Chen_DataMining_HW1

Xuejun Chen

2018/2/10

Load packages

```
library(tidyverse)
```

```
## -- Attaching packages ----------------------------------------------------- tidyverse 1.2.1 --
```

```
## <U+221A> ggplot2 2.2.1     <U+221A> purrr   0.2.4
## <U+221A> tibble  1.4.2     <U+221A> dplyr   0.7.4
## <U+221A> tidyr   0.7.2     <U+221A> stringr 1.2.0
## <U+221A> readr   1.1.1     <U+221A> forcats 0.2.0
```

```
## -- Conflicts -------------------------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(magrittr)
```

```
##
## Attaching package: 'magrittr'
```

```
## The following object is masked from 'package:purrr':
##
##     set_names
```

```
## The following object is masked from 'package:tidyr':
##
##     extract
```

```
library(class)  # KNN
library(leaps)  # function regsubsets
library(glmnet) # ridge/lasso regression
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following object is masked from 'package:tidyr':
##
##     expand
```

```
## Loading required package: foreach
```

```
##
## Attaching package: 'foreach'
```

```
## The following objects are masked from 'package:purrr':
##
##     accumulate, when
```

```
## Loaded glmnet 2.0-13
```

a)

```
# Load data
load("/Users/AudreyChen/Desktop/Data\ Mining/hw1.RData")

# Check the dimensions
dim(train2)
```

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

```
dim(train3)
```

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

```
dim(test2)
```

```
## [1] 198 256
```

```
dim(test3)
```

```
## [1] 166 256
```

```
# Create training and testing data sets
xx.train <- rbind(train2,train3)
xx.test <- rbind(test2,test3)

# create y's as binary outcomes
# if 2, false. if 3, true
y.train <- rep(FALSE, 1389)
y.train[732:1389] <- TRUE
y.test <- rep(FALSE,364)
y.test[199:364] <- TRUE

# OLS
reg <- lm(y.train ~ xx.train)

# Training and testing errors
pred.train <- predict(reg, as.data.frame(xx.train)) > .5
pred.test <- (cbind(1, xx.test) %*% reg$coef) > .5
reg.test.err <- mean(pred.test != y.test)
reg.train.err <- mean(pred.train != y.train)

# Classification by k-nearest neighbors
k <- c(1, 3, 5, 7, 15)
k.test.err <- rep(NA, length(k))
k.train.err <- rep(NA, length(k))

for (i in 1:length(k)) {
  k.pred.test <- knn(train = xx.train,test = xx.test, y.train, k[i], prob=FALSE)
  k.test.err[i] <- mean(k.pred.test != y.test)

  k.pred.train <- knn(train = xx.train,test = xx.train, y.train, k[i], prob=FALSE)
  k.train.err[i] <- mean(k.pred.train != y.train)
}

# Compare
```

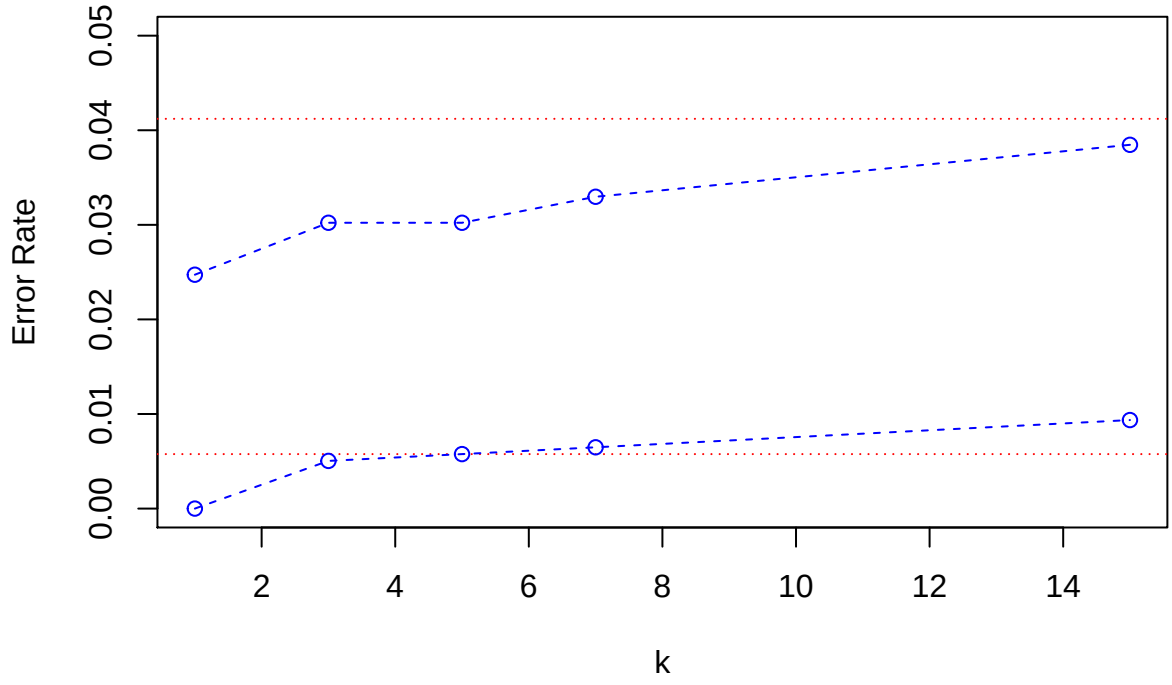
```r
data.frame(k,k.test.err, k.train.err)
```

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

```r
data.frame(reg.test.err,reg.train.err)
```

```
##   reg.test.err reg.train.err
## 1   0.04120879   0.005759539
```

```r
# Plot the errors
plot(c(1, 15), c(0, 0.05), type = "n", ylab = "Error Rate", xlab = "k")
abline(h = reg.test.err, col = 2, lty = 3)
abline(h = reg.train.err, col = 2, lty = 3)
points(k, k.test.err, col = 4)
lines(k, k.test.err, col = 4, lty = 2)
points(k, k.train.err, col = 4)
lines(k, k.train.err, col = 4, lty = 2)
```



b)

We cannot obtain the best subset selection result for all k if we do not manipulate the arguments of the function regsubsets(). Because regsubsets() only reports results up to the best eight-variable model by default and there is a k assigned as 15.

c)

```
dat <- as.data.frame(cbind(y.train,xx.train))
# regfit.full <- regsubsets(y.train ~ ., really.big=TRUE,data=dat)
# Cannot obtain any result.
```

d)

```
regfit.full <- regsubsets(y.train~., dat, nvmax=3, really.big=TRUE)

# nvmax changes the default of returning 8 best variables to 3 best variables.
reg.summary <- summary(regfit.full)
which(reg.summary$outmat[3,]=="*")
```

```
## V104 V166 V249
##  104  166  249
```

```
# Colomn 104, 166, 249 are 3 variables for the best model.
```

e)

```
# Need to add method="forward" to run forward selection
# Default: subsets of each size up to 8
regfit.fwd <- regsubsets(y.train~., dat, method="forward")

# Models with 9 predictors
regfit.fwd.9 <- regsubsets(y.train~., dat, nvmax=9, method="forward")
reg9.summary <- summary(regfit.fwd.9)
which(reg9.summary$outmat[9,]=="*")
```

```
## V104 V122 V124 V148 V165 V168 V183 V200 V249
##  104  122  124  148  165  168  183  200  249
```

```
# Colomn 104, 122, 124, 148, 165, 168, 183, 200, 249 are 9 predictors for the best model.
```

f)

```
regfit.fwd.256 <- regsubsets(y.train~., dat, nvmax=256, method="forward")
reg256.summary <- summary(regfit.fwd.256)
which(reg256.summary$outmat[9,]=="*")
```

```
## V104 V122 V124 V148 V165 V168 V183 V200 V249
##  104  122  124  148  165  168  183  200  249
```

```
# Colomns 104  122  124  148  165  168  183  200  249 are the 9 predictors.
which(reg256.summary$outmat[3,]=="*")
```

```
## V165 V168 V249
##  165  168  249
```

```
# Colomns 165, 168, 249 are retained.
```

g)

```
regfit.bwd.256 <- regsubsets(y.train~., dat, nvmax=256, method="backward")
reg.bwd256.summary <- summary(regfit.bwd.256)
which(reg.bwd256.summary$outmat[3,]=="*")
```

```
## V104 V166 V249
##  104  166  249
```

```
# 104  166  249
```

Part d and part g give the same result. The only common predictor using forward selection is colomn 249.

h)

Plot RSS, adjusted R2, Cp, and BIC for all of the models.

```
par(mfrow = c(2,2))

plot(reg256.summary$rss, xlab = "Number of Variables", ylab = "RSS", type = "l")
plot(reg256.summary$adjr2, xlab = "Number of Variables", ylab = "Adjusted RSq", type = "l")
which.max(reg256.summary$adjr2)
```

```
## [1] 133
```

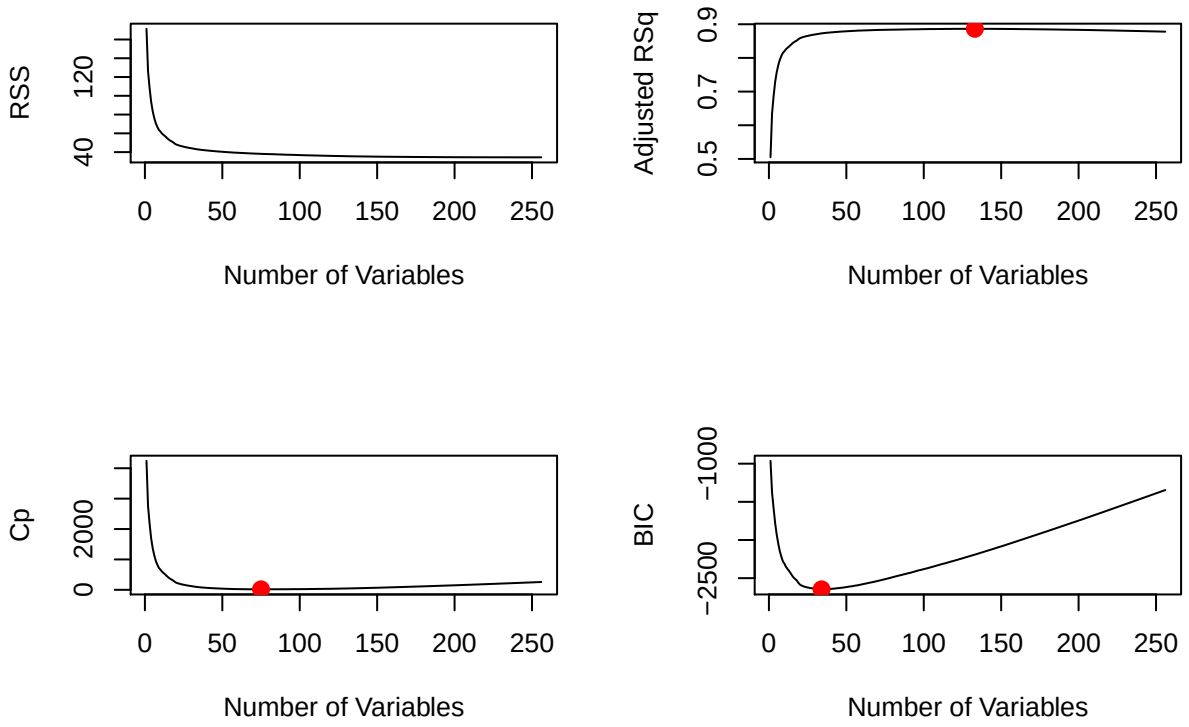
```
points(which.max(reg256.summary$adjr2),
       reg256.summary$adjr2[which.max(reg256.summary$adjr2)], col = "red", cex = 2, pch = 20)
plot(reg256.summary$cp, xlab = "Number of Variables", ylab = "Cp", type = 'l')
which.min(reg256.summary$cp)
```

```
## [1] 75
```

```
points(which.min(reg256.summary$cp),
       reg256.summary$cp[which.min(reg256.summary$cp)], col = "red", cex = 2, pch = 20)
plot(reg256.summary$bic, xlab = "Number of Variables", ylab = "BIC", type = 'l')
which.min(reg256.summary$bic)
```

```
## [1] 34
```

```
points(which.min(reg256.summary$bic),
       reg256.summary$bic[which.min(reg256.summary$bic)], col = "red", cex = 2, pch = 20)
```



i)

```
par(mfrow = c(2,2))
plot(reg.bwd256.summary$rss,xlab="Number of Variables",ylab="RSS",type = "l")
plot(reg.bwd256.summary$adjr2,xlab="Number of Variables",ylab="Adjusted RSq",type="l")
which.max(reg.bwd256.summary$adjr2)
```

```
## [1] 113
```

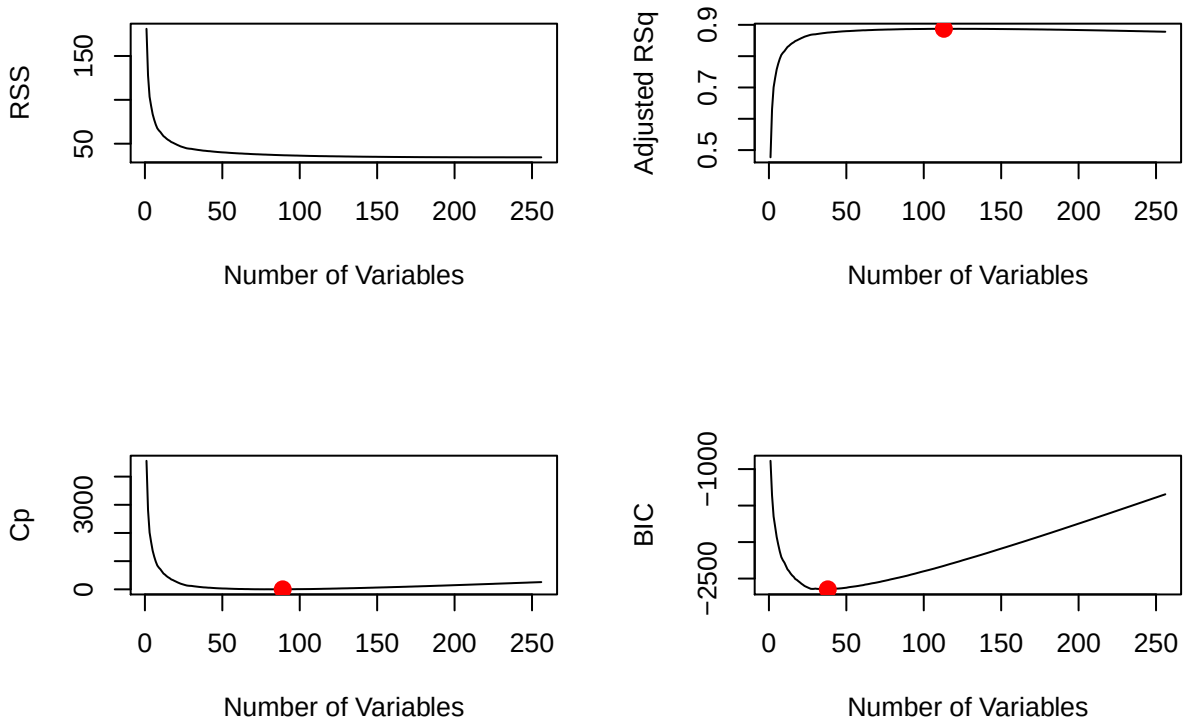
```
points(which.max(reg.bwd256.summary$adjr2),
       reg.bwd256.summary$adjr2[which.max(reg.bwd256.summary$adjr2)], col="red",cex=2,pch=20)
plot(reg.bwd256.summary$cp,xlab="Number of Variables",ylab="Cp",type='l')
which.min(reg.bwd256.summary$cp)
```

```
## [1] 89
```

```
points(which.min(reg.bwd256.summary$cp),
       reg.bwd256.summary$cp[which.min(reg.bwd256.summary$cp)],col="red",cex=2,pch=20)
plot(reg.bwd256.summary$bic,xlab="Number of Variables",ylab="BIC",type='l')
which.min(reg.bwd256.summary$bic)
```

```
## [1] 38
```

```
points(which.min(reg.bwd256.summary$bic),
       reg.bwd256.summary$bic[which.min(reg.bwd256.summary$bic)],col="red",cex=2,pch=20)
```



j)

```
# CP
coef.cp <- coefficients(regfit.fwd.256, id = which.min(reg256.summary$cp))
yhat.cp <- (cbind(1, xx.train[,names(coef.cp[-1])]) %*% coef.cp) > .5
fwd.cp.train.err <- mean(yhat.cp != y.train)

yhat.cp2 <- (cbind(1, xx.test[,names(coef.cp[-1])]) %*% coef.cp) > .5
fwd.cp.test.err <- mean(yhat.cp2 != y.test)

# BIC
coef.bic <- coefficients(regfit.fwd.256, id = which.min(reg256.summary$bic))
yhat.bic <- (cbind(1, xx.train[,names(coef.bic[-1])]) %*% coef.bic ) > .5
fwd.bic.train.err <- mean(yhat.bic != y.train)

yhat.bic2 <- (cbind(1, xx.test[,names(coef.bic[-1])]) %*% coef.bic) > .5
fwd.bic.test.err <- mean(yhat.bic2 != y.test)

data.frame(fwd.cp.train.err,fwd.cp.test.err,fwd.bic.train.err,fwd.bic.test.err)
```

```
##   fwd.cp.train.err fwd.cp.test.err fwd.bic.train.err fwd.bic.test.err
## 1      0.007199424      0.03846154        0.01223902       0.04395604
```

k)

```
# CP
coef.cp <- coefficients(regfit.bwd.256, id = which.min(reg.bwd256.summary$cp))
yhat.cp <- (cbind(1, xx.train[,names(coef.cp[-1])]) %*% coef.cp) > .5
bwd.cp.train.err <- mean(yhat.cp != y.train)

yhat.cp2 <- (cbind(1, xx.test[,names(coef.cp[-1])]) %*% coef.cp) > .5
bwd.cp.test.err <- mean(yhat.cp2 != y.test)

# BIC
coef.bic <- coefficients(regfit.bwd.256, id = which.min(reg.bwd256.summary$bic))
yhat.bic <- (cbind(1, xx.train[,names(coef.bic[-1])]) %*% coef.bic) > .5
bwd.bic.train.err <- mean(yhat.bic != y.train)

yhat.bic2 <- (cbind(1, xx.test[,names(coef.bic[-1])]) %*% coef.bic) > .5
bwd.bic.test.err <- mean(yhat.bic2 != y.test)

data.frame(bwd.cp.train.err,bwd.cp.test.err,bwd.bic.train.err,bwd.bic.test.err)
```

```
##   bwd.cp.train.err bwd.cp.test.err bwd.bic.train.err bwd.bic.test.err
## 1      0.006479482      0.04395604        0.01295896        0.0467033
```

The best model according to Cp from backward selection has the smallest training error, 0.006479482. The best model according to Cp from forward selection has the smallest test error, 0.03846154. The best one is the model according to Cp from forward selection because it has the smallest test error and its training error is small enough as well.

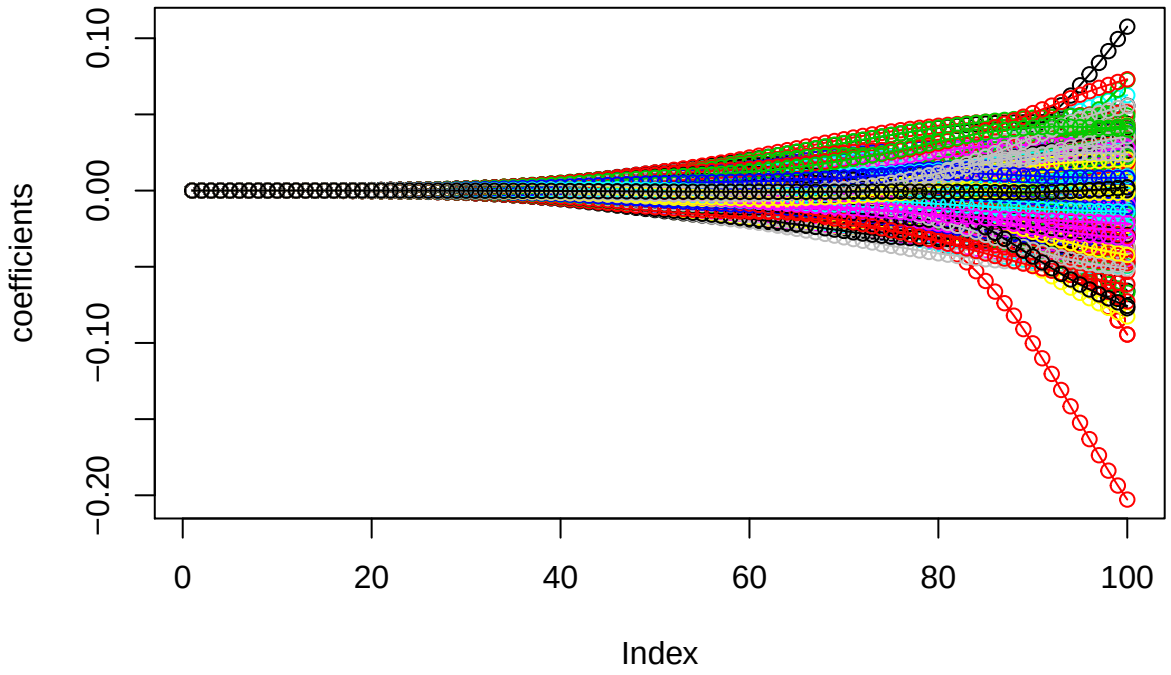
l)

```
lambda.grid <- 10^seq(4,-3,length=100)
# No need to standardize because the variables are on the same scale already.

ridge.mod <- glmnet(xx.train, y.train, alpha=0, lambda=lambda.grid, standardize = FALSE)
coeff.matrix <- coef(ridge.mod)
dim(coeff.matrix)
```

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

```
par(mfrow=c(1,1))
plot(coeff.matrix[2,],ylim=c(min(coeff.matrix[-1,]),max(coeff.matrix[-1,])),col=2,type="o",ylab="coeffi
for(i in 2:257){
  lines(coeff.matrix[i,],col=i,type="o")
}
```

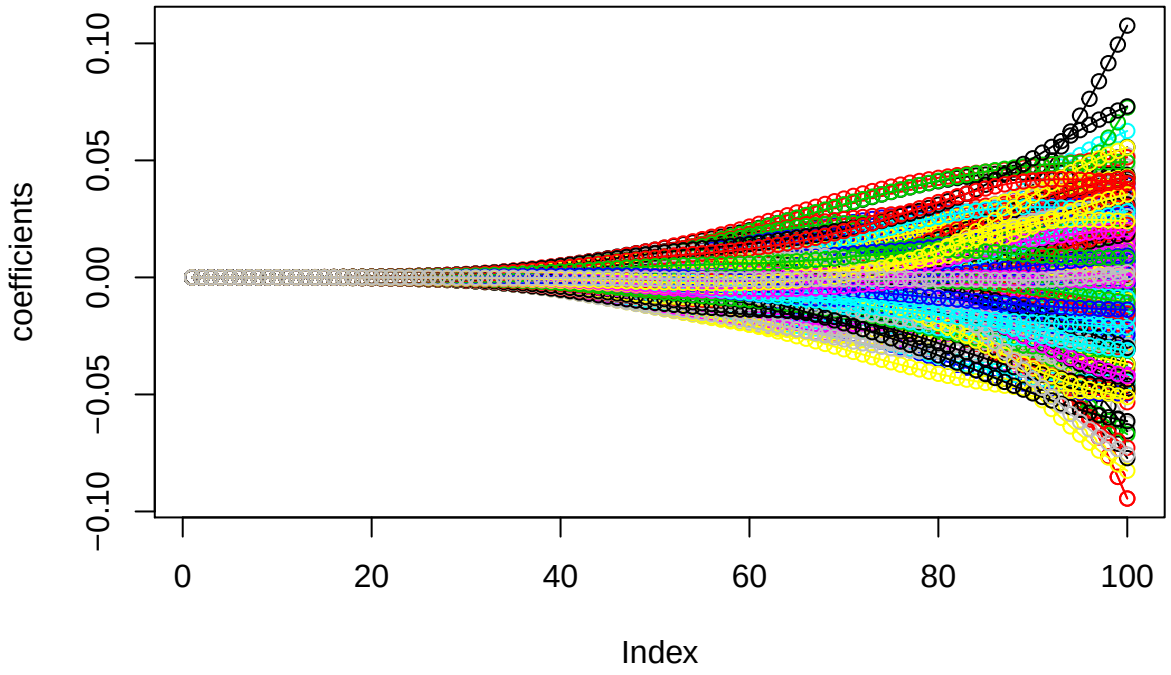



```
which.min(coeff.matrix[,100])
```

```
## V129
##  130
```

```
# remove colomn 130 from the coefficient matrix
coeff.matrix <- coeff.matrix[-130, ]

plot(coeff.matrix[2,],ylim=c(min(coeff.matrix[-1,]),max(coeff.matrix[-1,])),col=2,type="o",ylab="coeffi
for(i in 2:256){
  lines(coeff.matrix[i,],col=i,type="o")
}
```



m)

```
# testing err
ridge.pred.test <- predict(ridge.mod,s=0,newx=xx.test,exact =TRUE,x=xx.train,y=y.train)
mean((ridge.pred.test-y.test)^2)
```

```
## [1] 0.1487813
```

```
# training err
ridge.pred.train <- predict(ridge.mod,s=0,newx=xx.train,exact =TRUE,x=xx.train,y=y.train)
mean((ridge.pred.train-y.train)^2)
```

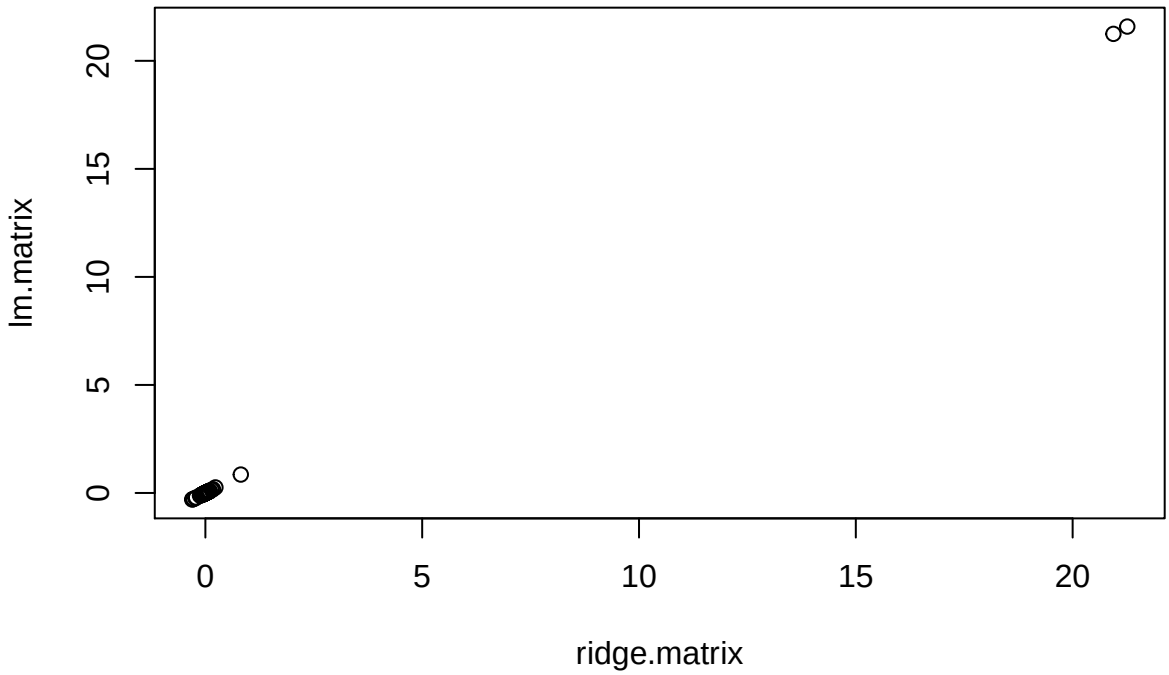
```
## [1] 0.02481558
```

n)

```
# ridge with s=0
ridge.reg <- predict(ridge.mod,s=0, exact = TRUE,type="coefficients",x=xx.train,y=y.train)
# lm
lm.reg <- lm(y.train~xx.train)

lm.matrix <- as.matrix(coef(lm.reg))
ridge.matrix <- as.matrix(ridge.reg)

plot(ridge.matrix, lm.matrix)
```




```
which(ridge.matrix == lm.matrix) == TRUE
```

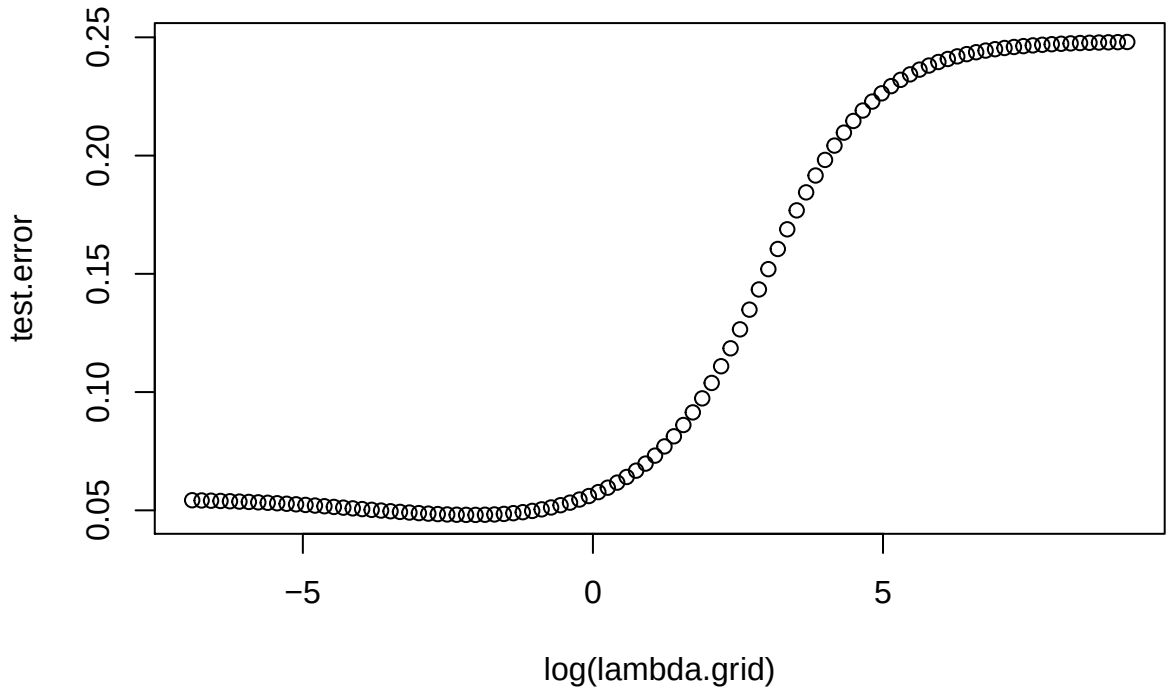
```
## logical(0)
```

```r
# not the same
```

o)

```r
# testing error
test.error <- rep(NA, 100)
for(i in 1:100){
  ridge.pred <- predict(ridge.mod,s=lambda.grid[i],newx=xx.test)
  test.error[i] <- mean((ridge.pred-y.test)^2)
}

plot(log(lambda.grid),test.error)
```



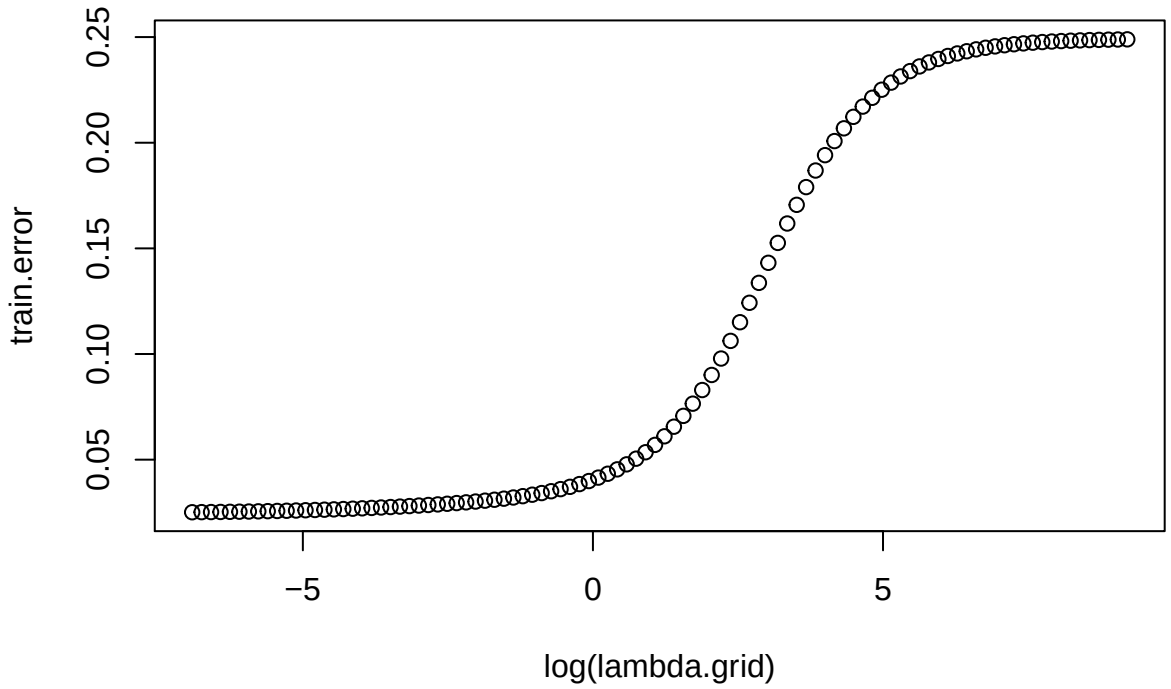

```r
# lambda value that gives the smallest test error
lambda.grid[which.min(test.error)]
```

```
## [1] 0.1321941
```

```r
# save the smallest test error of ridge regression
ridge.test.error = test.error[which.min(test.error)]

# training error
train.error <- rep(NA, 100)
for(i in 1:100){
  ridge.pred2 <- predict(ridge.mod,s=lambda.grid[i],newx=xx.train)
  train.error[i] <- mean((ridge.pred2-y.train)^2)
  }

plot(log(lambda.grid),train.error)
```




```
# lambda value that gives the smallest training error
lambda.grid[which.min(train.error)]
```

```
## [1] 0.001
```

```
# save the smallest training error of ridge regression
ridge.train.error = train.error[which.min(train.error)]

# We should use the lambda = 0.1321941 that gives the smallest testing error.
ridge.test <- predict(ridge.mod, s = 0.1321941,newx=xx.test)
ridge.train <- predict(ridge.mod, s = 0.1321941,newx=xx.train)

# training error
mean((ridge.train-y.train)^2)
```

```
## [1] 0.03017493
```

```
# test error
mean((ridge.test-y.test)^2)
```

```
## [1] 0.04807068
```

```
# misclassification rate for future observations
res <- ridge.test > 0.5
mean(res != y.test)
```

```
## [1] 0.02472527
```

p)

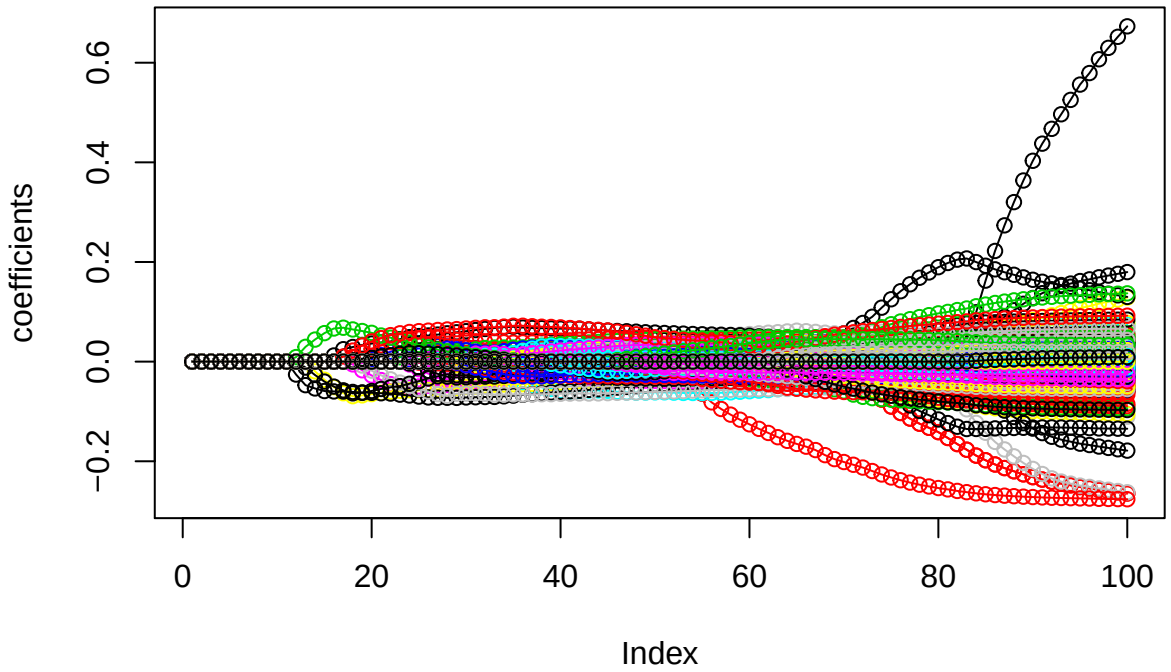
Lasso

```
lambda.grid <- 10^seq(0,-5,length=100)
# variables are on the same scale already.

lasso.mod <- glmnet(xx.train,y.train,alpha=1,lambda=lambda.grid,standardize = FALSE)  # alpha = 0 for L
coeff.lasso <- coef(lasso.mod)
dim(coeff.lasso)
```

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

```
par(mfrow=c(1,1))
plot(coeff.lasso[2,],ylim=c(min(coeff.lasso[-1,]),max(coeff.lasso[-1,])),col=2,type="o",ylab="coefficie
for(i in 2:257){
  lines(coeff.lasso[i,],col=i,type="o")
}
```

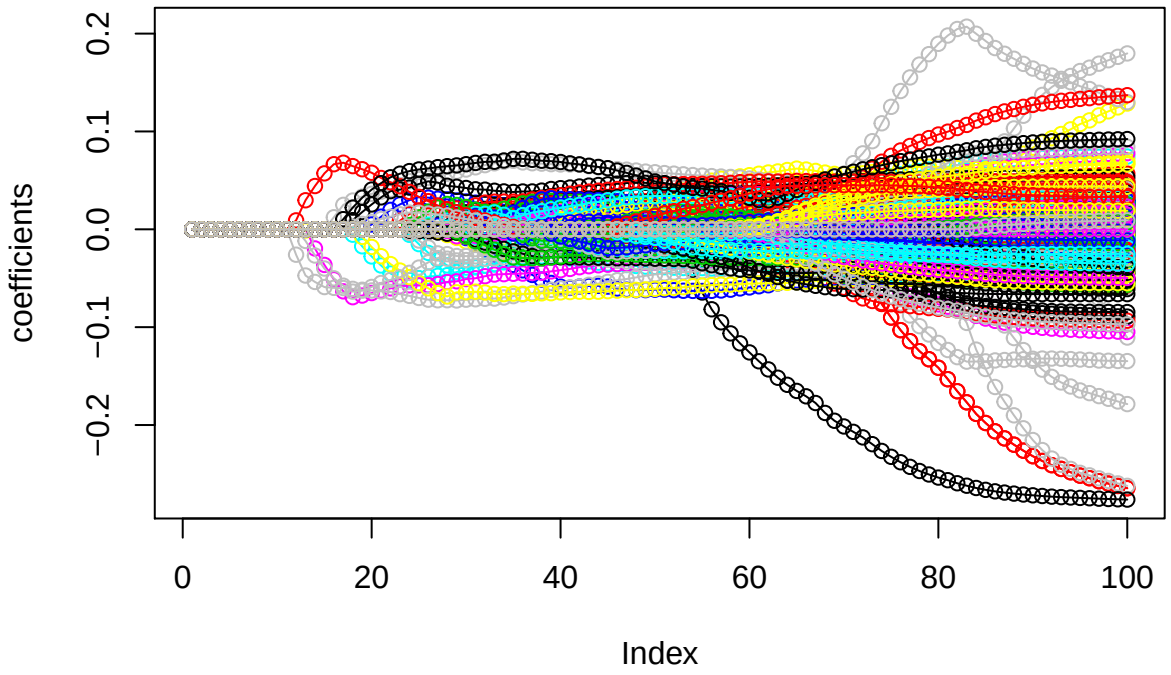



```
which.max(coeff.lasso[,100])
```

```
## V32
##  33
```

```
# remove 33
coeff.lasso <- coeff.lasso[-33, ]

plot(coeff.lasso[2,],ylim=c(min(coeff.lasso[-1,]),max(coeff.lasso[-1,])),col=2,type="o",ylab="coefficie
for(i in 2:256){
  lines(coeff.lasso[i,],col=i,type="o")
}
```



q)

```
# testing err
lasso.pred.test <- predict(lasso.mod,s=0,newx=xx.test,exact =TRUE,x=xx.train,y=y.train)
mean((lasso.pred.test-y.test)^2)
```

```
## [1] 0.1023325
```

```
# training err
lasso.pred.train <- predict(lasso.mod,s=0,newx=xx.train,exact =TRUE,x=xx.train,y=y.train)
mean((lasso.pred.train-y.train)^2)
```

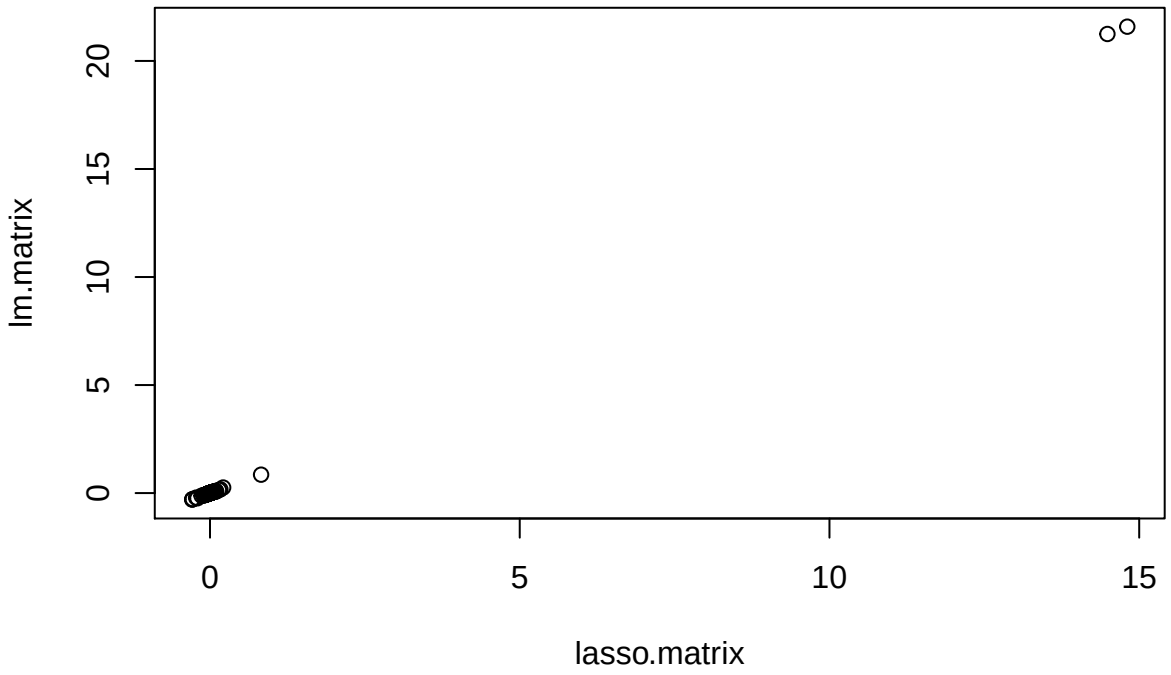
```
## [1] 0.02481378
```

r)

```
# lasso with s=0
lasso.reg <- predict(lasso.mod,s=0, exact = TRUE,type="coefficients",x=xx.train,y=y.train)
# lm
lm.reg <- lm(y.train~xx.train)

lm.matrix <- as.matrix(coef(lm.reg))
lasso.matrix <- as.matrix(lasso.reg)

plot(lasso.matrix, lm.matrix)
```




```
which(lasso.matrix == lm.matrix) == TRUE
```

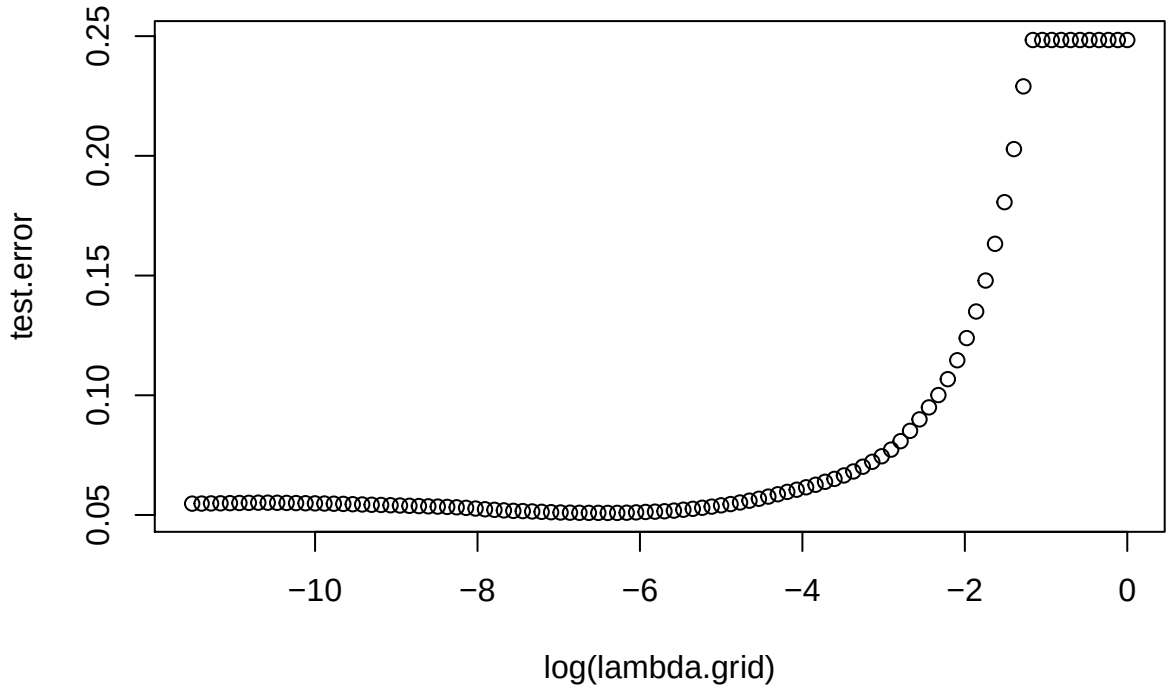
```
## logical(0)
```

```
# not the same
```

s)

```
# testing error
test.error <- rep(NA, 100)
for(i in 1:100){
  lasso.pred <- predict(lasso.mod,s=lambda.grid[i],newx=xx.test)
  test.error[i] <- mean((lasso.pred-y.test)^2)
}

plot(log(lambda.grid),test.error)
```



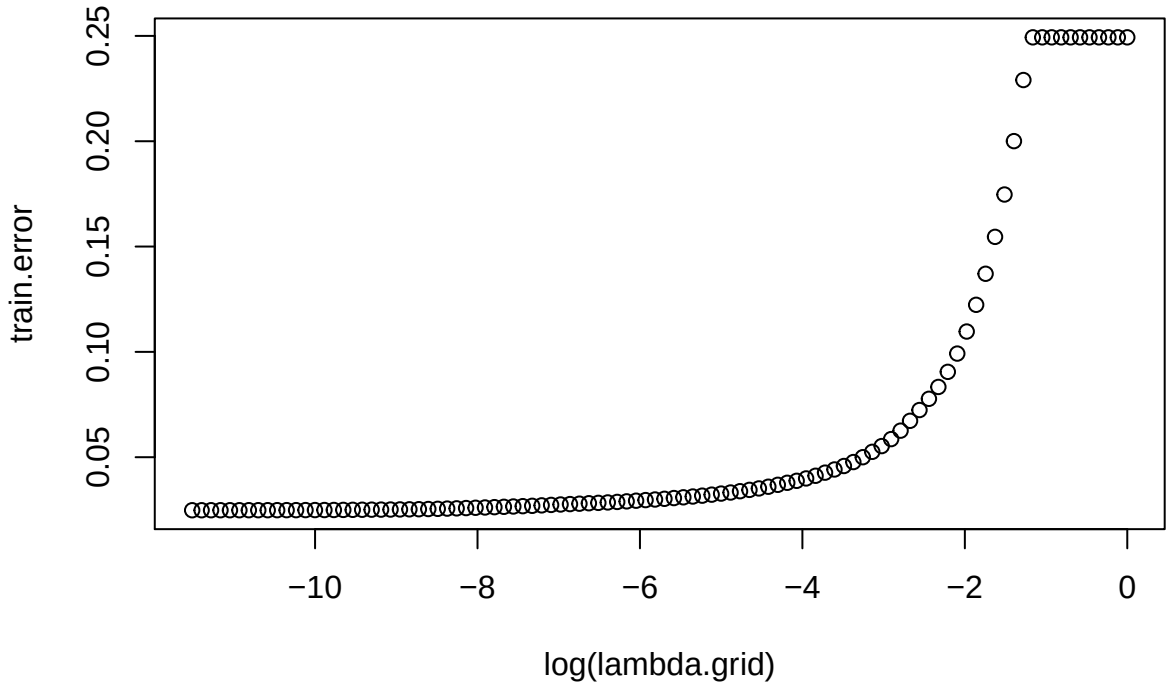

```
# lambda value that gives the smallest test error
lambda.grid[which.min(test.error)]
```

```
## [1] 0.001668101
```

```
# save the smallest test error of lasso regression
lasso.test.error = test.error[which.min(test.error)]

# training error
train.error <- rep(NA, 100)
for(i in 1:100){
  lasso.pred2 <- predict(lasso.mod,s=lambda.grid[i],newx=xx.train)
  train.error[i] <- mean((lasso.pred2-y.train)^2)
  }

plot(log(lambda.grid),train.error)
```




```
# lambda value that gives the smallest training error
lambda.grid[which.min(train.error)]
```

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

```
# save the smallest training error of lasso regression
lasso.train.error = train.error[which.min(train.error)]

# We should use the lambda = 0.001668101 that gives the smallest testing error.
lasso.test <- predict(lasso.mod, s = 0.001668101,newx=xx.test)
lasso.train <- predict(lasso.mod, s = 0.001668101,newx=xx.train)

# training error
mean((lasso.train-y.train)^2)
```

```
## [1] 0.02852913
```

```
# test error
mean((lasso.test-y.test)^2)
```

```
## [1] 0.05085839
```

```
# misclassification rate for future observations
res <- lasso.test > 0.5
mean(res != y.test)
```

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

t)

```
KNN.err <- cbind(k,k.test.err, k.train.err)
colnames(KNN.err) <- c("K","Training error","Testing error")
KNN.err
```

```
##       K Training error Testing error
## [1,]  1     0.02472527   0.000000000
## [2,]  3     0.03021978   0.005039597
## [3,]  5     0.03021978   0.005759539
## [4,]  7     0.03296703   0.006479482
## [5,] 15     0.03846154   0.009359251
```

```
m1 <- cbind(mean((ridge.train-y.train)^2),mean((ridge.test-y.test)^2))
m2 <- cbind(mean((lasso.train-y.train)^2),mean((lasso.test-y.test)^2))
m3 <- cbind(reg.train.err,reg.test.err)
m4 <- cbind(fwd.cp.train.err,fwd.cp.test.err)
m5 <- cbind(fwd.bic.train.err,fwd.bic.test.err)
m6 <- cbind(bwd.cp.train.err,bwd.cp.test.err)
m7 <- cbind(bwd.bic.train.err,bwd.bic.test.err)

rl <- rbind(m1,m2,m3,m4,m5,m6,m7)
rownames(rl) <- c("Ridge","Lasso","OLS","Forward-Cp","Forward-BIC","Backward-Cp","Backward-BIC")
colnames(rl) <- c("Training error","Testing error")

rl
```

```
##              Training error Testing error
## Ridge           0.030174931    0.04807068
## Lasso           0.028529134    0.05085839
## OLS             0.005759539    0.04120879
## Forward-Cp      0.007199424    0.03846154
## Forward-BIC     0.012239021    0.04395604
## Backward-Cp     0.006479482    0.04395604
## Backward-BIC    0.012958963    0.04670330
```

```
# Forward-Cp seems to have the smallest testing error.
```