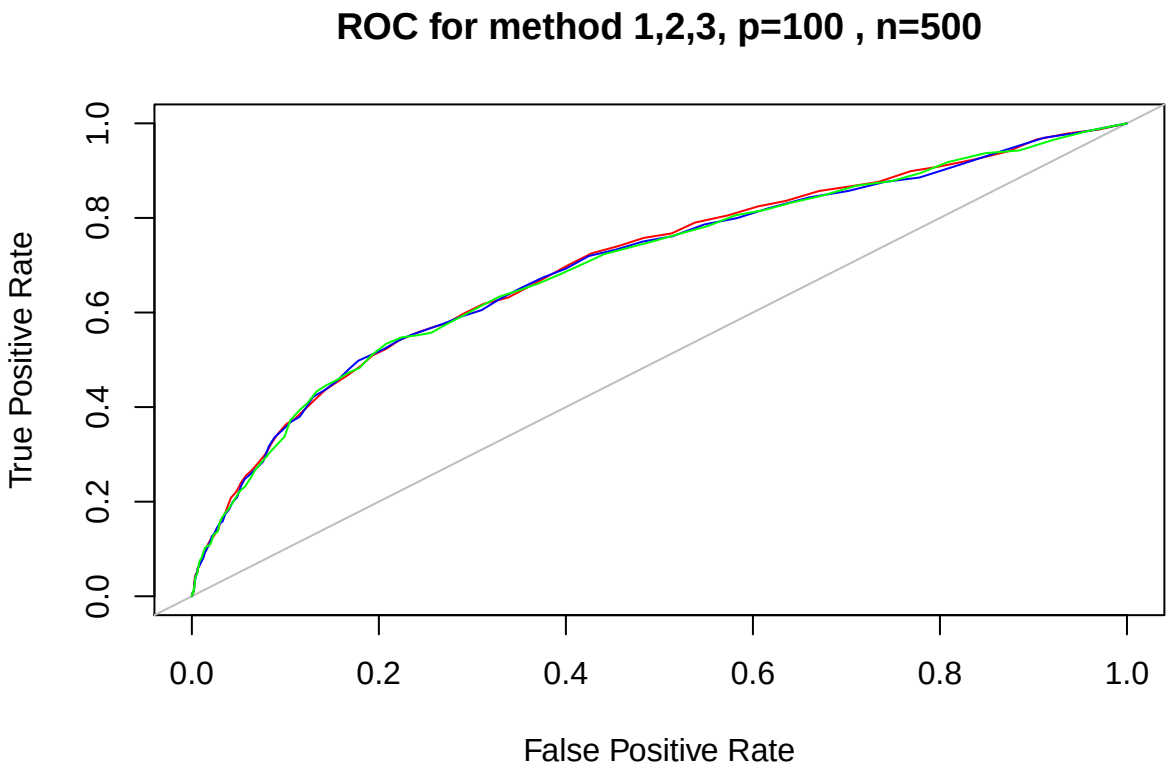
```
result.50.500=one.time.run(50,500,lambda)
result.100.200=one.time.run(100,200,lambda)
result.100.500=one.time.run(100,500,lambda)
```




```
result.100.1000=one.time.run(100,1000,lambda)
result.150.500=one.time.run(150,500,lambda)
```

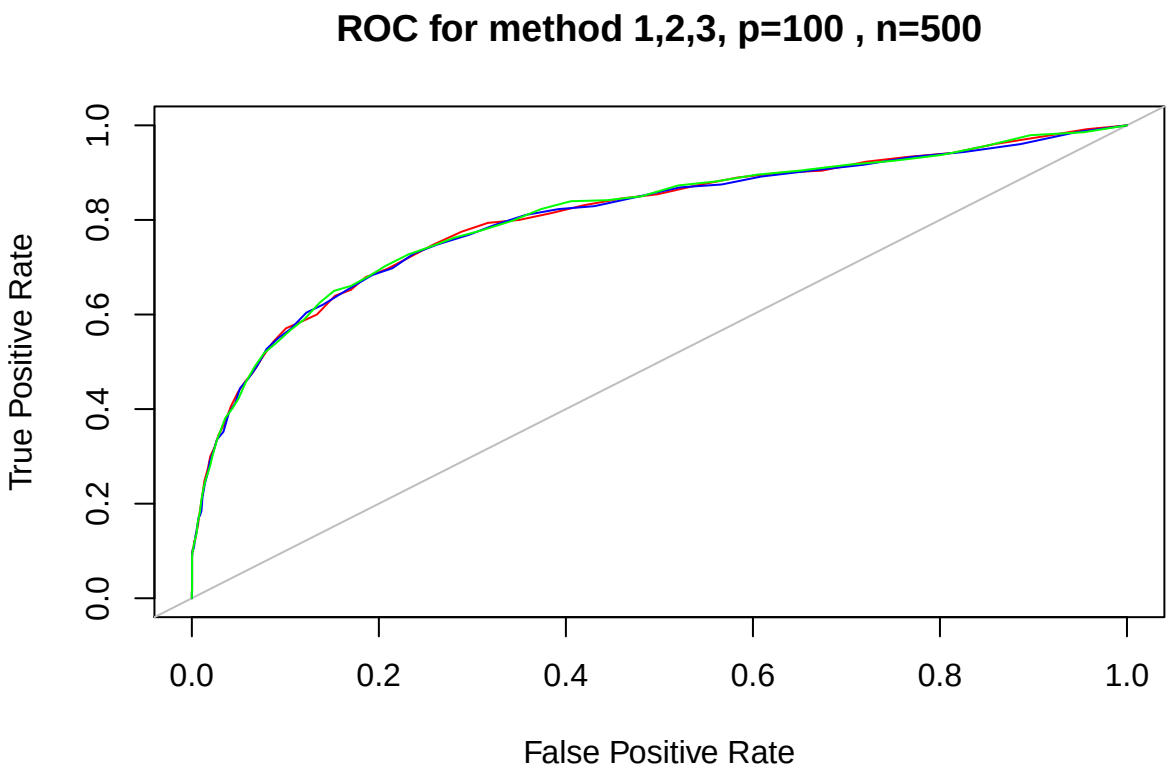
ROC curve across methods, p=100, n=500

```
plot(x = c(0, 1),y = c(0, 1),type = "n",main = "ROC for method 1,2,3, p=100 , n=500",xlab = "False Posi
abline(0,1,col="grey")
lines(result.100.500[[2]][[1]][,2],result.100.500[[2]][[1]][,1],col="red")
lines(result.100.500[[2]][[2]][,2],result.100.500[[2]][[2]][,1],col="blue")
lines(result.100.500[[2]][[3]][,2],result.100.500[[2]][[3]][,1],col="green")
```



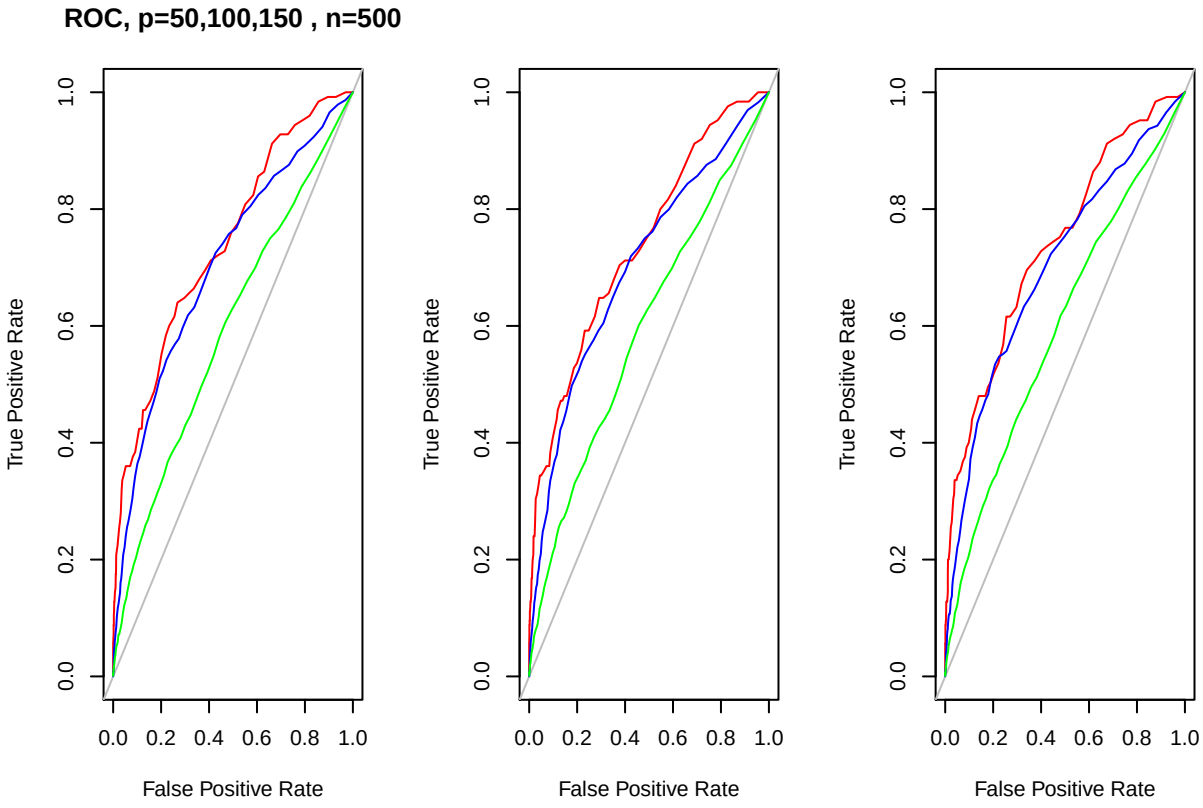
ROC curve across methods, p=100, n=1000

```
plot(x = c(0, 1),y = c(0, 1),type = "n",main = "ROC for method 1,2,3, p=100 , n=500",xlab = "False Posi
abline(0,1,col="grey")
lines(result.100.1000[[2]][[1]][,2],result.100.1000[[2]][[1]][,1],col="red")
lines(result.100.1000[[2]][[2]][,2],result.100.1000[[2]][[2]][,1],col="blue")
lines(result.100.1000[[2]][[3]][,2],result.100.1000[[2]][[3]][,1],col="green")
```



ROC curves, same n, p changes

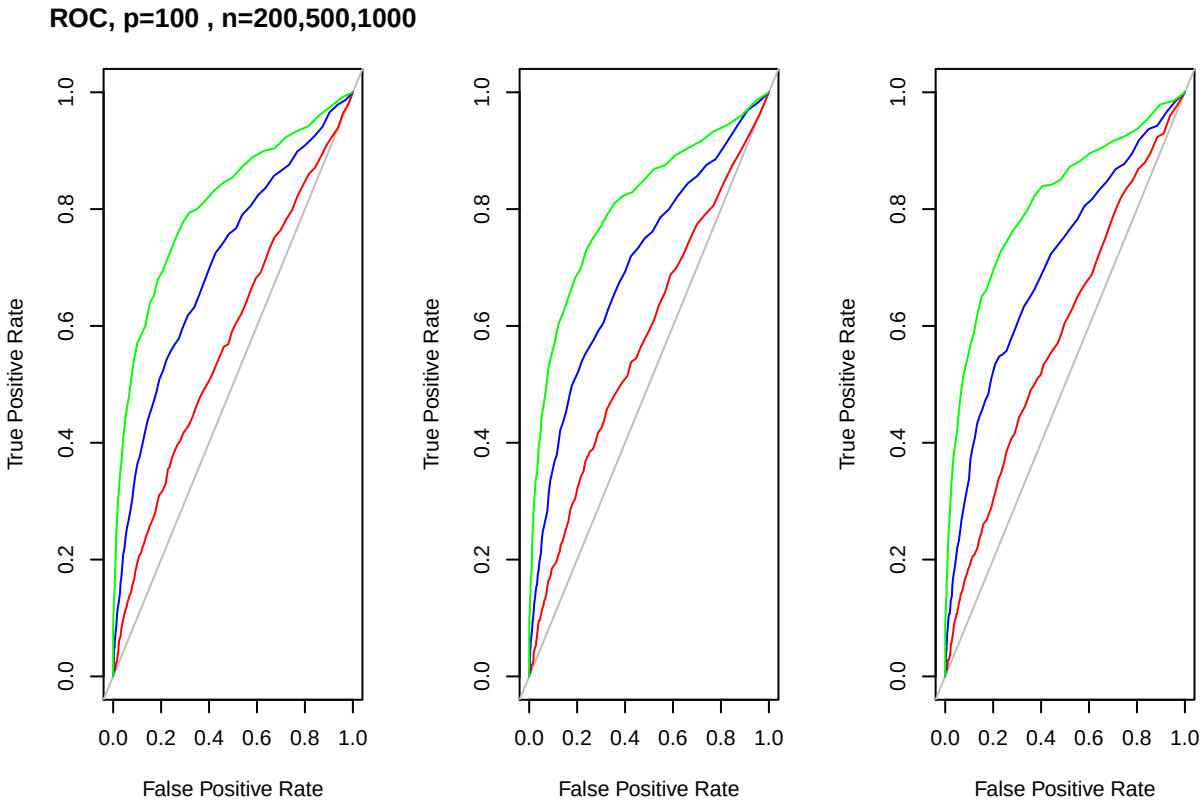
```
par(mfrow=c(1,3))
plot(x = c(0, 1),y = c(0, 1),type = "n",main = "ROC, p=50,100,150 , n=500",xlab = "False Positive Rate"
abline(0,1,col="grey")
lines(result.50.500[[2]][[1]][,2],result.50.500[[2]][[1]][,1],col="red")
lines(result.100.500[[2]][[1]][,2],result.100.500[[2]][[1]][,1],col="blue")
lines(result.150.500[[2]][[1]][,2],result.150.500[[2]][[1]][,1],col="green")
plot(x = c(0, 1),y = c(0, 1),type = "n",xlab = "False Positive Rate", ylab = "True Positive Rate")
abline(0,1,col="grey")
lines(result.50.500[[2]][[2]][,2],result.50.500[[2]][[2]][,1],col="red")
lines(result.100.500[[2]][[2]][,2],result.100.500[[2]][[2]][,1],col="blue")
lines(result.150.500[[2]][[2]][,2],result.150.500[[2]][[2]][,1],col="green")
plot(x = c(0, 1),y = c(0, 1),type = "n", xlab = "False Positive Rate", ylab = "True Positive Rate")
abline(0,1,col="grey")
lines(result.50.500[[2]][[3]][,2],result.50.500[[2]][[3]][,1],col="red")
lines(result.100.500[[2]][[3]][,2],result.100.500[[2]][[3]][,1],col="blue")
lines(result.150.500[[2]][[3]][,2],result.150.500[[2]][[3]][,1],col="green")
```




```
par(mfrow=c(1,1))
```

ROC curves, same p, n changes

```
par(mfrow=c(1,3))
plot(x = c(0, 1),y = c(0, 1),type = "n",main = "ROC, p=100 , n=200,500,1000",xlab = "False Positive Rat
abline(0,1,col="grey")
lines(result.100.200[[2]][[1]][,2],result.100.200[[2]][[1]][,1],col="red")
lines(result.100.500[[2]][[1]][,2],result.100.500[[2]][[1]][,1],col="blue")
lines(result.100.1000[[2]][[1]][,2],result.100.1000[[2]][[1]][,1],col="green")
plot(x = c(0, 1),y = c(0, 1),type = "n",xlab = "False Positive Rate", ylab = "True Positive Rate")
abline(0,1,col="grey")
lines(result.100.200[[2]][[2]][,2],result.100.200[[2]][[2]][,1],col="red")
lines(result.100.500[[2]][[2]][,2],result.100.500[[2]][[2]][,1],col="blue")
lines(result.100.1000[[2]][[2]][,2],result.100.1000[[2]][[2]][,1],col="green")
plot(x = c(0, 1),y = c(0, 1),type = "n",xlab = "False Positive Rate", ylab = "True Positive Rate")
abline(0,1,col="grey")
lines(result.100.200[[2]][[3]][,2],result.100.200[[2]][[3]][,1],col="red")
lines(result.100.500[[2]][[3]][,2],result.100.500[[2]][[3]][,1],col="blue")
lines(result.100.1000[[2]][[3]][,2],result.100.1000[[2]][[3]][,1],col="green")
```




```
par(mfrow=c(1,1))
```

50 times

We run the code fifty times to evaluate mean, variance and stardard deviation of the AUROC.

```
p=100
n=500
set.seed(06122017)
auroc=matrix(0,50,3)
for(t in 1:50)
{
  auroc[t,]=no.graph.run(p,n,lambda)
}
```

```
apply(auroc,2,mean)
```

```
## [1] 0.6512545 0.6500842 0.6533424
```

```
apply(auroc,2,sd)
```

```
## [1] 0.02899642 0.02859026 0.02875512
```

```
apply(auroc,2,var)
```

```
## [1] 0.0008407924 0.0008174031 0.0008268572
```