Financial Data Mining

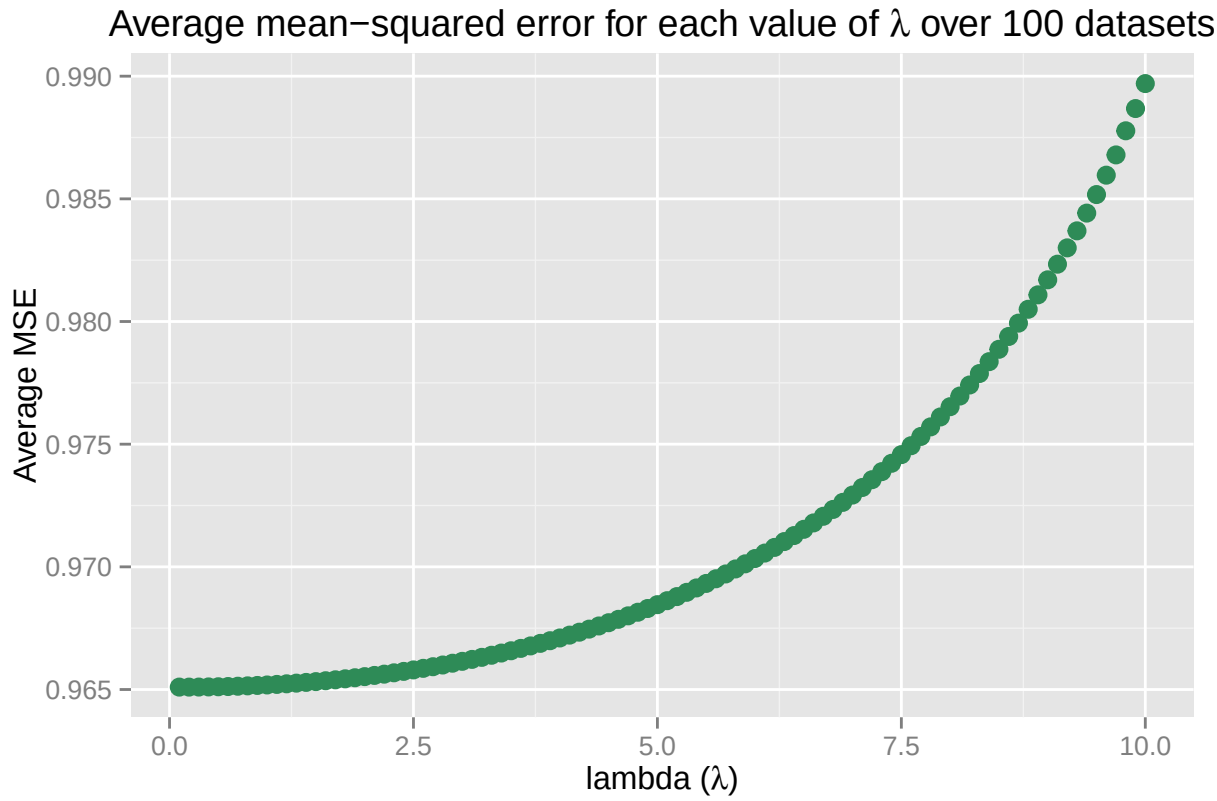
Homework 1
Yen-Hsiu Chang

8. Simulations with ridge regression

```
rm(list=ls(all=TRUE))
set.seed(100)
# Set parameters
N <- 500
p <- 10000
beta <- rep(1,10000)
lambda.grid <- seq(10,0.1,length=100)
```

```
## Loading required package: Matrix
## Loading required package: foreach
## Loaded glmnet 2.0-2
```

```
MSE <- matrix(NA,100,100)
# Run simulations with ridge regression
ptm <- proc.time()
for(i in 1:100){
  x <- matrix(rnorm(N*p),nrow=N)
  epsilon <- rnorm(N)
  y <- x%*%beta + epsilon
  ridge.mdl <- glmnet(x,y,alpha=0,lambda=lambda.grid,intercept=FALSE)
   # compute the mean-squared error (MSE)
  MSE[i,] <- apply((coef(ridge.mdl)[-1,]-beta)^2,2,mean)
}
total.time <- proc.time() - ptm

library(ggplot2)
# Compute average MSE for each value of lambda over the 100 datasets
MSE.avg <- apply(MSE,2,mean)

# Plot MSE for each value of lambda
g <- ggplot(data.frame(lambda.grid,MSE.avg),aes(lambda.grid,MSE.avg))+
  geom_point(pch=19,cex=3,color="seagreen")+
  labs(x=expression(paste("lambda (",lambda,")")),y="Average MSE",
       title=expression(paste("Average mean-squared error for each value of ",lambda," over 100 dataset
g
```




```
lambda.grid[which.min(MSE.avg)]
```

```
## [1] 0.2
```

When λ equals 0.2, it corresponds to the smallest average MSE and total run-time for the simulations is 376.93 seconds.

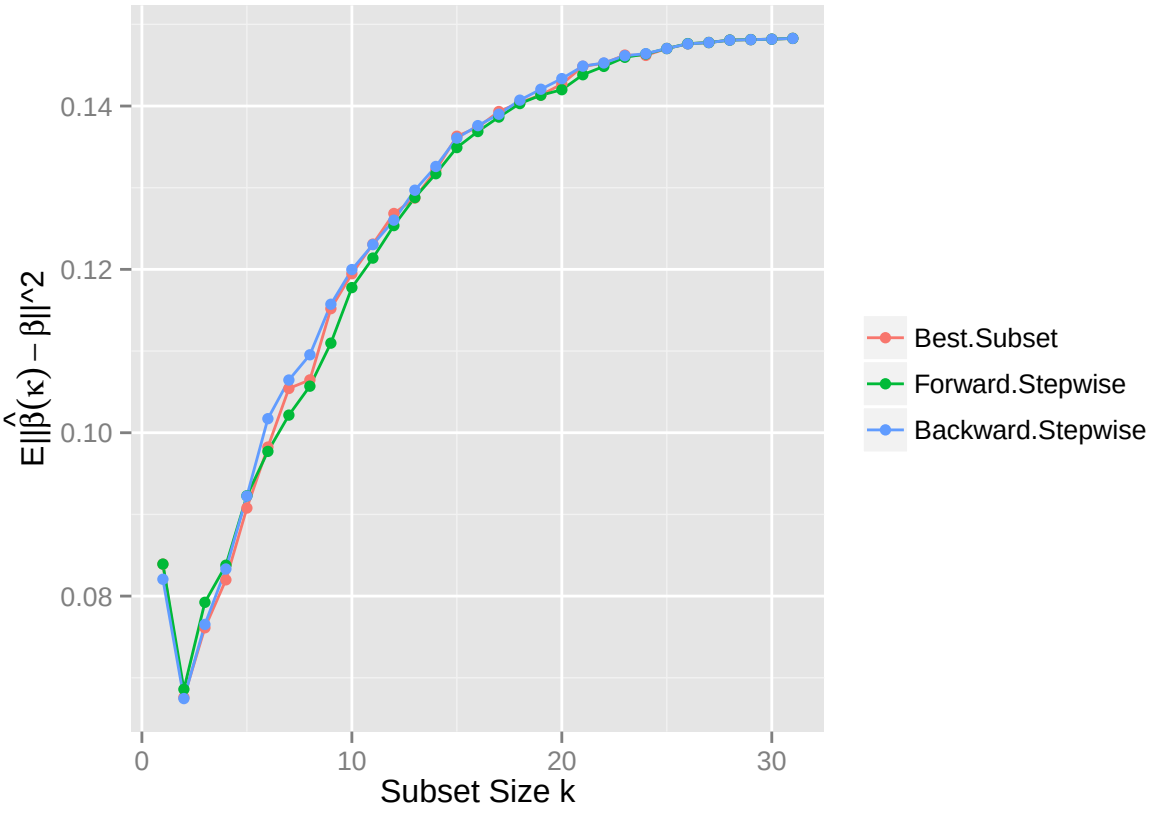
9. Simulations with subset selection

```
rm(list=ls(all=TRUE))
set.seed(50)
library(leaps)
N <- 300
p <- 31
mu <- rep(0,31)
sigma <- matrix(.85, nrow=p, ncol=p) + diag(p)*.15 # pairwise correlations 0.85
beta <- sample(c(rnorm(10,0,sqrt(0.4)),rep(0,21)))
# Run 50 simulations
library(MASS)
MSE.full <- matrix(NA,50,31)
MSE.fwd <- matrix(NA,50,31)
MSE.bwd <- matrix(NA,50,31)
for(i in 1:50){
  x <- mvrnorm(300,mu,sigma)
```

```
  epsilon <- rnorm(N,0,sqrt(6.25))
  y <- x%*%beta + epsilon
  regfit.full <- regsubsets(x,y,nvmax=p)
  regfit.fwd <- regsubsets(x,y,nvmax=p,method="forward")
  regfit.bwd <- regsubsets(x,y,nvmax=p,method="backward")
  m1 <- t(summary(regfit.full)$which)
  m1[m1==TRUE]<-unlist(coef(regfit.full,1:31))
  m2 <- t(summary(regfit.fwd)$which)
  m2[m2==TRUE]<-unlist(coef(regfit.fwd,1:31))
  m3 <- t(summary(regfit.bwd)$which)
  m3[m3==TRUE]<-unlist(coef(regfit.bwd,1:31))

  MSE.full[i,] <- apply((m1-c(0,beta))^2,2,mean)
  MSE.fwd[i,] <- apply((m2-c(0,beta))^2,2,mean)
  MSE.bwd[i,] <- apply((m3-c(0,beta))^2,2,mean)
}

MSE.full.avg <- apply((MSE.full),2,mean)
MSE.fwd.avg <- apply((MSE.fwd),2,mean)
MSE.bwd.avg <- apply((MSE.bwd),2,mean)
k <- 1:31

dataplot <- data.frame(k=k,"Best Subset"=MSE.full.avg,"Forward Stepwise"=MSE.fwd.avg,"Backward Stepwise
```

```
library(reshape)
```

```
dataplot <- melt(dataplot,id="k")
g <- ggplot(dataplot,aes(x=k,y=value,colour=variable))+
  geom_line()+
  geom_point()+
  labs(x="Subset Size k" ,y=expression(paste("E||",hat(beta)(kappa)-beta,"||^2")))+
  scale_color_hue(name="")
g
```



10. Zipcode data with all methods in Chapters 1-3

(a)

```
rm(list=ls(all=TRUE))
# Data
load("hw1.RData")
dim(train2); dim(train3)
```

```
## [1] 731 256
```

```
## [1] 658 256
```

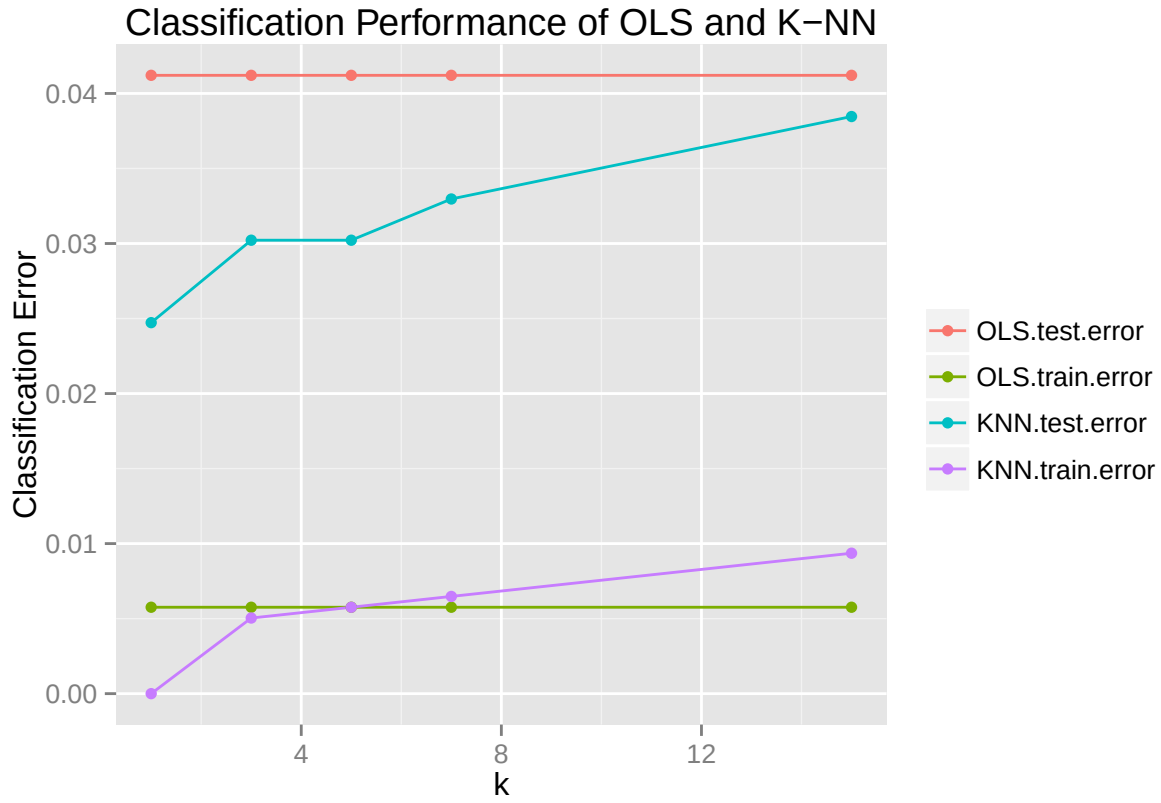
```
y.train <- c(rep(0,dim(train2)[1]),rep(1,dim(train3)[1]))
x.train <- rbind(train2,train3)
y.test <- c(rep(0,dim(test2)[1]),rep(1,dim(test3)[1]))
x.test <- rbind(test2,test3)
# OLS
lm1 <- lm(y.train~x.train)
y.hat.train <- predict(lm1)
y.hat.test <- cbind(1,x.test)%*%coef(lm1)
lm.train.error <- mean((y.hat.train>.5)!=y.train)
lm.test.error <- mean((y.hat.test>.5)!=y.test)
# k-NN
```

```
library(class)
k.vec <- c(1,3,5,7,15)
y.hat <- sapply(k.vec,function(k){knn(x.train, rbind(x.train,x.test),
                    y.train, k = k)})
knn.train.error <- colMeans((y.hat[1:length(y.train),]>.5)!=y.train)
knn.test.error <- colMeans((y.hat[(length(y.train)+1):nrow(y.hat),]>.5)!=y.test)

# Create an errors data frame
errors <- data.frame("OLS test error"=lm.test.error,"OLS train error"=lm.train.error,k=k.vec,"KNN test
```

The following table is the training and test error for linear regression and k-nearest neighbor classification

```
##   OLS.test.error OLS.train.error  k KNN.test.error KNN.train.error
## 1     0.04120879     0.005759539  1     0.02472527     0.000000000
## 2     0.04120879     0.005759539  3     0.03021978     0.005039597
## 3     0.04120879     0.005759539  5     0.03021978     0.005759539
## 4     0.04120879     0.005759539  7     0.03296703     0.006479482
## 5     0.04120879     0.005759539 15     0.03846154     0.009359251
```

```
# Plot the error curves
library(reshape)
errors <- melt(errors, id="k")
library(ggplot2)
p <- ggplot(errors,aes(x=k,y=value,colour=variable))+
  geom_line()+
  geom_point()+
  labs(x="k",y="Classification Error",title="Classification Performance of OLS and K-NN")+
  scale_color_hue(name="")
p
```



(b)

No, we cannot obtain the best subset selection for all k if we do not manipulate the arguments of the function 'regsubsets'. We have to specify `really.big=T`, so that it can perform exhaustive search on more than 50 variables.

```
#reg.best <- regsubsets(y~.,train)
```

(c)

No, it needs to take a very long time,because best subset selection is only feasible for small p, 30 or 40.

```
#reg.best2 <- regsubsets(y~.,train,really.big=T)
```

(d)

```
reg.best3 <- regsubsets(y~.,train,nvmax=3,really.big=T)
coef(reg.best3,3)
```

```
## (Intercept)      x.V104      x.V166      x.V249
##   0.4941865   0.1732184  -0.2354972   0.2287003
```

`nvmax` specifies maximum size of subsets to examine. Variable number 104, variable number 166, variable number 249 are retained by the best model with only 3 variables

(e)

```
reg.best4 <- regsubsets(y~.,train,really.big=T,method="forward")
#coef(reg.best4,9)
```

No, we have to specify the `method="forward"` and we cannot find out which variables are retained when considering models with 9 predictors, because the default maximum size of subsets to examine is 8.

(f)

```
reg.best5 <- regsubsets(y~.,train,nvmax=256,really.big=T,method="forward")
coef(reg.best5,9)
```

```
## (Intercept)      x.V104      x.V122      x.V124      x.V148      x.V165
##  0.33970687  0.12539662  0.11046186  0.08144285 -0.09140685 -0.10677300
##      x.V168      x.V183      x.V200      x.V249
## -0.10616192 -0.09385240 -0.07602215  0.10295150
```

```
coef(reg.best5,3)
```

```
## (Intercept)      x.V165      x.V168      x.V249
##   0.4051336  -0.1744623  -0.2142737   0.2217201
```

Yes, variable number 104,122,124,148,165,168,183,200,249 are retined. And variable number 165,168,249 are retained when we consider models with 3 predictors.

(g)

```
reg.best6 <- regsubsets(y~.,train,nvmax=256,really.big=T,method="backward")
coef(reg.best6,3)
```

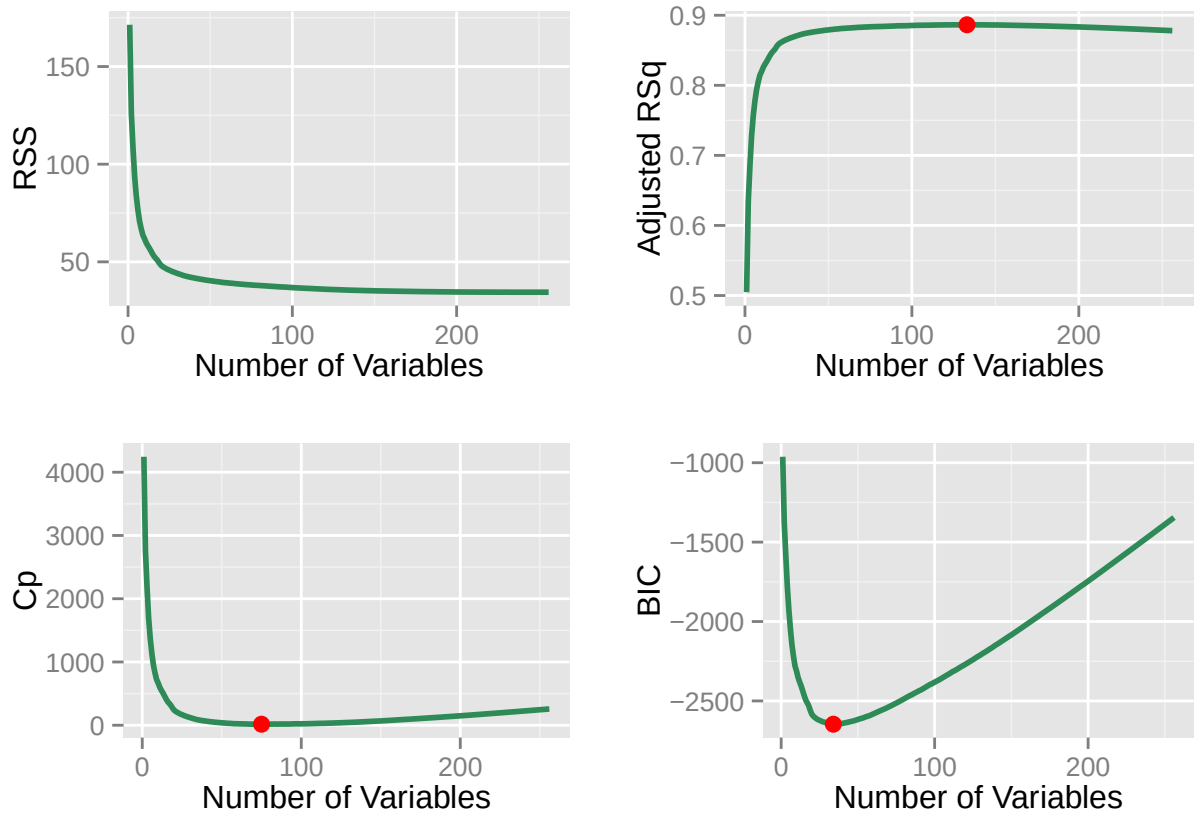
```
## (Intercept)      x.V104      x.V166      x.V249
##   0.4941865   0.1732184  -0.2354972   0.2287003
```

```
m.3 <- data.frame("variable"=c("Intercept",104,166,249),"best selection"=coef(reg.best3,3),"backward"=c
rownames(m.3) <- c()
```

Variable number 104,166,249 are retained when we consider models with 3 predictors. Comepare the results from parts (d),(f) and (g)

```
##    variable best.selection   backward  Variable    forward
## 1 Intercept      0.4941865  0.4941865 Intercept  0.4051336
## 2       104      0.1732184  0.1732184       165 -0.1744623
## 3       166     -0.2354972 -0.2354972       168 -0.2142737
## 4       249      0.2287003  0.2287003       249  0.2217201
```

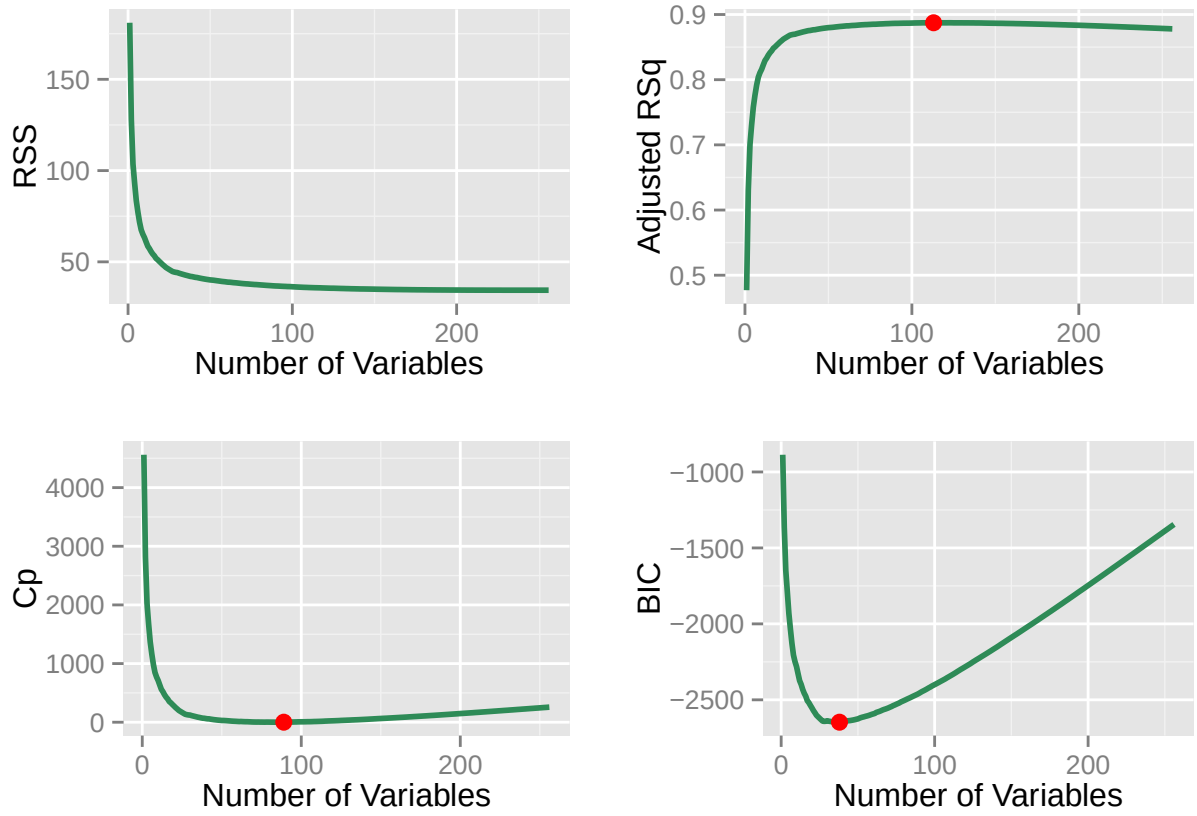
(h)



The following table is the number of variables that should be chosen according to these criteria

```
##                RSS adjr2 Cp BIC
## # of Variables 256   133 75  34
```

(i)



The following table is the number of variables that should be chosen according to these criteria

```
##                RSS adjr2 Cp BIC
## # of Variables 256   113 89  38
```

(j)

```
fwd.summary <- summary(reg.best5)
# Cp
fwd.cp.coef <- fwd.summary$which[75,]
fwd.cp.coef[fwd.cp.coef==TRUE]<- coef(reg.best5,75)
y.hat.train.cp <- cbind(1,x.train)%*%fwd.cp.coef
y.hat.test.cp <- cbind(1,x.test)%*%fwd.cp.coef
cp.train.error <- mean((y.hat.train.cp>.5)!=y.train)
cp.test.error <- mean((y.hat.test.cp>.5)!=y.test)
# BIC
fwd.bic.coef <- fwd.summary$which[34,]
fwd.bic.coef[fwd.bic.coef==TRUE]<- coef(reg.best5,34)
y.hat.train.bic <- cbind(1,x.train)%*%fwd.bic.coef
y.hat.test.bic <- cbind(1,x.test)%*%fwd.bic.coef
bic.train.error <- mean((y.hat.train.bic>.5)!=y.train)
bic.test.error <- mean((y.hat.test.bic>.5)!=y.test)
err.matrix1 <- data.frame(Cp=c(cp.train.error,cp.test.error),BIC=c(bic.train.error,bic.test.error))
rownames(err.matrix1) <- c("train error","test error")
```

The following table is the training and testing errors of the best models according to Cp and BIC from the forward selection

```
##                       Cp        BIC
## train error 0.007199424 0.01223902
## test error  0.038461538 0.04395604
```

(k)

```
bwd.summary <- summary(reg.best6)
# Cp
bwd.cp.coef <- bwd.summary$which[89,]
bwd.cp.coef[bwd.cp.coef==TRUE]<- coef(reg.best6,89)
y.hat.train.cp <- cbind(1,x.train)%*%bwd.cp.coef
y.hat.test.cp <- cbind(1,x.test)%*%bwd.cp.coef
cp.train.error <- mean((y.hat.train.cp>.5)!=y.train)
cp.test.error <- mean((y.hat.test.cp>.5)!=y.test)
# BIC
bwd.bic.coef <- bwd.summary$which[38,]
bwd.bic.coef[bwd.bic.coef==TRUE]<- coef(reg.best6,38)
y.hat.train.bic <- cbind(1,x.train)%*%bwd.bic.coef
y.hat.test.bic <- cbind(1,x.test)%*%bwd.bic.coef
bic.train.error <- mean((y.hat.train.bic>.5)!=y.train)
bic.test.error <- mean((y.hat.test.bic>.5)!=y.test)
err.matrix2 <- data.frame(Cp=c(cp.train.error,cp.test.error),BIC=c(bic.train.error,bic.test.error))
rownames(err.matrix2) <- c("train error","test error")
```

The following table is the training and testing errors of the best models according to Cp and BIC from the backward selection

```
##                       Cp        BIC
## train error 0.006479482 0.01295896
## test error  0.043956044 0.04670330
```

```
##             Cp(forward) BIC(forward) Cp(backward) BIC(backward)
## train error 0.007199424   0.01223902  0.006479482    0.01295896
## test error  0.038461538   0.04395604  0.043956044    0.04670330
```

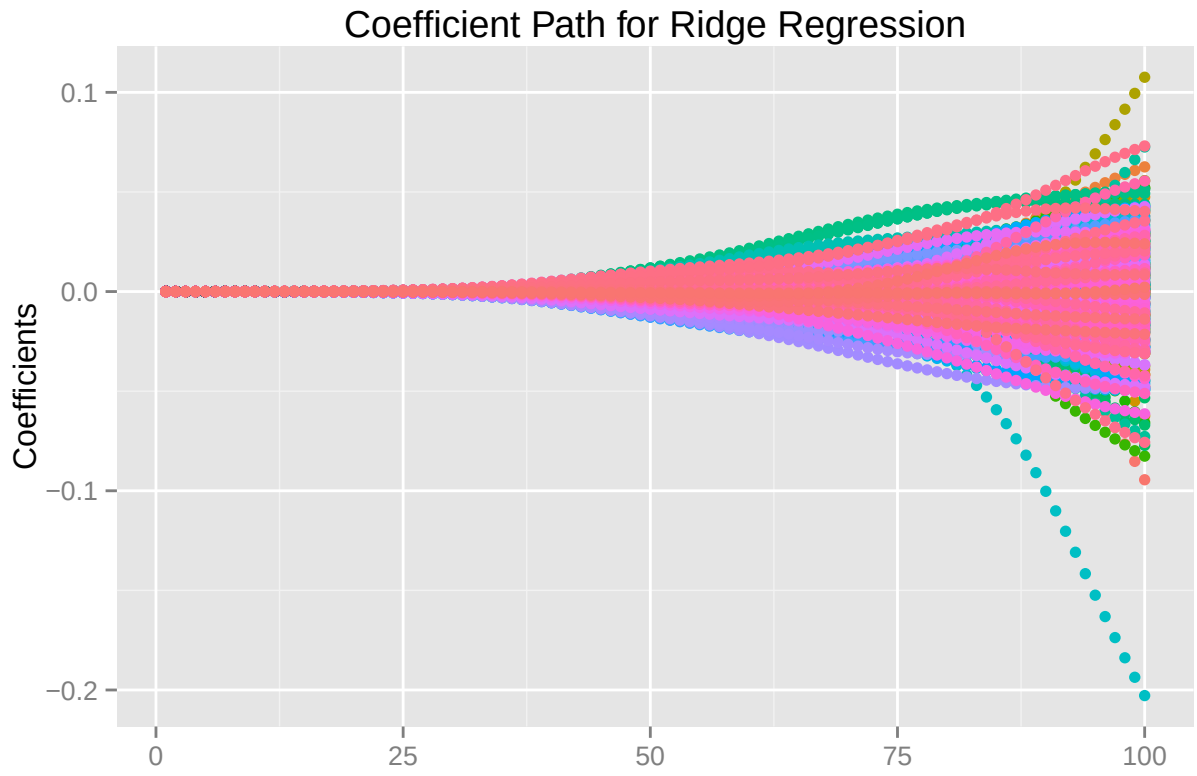
The model based on Cp from the backward selection has the smallest training error. And The model based on Cp from the forward selection has the smallest testing error. Hence, the model according to Cp from the forward selection is the best.

(l) This is because when standardizing the variables, the variables will be divided by their standard deviations. And ridge solutions are not equivariant under scaling of the inputs. One cannot obtain the regression coefficients that are exactly adjusted back to their original scale. Therefore,if variables are in the same units already, there is no need to standardize them.

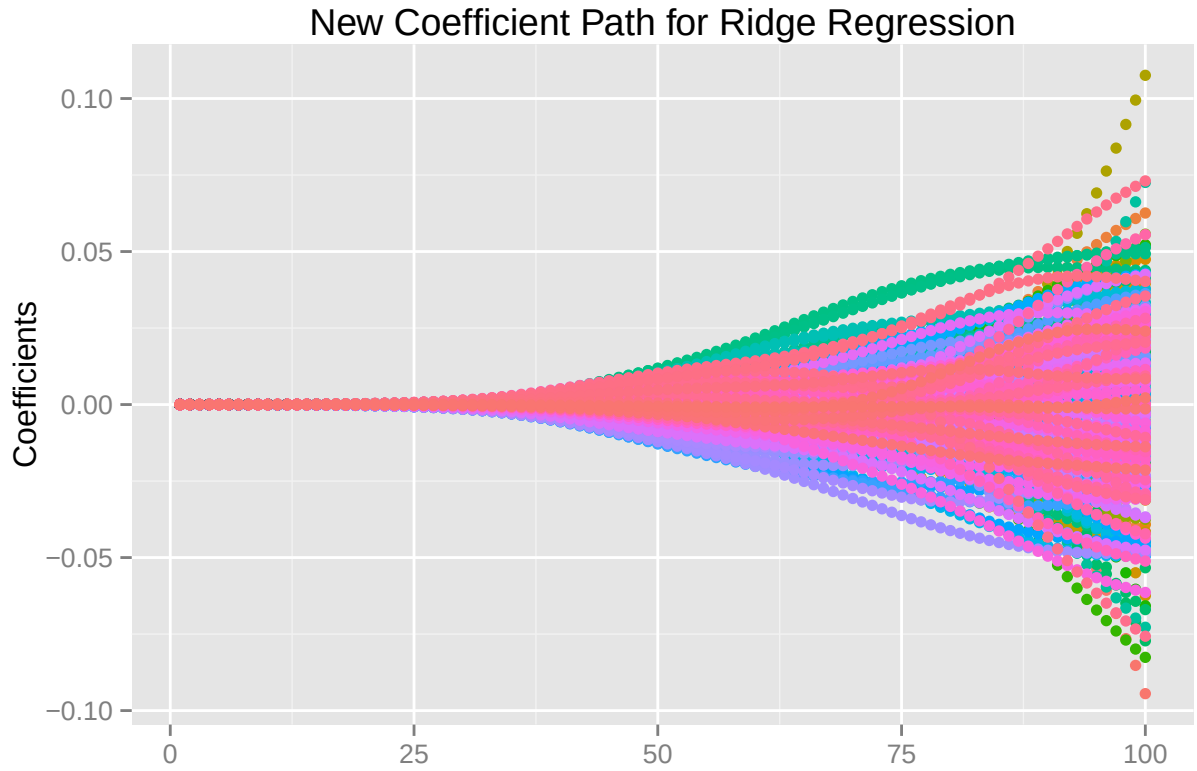
```
library(glmnet)
lambda.grid <- 10^seq(4,-3,length=100)
ridge.mod <- glmnet(x.train,y.train,alpha=0,lambda=lambda.grid,standardize=FALSE)
coef.matrix <- coef(ridge.mod) ; dim(coef.matrix)
```

```
## [1] 257 100
```

```
coef.matrix <- t(as.matrix(coef.matrix))
coef.matrix <- data.frame(lambda=1:100,coef.matrix)
coef.matrix.plot <- melt(coef.matrix[,-2],id="lambda")
g1 <- ggplot(coef.matrix.plot,aes(x=lambda,y=value,colour=variable))+
  geom_point()+
  scale_color_hue(guide=F)+
  labs(x="",y="Coefficients",title="Coefficient Path for Ridge Regression")
g1
```




```
# Replot the path
ind <- which.min(coef.matrix[100,-c(1,2)])
coef.matrix.plot.new <- melt(coef.matrix[,-c(2,ind+2)],id="lambda")
g2 <- ggplot(coef.matrix.plot.new,aes(x=lambda,y=value,colour=variable))+
  geom_point()+
  scale_color_hue(guide=F)+
  labs(x="",y="Coefficients",title="New Coefficient Path for Ridge Regression")
g2
```



(m)

```
pred.train.0 <- predict(ridge.mod,s=0,newx=x.train,exact=TRUE)
pred.test.0 <- predict(ridge.mod,s=0,newx=x.test,exact=TRUE)
# train error
ridge.train.error <- mean((pred.train.0>0.5)!=y.train);ridge.train.error
```

```
## [1] 0.005759539
```

```
# test error
ridge.test.error <- mean((pred.test.0>0.5)!=y.test);ridge.test.error
```

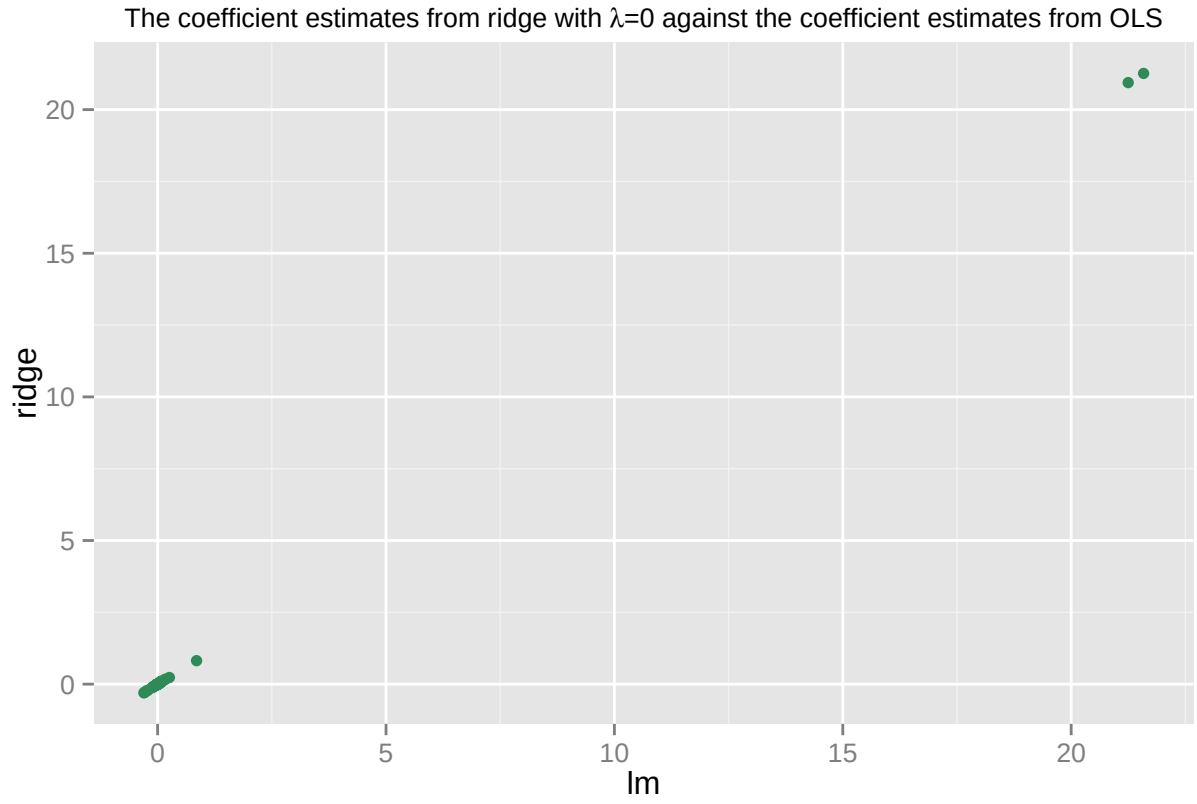
```
## [1] 0.04120879
```

The testing and training errors are 0.04120879 and 0.005759539, which are obtained by setting $\lambda = 0$ in ridge regression

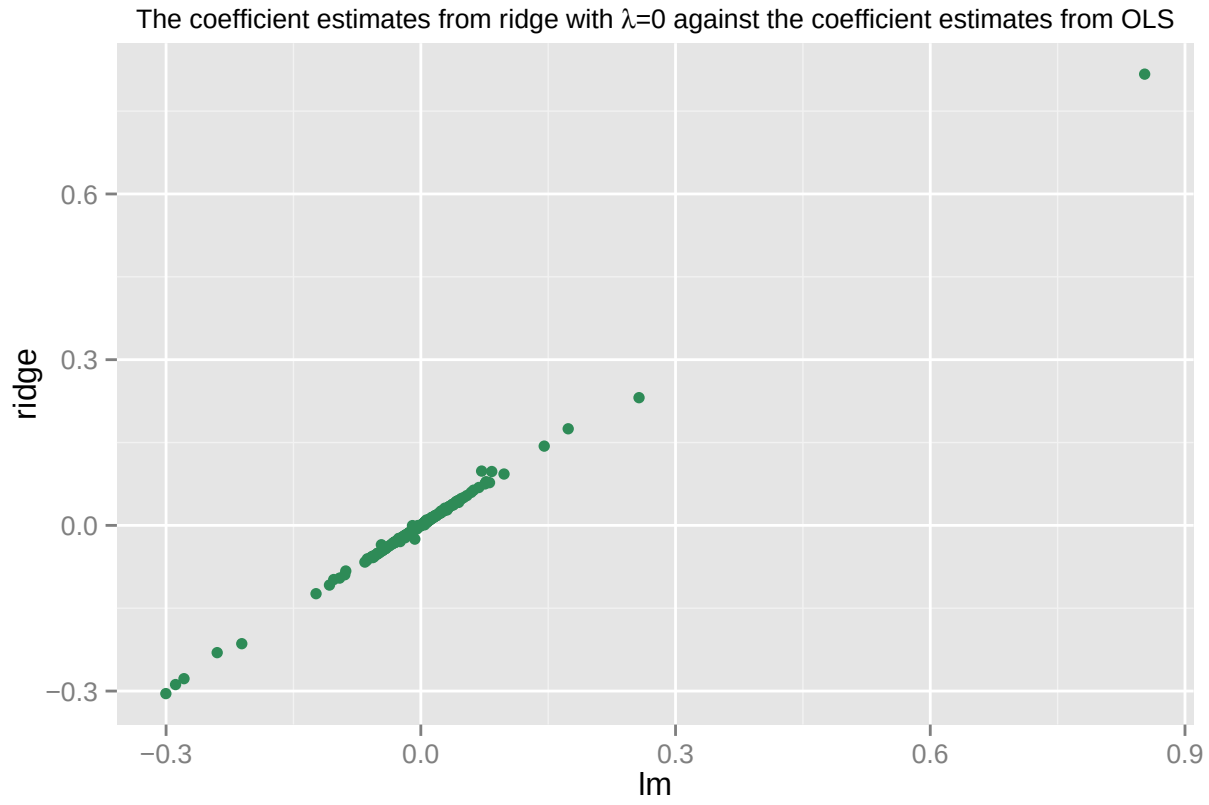
(n)

```
# setting lambda=0 and exact=T to obtain the coefficients
coef.0 <-  predict(ridge.mod,s=0,type="coefficients",exact=TRUE)
coef.ols.ridge <- data.frame(ridge=as.vector(coef.0),lm=coef(lm1))
g1 <- ggplot(coef.ols.ridge,aes(x=lm,y=ridge))+
```

```
  geom_point(colour="seagreen")+
  labs(title=expression(paste("The coefficient estimates from ridge with ",lambda
                              ,"=0 against the coefficient estimates from OLS")))+
  theme(plot.title = element_text(size=10))
g1
```




```
# Delete intercept and varianle number 16
g2 <- ggplot(coef.ols.ridge[-which(coef.ols.ridge[,1]>20),],aes(x=lm,y=ridge))+
  geom_point(colour="seagreen")+
  labs(title=expression(paste("The coefficient estimates from ridge with ",lambda
                              ,"=0 against the coefficient estimates from OLS")))+
  theme(plot.title = element_text(size=10))
g2
```



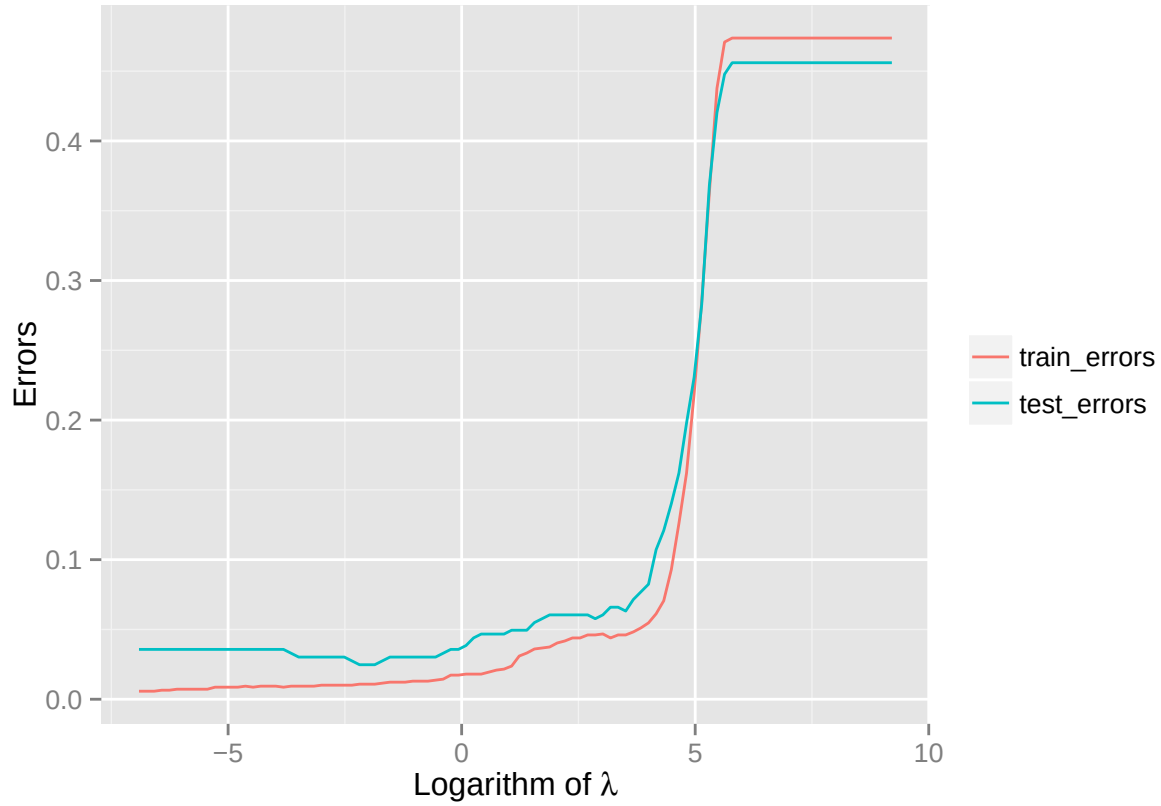

```
# Use linear regression to check if they are the same
ridge.ols.lm.mdl <- lm(ridge~lm,coef.ols.ridge)
coef(ridge.ols.lm.mdl)
```

```
##   (Intercept)            lm
## -6.412676e-05  9.851975e-01
```

From the plot and the regression coefficient, the two coefficient estimates are almost the same. The tiny difference is due to different fitting methods in these two regression methods.

(o)

```
# Compute the train and test errors for ridge regression for all choiches of lambda
# on the grid
y.hat.train.grid <- cbind(1,x.train)%*%as.matrix(coef(ridge.mod))
y.hat.test.grid <- cbind(1,x.test)%*%as.matrix(coef(ridge.mod))
train.error.grid <- colMeans((y.hat.train.grid>.5)!=y.train)
test.error.grid <- colMeans((y.hat.test.grid>.5)!=y.test)
# Plot these errors against the logarithm of lambda
errors.grid <- data.frame(lambda=log(lambda.grid),train_errors=train.error.grid
                          ,test_errors=test.error.grid)
p <- ggplot(melt(errors.grid,id="lambda"),aes(x=lambda,y=value,colour=variable))+
  geom_line()+
  labs(x=expression(paste("Logarithm of ",lambda)),y="Errors")+
  scale_color_hue(name="")
p
```




```
lambda.grid[which.min(train.error.grid)]
```

```
## [1] 0.001384886
```

```
lambda.grid[which.min(test.error.grid)]
```

```
## [1] 0.1555676
```

```
# Test and train errors for the current dataset
ridge.best.train.error <- train.error.grid[which.min(test.error.grid)]
ridge.best.train.error
```

```
##        s68
## 0.01079914
```

```
ridge.best.test.error<-test.error.grid[which.min(test.error.grid)]
ridge.best.test.error
```

```
##        s68
## 0.02472527
```

```
ridge.best <- c(ridge.best.train.error,ridge.best.test.error)
```

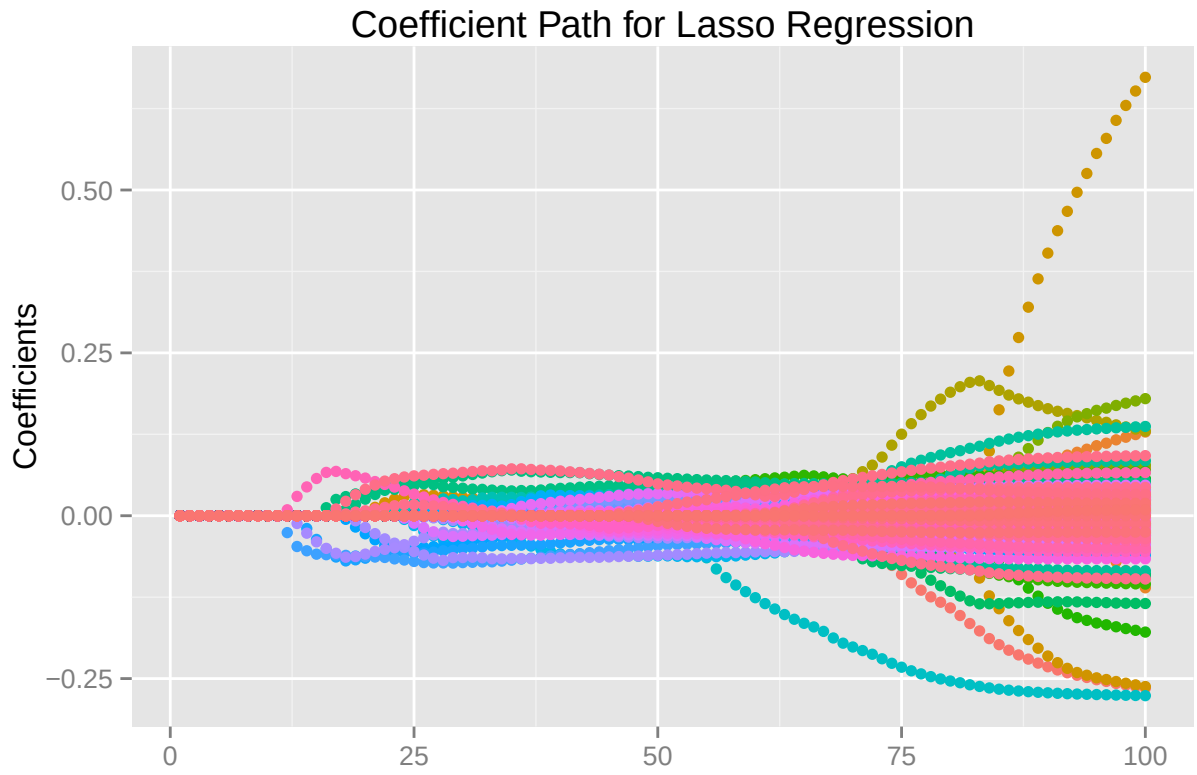
The lambda value that gives the smallest test error is 0.1555676. The lambda value that gives the smallest train error is 0.001384886. We should use $\lambda = 0.1555676$ for classification on a future observation. When using this λ, the test error is 0.02472527, the train error 0.01079914 for the current dataset. The chance for misclassification on a future observation is 2.472527%

(p)

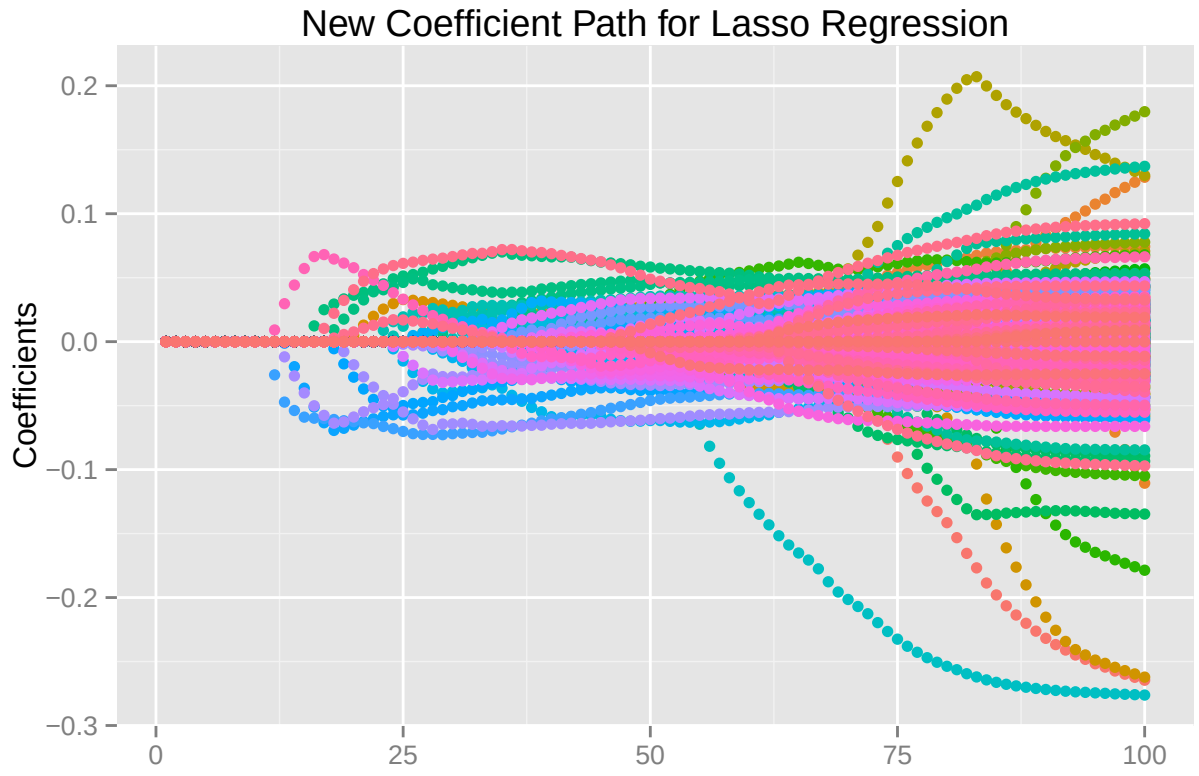
```
n.lambda <- 100
lambda.grid <- 10^seq(0,-5,length=n.lambda)
lasso.mod <- glmnet(x.train,y.train,alpha=1,lambda=lambda.grid,standardize=FALSE)
coef.matrix <- coef(lasso.mod) ; dim(coef.matrix)
```

```
## [1] 257 100
```

```
coef.matrix <- t(as.matrix(coef.matrix))
coef.matrix <- data.frame(lambda=1:100,coef.matrix)
coef.matrix.plot <- melt(coef.matrix[,-2],id="lambda")
g1 <- ggplot(coef.matrix.plot,aes(x=lambda,y=value,colour=variable))+
  geom_point()+
  scale_color_hue(guide=F)+
  labs(x="",y="Coefficients",title="Coefficient Path for Lasso Regression")
g1
```



```
# Replot the path
ind <- which.max(coef.matrix[100,-c(1,2)])
coef.matrix.plot.new <- melt(coef.matrix[,-c(2,ind+2)],id="lambda")
g2 <- ggplot(coef.matrix.plot.new,aes(x=lambda,y=value,colour=variable))+
  geom_point()+
  scale_color_hue(guide=F)+
  labs(x="",y="Coefficients",title="New Coefficient Path for Lasso Regression")
g2
```



(q)

```
pred.train.0 <- predict(lasso.mod,s=0,newx=x.train,exact=TRUE)
pred.test.0 <- predict(lasso.mod,s=0,newx=x.test,exact=TRUE)
# train error
lasso.train.error <- mean((pred.train.0>0.5)!=y.train);lasso.train.error
```

```
## [1] 0.005759539
```

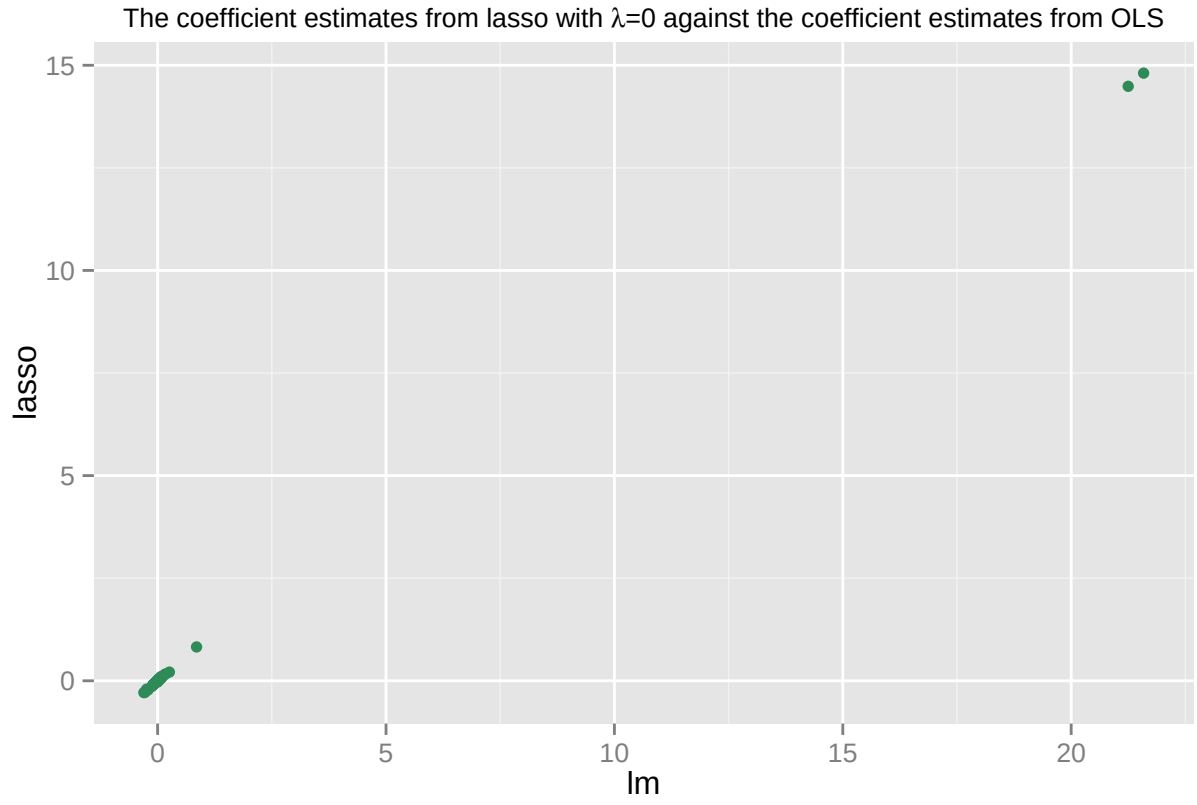
```
# test error
lasso.test.error <- mean((pred.test.0>0.5)!=y.test);lasso.test.error
```

```
## [1] 0.03846154
```

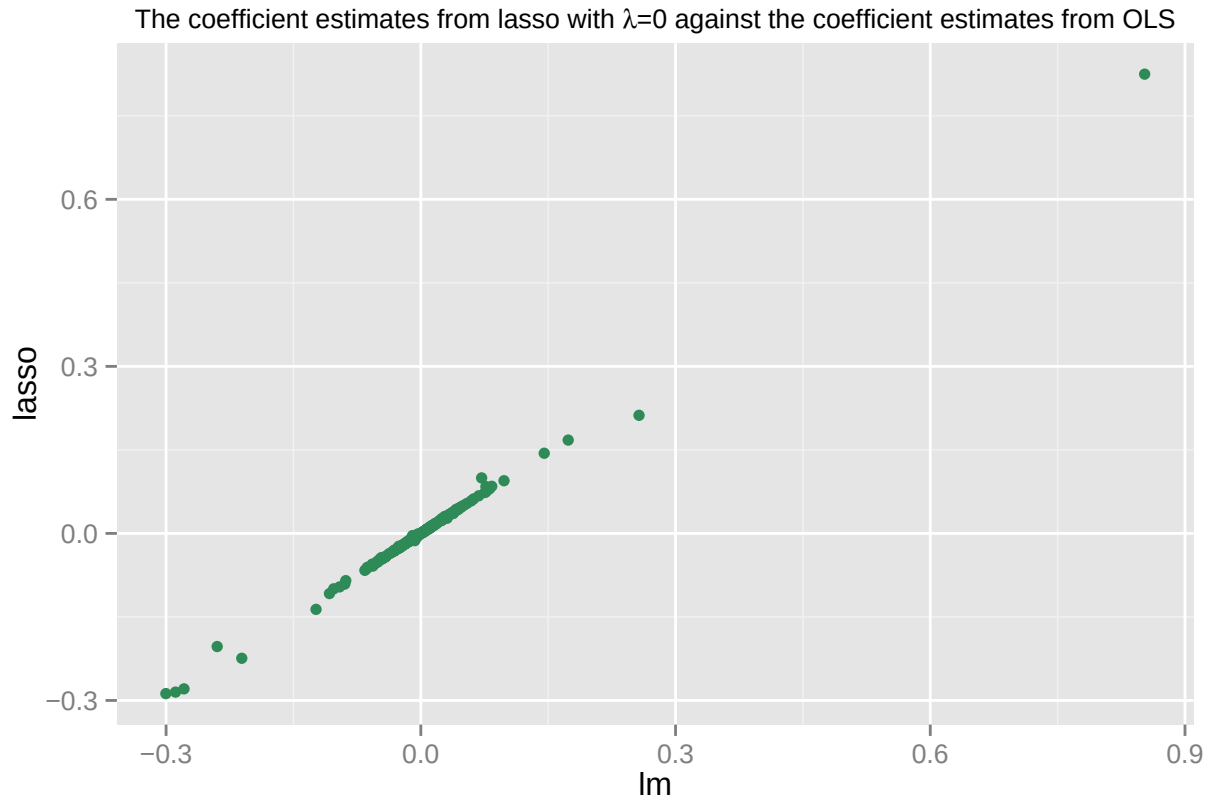
The testing and training errors are 0.03846154 and 0.005759539, which are obtained by setting $\lambda = 0$ in lasso regression.

(r)

```
# setting lambda=0 and exact=T to obtain the coefficients
coef.0 <-  predict(lasso.mod,s=0,type="coefficients",exact=TRUE)
coef.ols.lasso <- data.frame(lasso=as.vector(coef.0),lm=coef(lm1))
g1 <- ggplot(coef.ols.lasso,aes(x=lm,y=lasso))+
  geom_point(colour="seagreen")+
  labs(title=expression(paste("The coefficient estimates from lasso with ",lambda
                              ,"=0 against the coefficient estimates from OLS")))+
  theme(plot.title = element_text(size=10))
g1
```




```
# Delete intercept and varianle number 16
g2 <- ggplot(coef.ols.lasso[-which(coef.ols.lasso[,1]>10),],aes(x=lm,y=lasso))+
  geom_point(colour="seagreen")+
  labs(title=expression(paste("The coefficient estimates from lasso with ",lambda
                              ,"=0 against the coefficient estimates from OLS")))+
  theme(plot.title = element_text(size=10))
g2
```



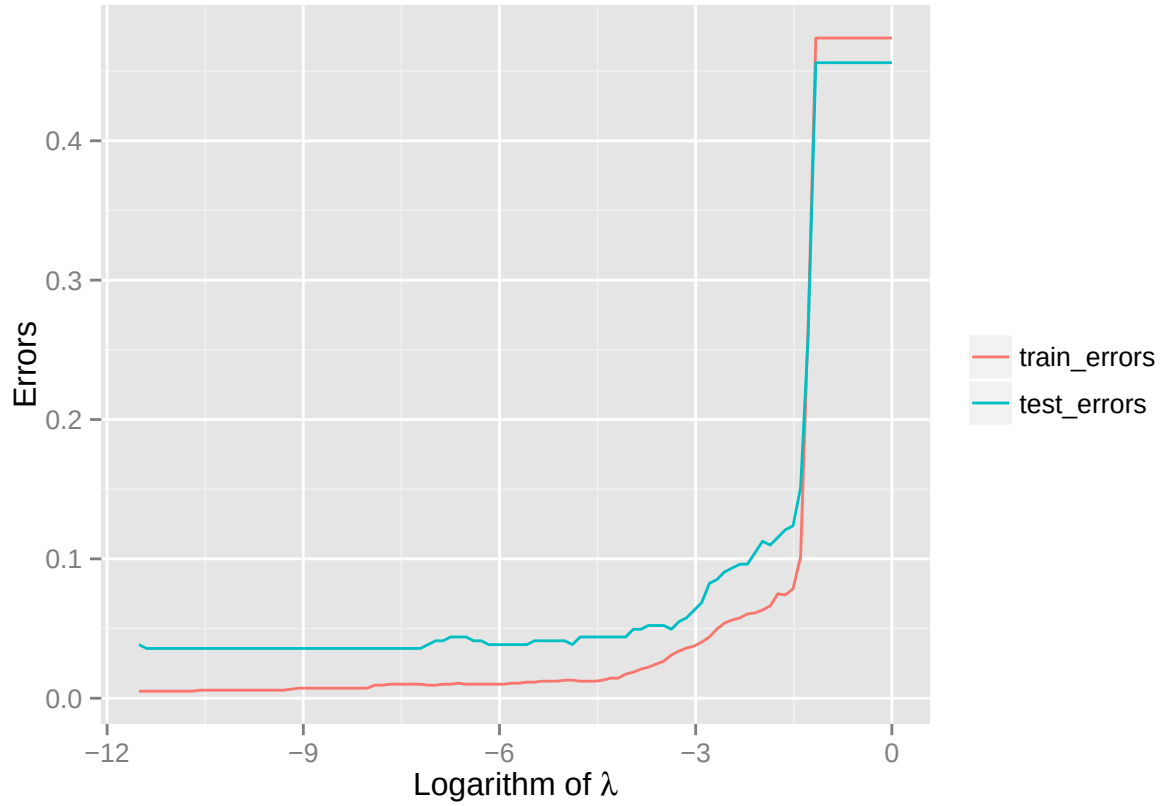

```
# Use linear regression to check if they are the same
lasso.ols.lm.mdl <- lm(lasso~lm,coef.ols.lasso)
coef(lasso.ols.lm.mdl)
```

```
##   (Intercept)            lm
## -0.0001293732  0.6844011733
```

In the plot, it looks like a linear relationship. However, the regression coefficient is much smaller than 1; therefore, they are not the same. This may be caused by the different fitting methods in `lm()` and `glmnet()`.

(s)

```
# Compute the train and test errors for lasso regression for all choiches of lambda
# on the grid
y.hat.train.grid <- cbind(1,x.train)%*%as.matrix(coef(lasso.mod))
y.hat.test.grid <- cbind(1,x.test)%*%as.matrix(coef(lasso.mod))
train.error.grid <- colMeans((y.hat.train.grid>.5)!=y.train)
test.error.grid <- colMeans((y.hat.test.grid>.5)!=y.test)
# Plot these errors against the logarithm of lambda
errors.grid <- data.frame(lambda=log(lambda.grid),train_errors=train.error.grid
                          ,test_errors=test.error.grid)
p <- ggplot(melt(errors.grid,id="lambda"),aes(x=lambda,y=value,colour=variable))+
  geom_line()+
  labs(x=expression(paste("Logarithm of ",lambda)),y="Errors")+
  scale_color_hue(name="")
p
```




```
lambda.grid[which.min(train.error.grid)]
```

```
## [1] 2.25702e-05
```

```
lambda.grid[which.min(test.error.grid)]
```

```
## [1] 0.0007390722
```

```
# Test and train errors for the current dataset
lasso.best.train.error <- train.error.grid[which.min(test.error.grid)]
lasso.best.train.error
```

```
##         s62
## 0.01007919
```

```
lasso.best.test.error <- test.error.grid[which.min(test.error.grid)]
lasso.best.test.error
```

```
##         s62
## 0.03571429
```

```
lasso.best <- c(lasso.best.train.error,lasso.best.test.error)
```

The lambda value that gives the smallest test error is 0.0007390722. The lambda value that gives the smallest train error is 2.25702e-05. We should use $\lambda = 0.0007390722$ for classification on a future observation. When using this λ, the test error is 0.03571429, the train error 0.01007919 for the current dataset. The chance for misclassification on a future observation is 3.571429%

(t)

```
summary <- data.frame("train error"=c(lm.train.error,knn.train.error),
                      "test error"=c(lm.test.error,knn.test.error))
colnames(summary) <- c("train error","test error")
summary.all <- rbind(summary,t(err.matrix1),t(err.matrix2),ridge.best,lasso.best)
rownames(summary.all) <- c("OLS","KNN(k=1)","KNN(k=3)","KNN(k=5)","KNN(k=7)","KNN(k=15)"
                           ,"Forward(Cp)","Forward(BIC)","Backward(Cp)","Backward(BIC)"
                           ,"Ridge","Lasso")
summary.all
```

```
##               train error test error
## OLS           0.005759539 0.04120879
## KNN(k=1)      0.000000000 0.02472527
## KNN(k=3)      0.005039597 0.03021978
## KNN(k=5)      0.005759539 0.03021978
## KNN(k=7)      0.006479482 0.03296703
## KNN(k=15)     0.009359251 0.03846154
## Forward(Cp)   0.007199424 0.03846154
## Forward(BIC)  0.012239021 0.04395604
## Backward(Cp)  0.006479482 0.04395604
## Backward(BIC) 0.012958963 0.04670330
## Ridge         0.010799136 0.02472527
## Lasso         0.010079194 0.03571429
```

Although the test errors of K-NN(k=1) and Ridge regression are both smaller than others, for k-nearest neighbors, when k=1, the model is too complex, and the predictions will have large variance. It may not be good for predicting. Hence, the Ridge regression could be the best method in this case.