Lab 1 Report

Mohammed Bakheet

11/24/2019

Assignment 1. Spam classification with nearest neighbors

#1.1 importing xlsx files

```
RNGkind(sample.kind = "Rounding")
data <- readxl::read_excel("D:/Desktop/Machine Learning/Machine Learning/lab01/spambase.xlsx")
data$Spam <- as.factor(data$Spam)
```

#Dividing data into testing and training

```
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data[id,]
test=data[-id,]
```

#2- Logistics Regression

#Prediction for testing data

```
glm.probs.train <- predict(logiModel, type = "response")
glm.probs <- predict(logiModel,newdata = test, type = "response")
#making prediction
glm.pred <- ifelse(glm.probs > 0.5, 1, 0)
glm.pred.train <- ifelse(glm.probs.train > 0.5, 1, 0)
#The confusion matrices
table(glm.pred.train,train$Spam)
```

```
##
## glm.pred.train   0   1
##              0 803  81
##              1 142 344
```

```
table(glm.pred,test$Spam)
```

```
##
## glm.pred   0   1
##        0 791  97
##        1 146 336
```

#When using the confusion matrix it appears that 1127 emails were classified correctly, whereas 243 emails were misclassified

#computing missclassification

```
testMisClassification <- (1 - mean(glm.pred == test$Spam))
testMisClassification
```

```
## [1] 0.1773723
```

#3- Classifying test data more than 0.8

```
glmPredtest <- ifelse(glm.probs > 0.8, 1L, 0L)
glmPredtestTrain <- ifelse(glm.probs.train > 0.8, 1L, 0L)

#Confustion matrix
table(glmPredtestTrain,train$Spam)
```

```
##
## glmPredtestTrain   0   1
##                0 941 335
##                1   4  90
```

```
table(glmPredtest,test$Spam)
```

```
##
## glmPredtest   0   1
##           0 926 367
##           1  11  66
```

```
#computing missclassification

testMisClassification08 <- (1 - mean(glmPredtest == test$Spam))
testMisClassification08
```

```
## [1] 0.2759124
```

#The misclassification rate when $P(Y = 1|X) > 0.5$, is 0.17. Whereas this rate when $P(Y = 1|X) > 0.8$, is 0.27

#4: standard classifier kknn() with k 30

```
library(kknn)
kknnResult <- kknn(Spam ~ ., train = train, test = test, k=30, distance = 2, kernel = "optimal")
kknnPredict <- fitted(kknnResult)
table(kknnPredict, test$Spam)
```

```
##
## kknnPredict   0   1
##           0 672 187
##           1 265 246
```

```
#Calculating misclassification when k = 30
#Misclassification for test data
miscTest <- (1 - mean(kknnPredict == test$Spam))
miscTest
```

```
## [1] 0.329927
```

```
#When K = 1
knnResult1 <- kknn(Spam ~ ., train = train, test = train, k=1)
knnResultTest1 <- kknn(Spam ~ ., train = train, test = test, k=1)
knnPredictTrain1 <- fitted(knnResult1)
knnPredictTest1 <- fitted(knnResultTest1)
table(knnPredictTrain1, train$Spam)
```

```
##
## knnPredictTrain1   0   1
##                0 945   0
##                1   0 425
```

```
table(knnPredictTest1, test$Spam)
```

```
##
## knnPredictTest1   0   1
##                0 640 177
##                1 297 256
```

```
#Calculating misclassification when k = 30
#Misclassification for training data
miscTrain1 <- (1 - mean(knnPredictTrain1 == train$Spam))
miscTrain1
```

```
## [1] 0
```

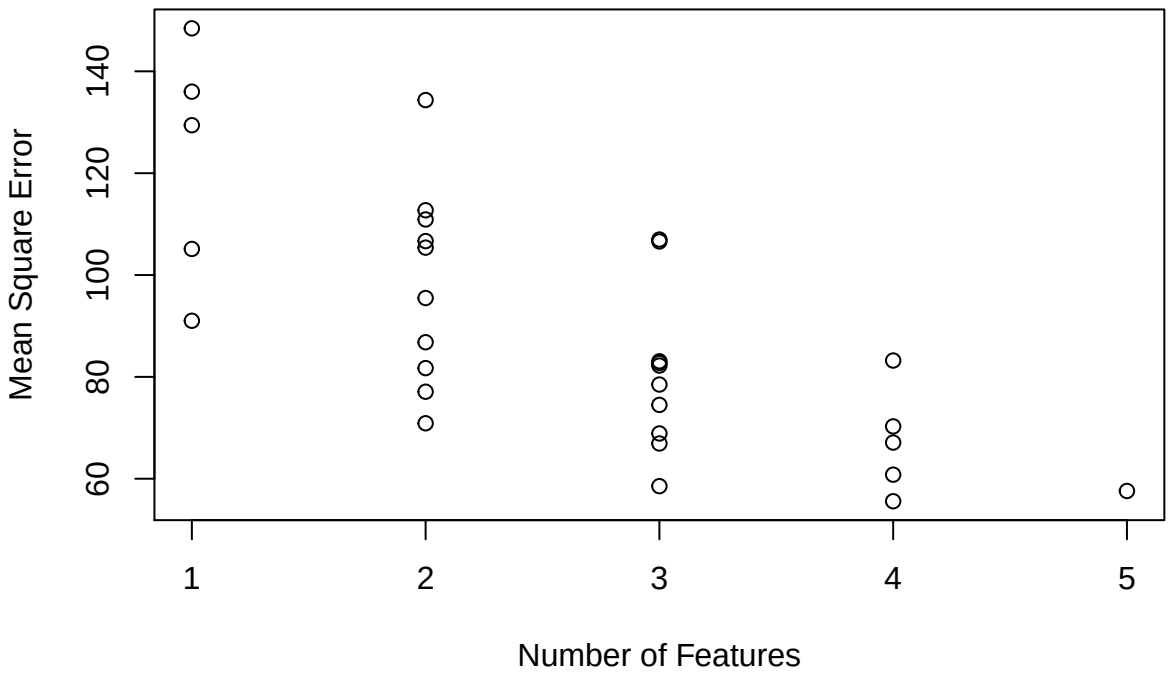
```
#Misclassification for test data
miscTest1 <- (1 - mean(knnPredictTest1 == test$Spam))
miscTest1
```

```
## [1] 0.3459854
```

#When using k = 1 the misclassification the error for the training has become zero which indicates overfitting.

Assignment 3. Feature selection by cross-validation in a linear model.

#linear regression
#In this linear regression model my report was different from the group report because of sample function that was used inside the folds loop, which gave a different result.




```
## $CV
## [1] 55.57471
##
## $Features
## [1] 1 0 1 1 1
```

#The graph represents the mean square error (MSE) on the y axis and the features on the x axis features are Agriculture, Examination, Education, Catholic, Infant.Mortality. If the feature is selected, then it's represented by 1, and if it's not selected it's represented by 0. The optimal number of feature selected by the model is four (Agriculture, Education, Catholic, and Infant.Mortality), the model didn't select the parameter (Examination), which means the military examination score doesn't effect fertility, which sounds logical.

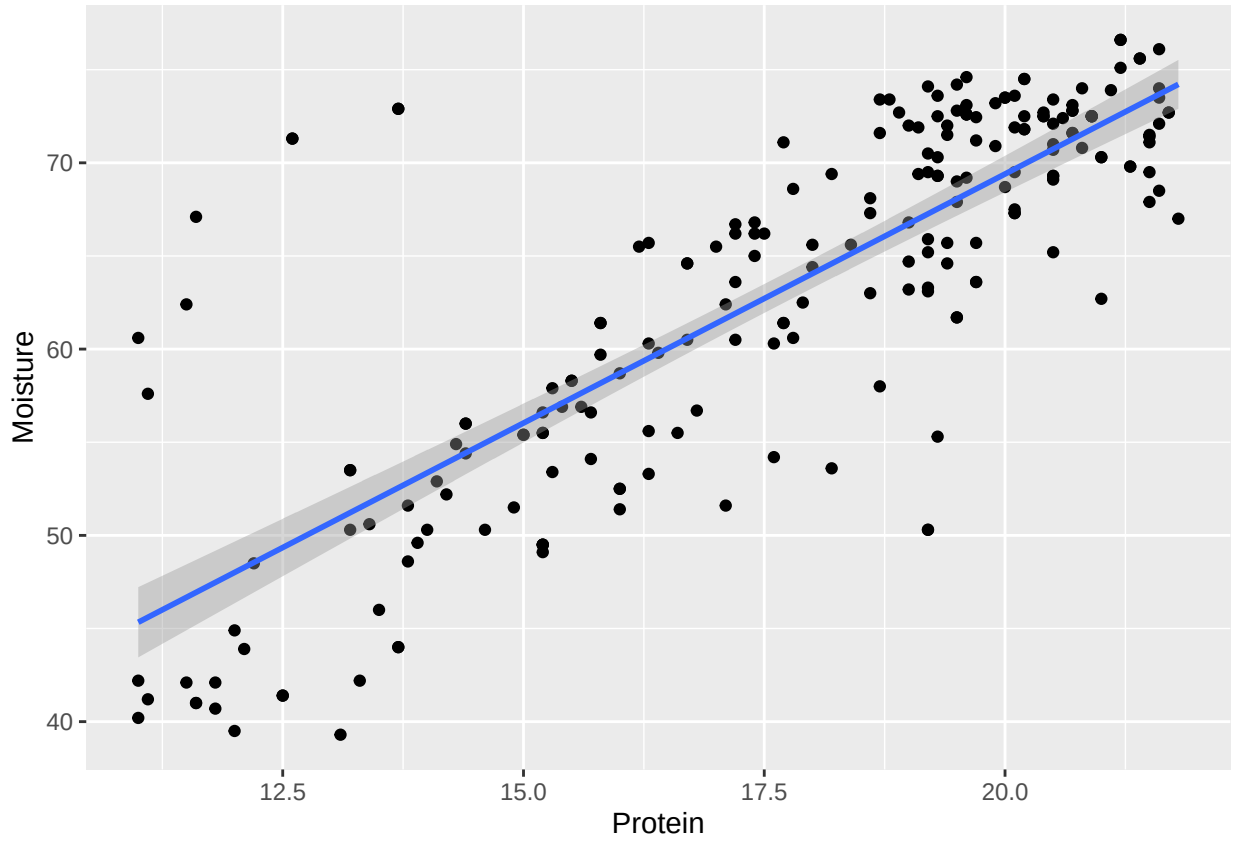
Assignment 4. Linear regression and regularization

#1) importing xlsx files for the data and plotting the moisture versus protein

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

```
## Loaded glmnet 3.0-1
```

```r
data <- readxl::read_excel("D:/Desktop/Machine Learning/Machine Learning/lab01/tecator.xlsx")
#Plotting Moisture versus Protein
ggplot(data,aes(x=Protein,y =Moisture )) +geom_point() +geom_smooth(method = "lm")
```



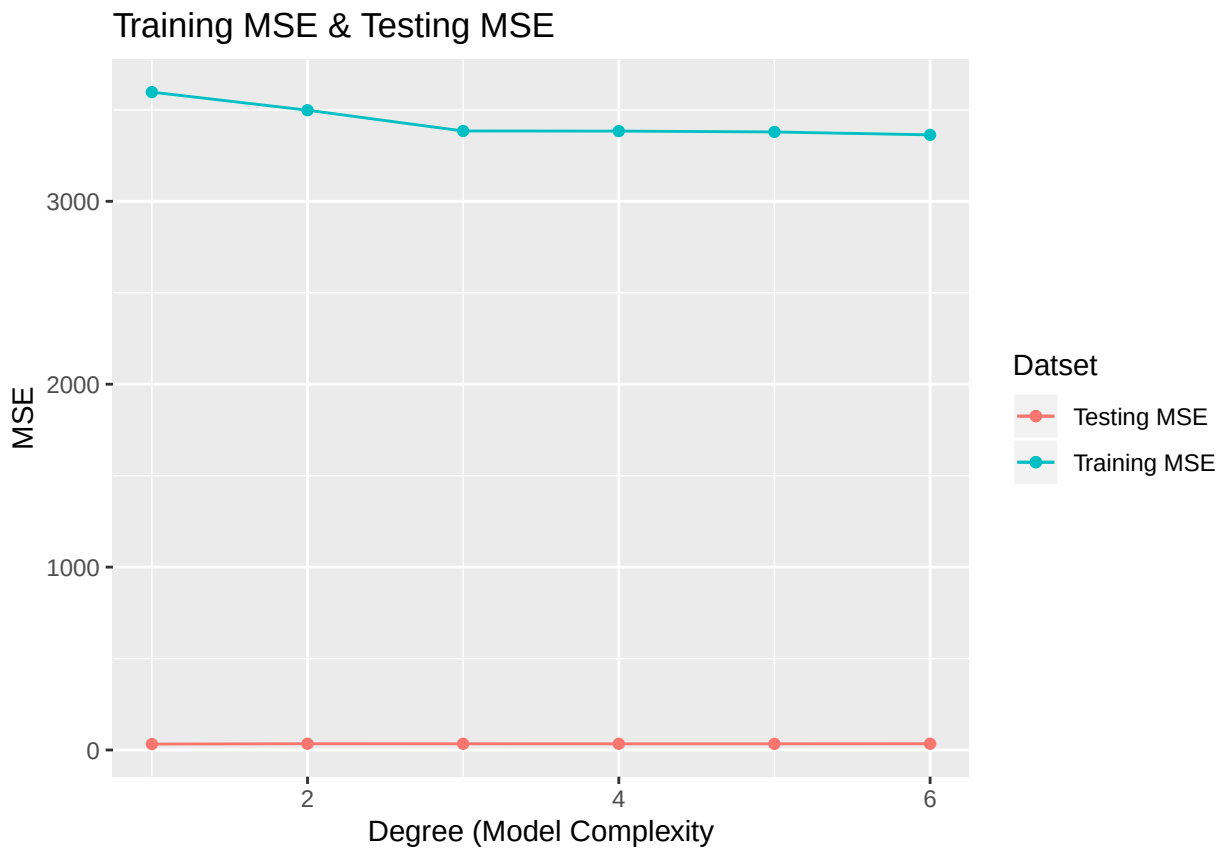
#From the plot, the data could be described very well in a linear model, and it appears that moisture and protein have a positive relationship. In the group report it wasn't described that the two variables could be described by a linear model.

#2) Mi is our Model -> N(M,Sigma^2)
#Mi = B0 + B1X + B2X^2 + ... + BiX^i
#it is appropriate to use MSE criterion when fitting this model because the Moisture is normally distributed and the mean square error (MSE) is the maximum likelihood estimator of the normal distribution, and we would assume the error is normally distributed as well.

#3)Dividing the data into training and testing, and fitting the models

```r
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data[id,]
test=data[-id,]

#Sending the data to get AICs for different models and plotting the MSEs

mses <- getAic(train,test)
ggplot(mses)+
```

```
  geom_line(aes(x = mses$c.1.6., y = mses$trainingMse, color = 'Training MSE')) +
  geom_point(aes(x = mses$c.1.6., y = mses$trainingMse, color = 'Training MSE')) +
  geom_line(aes(x = mses$c.1.6., y = mses$testingMse, color = 'Testing MSE')) +
  geom_point(aes(x = mses$c.1.6., y = mses$testingMse, color = 'Testing MSE')) +
  scale_color_discrete(name = 'Datset') +
  ggtitle('Training MSE & Testing MSE') +
  xlab('Degree (Model Complexity') +
  ylab('MSE')
```



#The best model is when degree = 3 with AIC of 658.3725, and MSE for testing of 39.88347. The more we increase the degree (power) of the model the bigget the error gets for the testing data, however, for the training data the mean square erros decreases as the model degree increases. The model for training has a high bias and low variance when the model was simple, and when the model gets highly complicated, the error increases also. The best model therefore, is when the model is its medium complexity with 3 degrees.

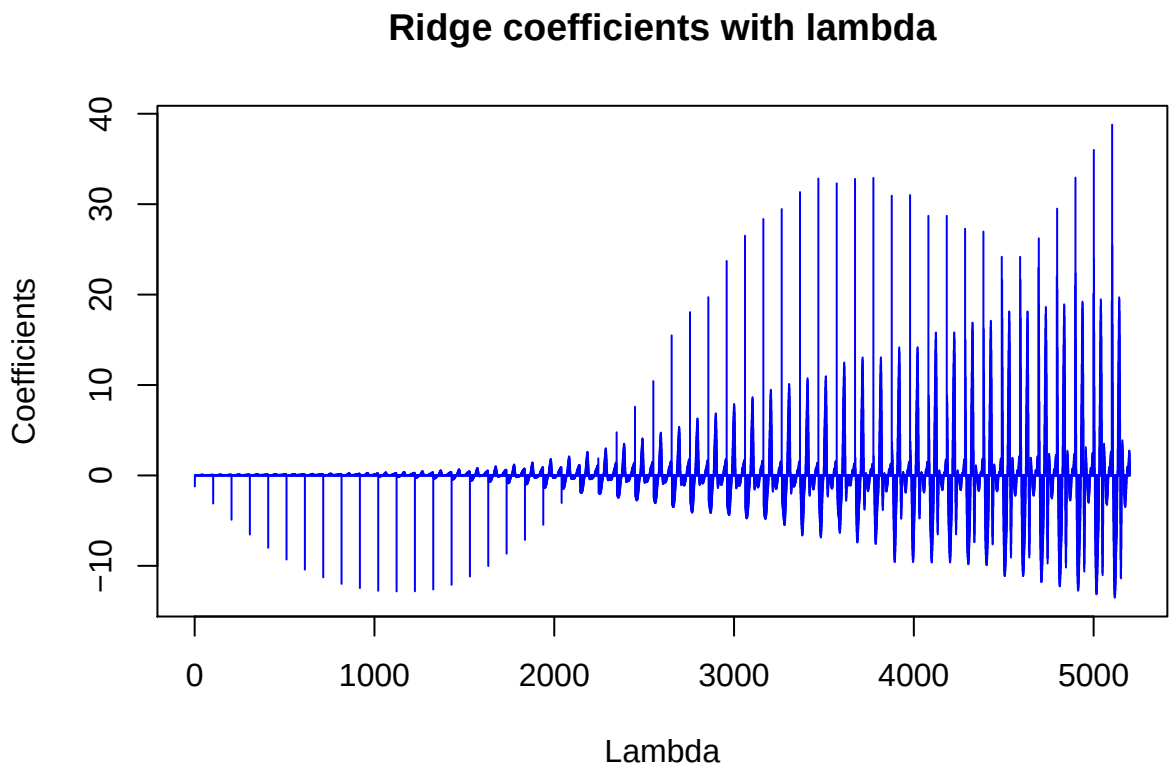
#4) Variable selection of a linear model

```
modelData <- data[,-c(ncol(data),ncol(data)-1)]
linearModelAic <- lm(formula = Fat ~ . , data = modelData)
aicValue <- stepAIC(linearModelAic,direction = "backward")
```

#63 variable are selected as a final model by stepAIC of the total number of channels, meaning AIC is smallest when using these 63 variables and excluding the rest.

#5) Fitting ridge regression

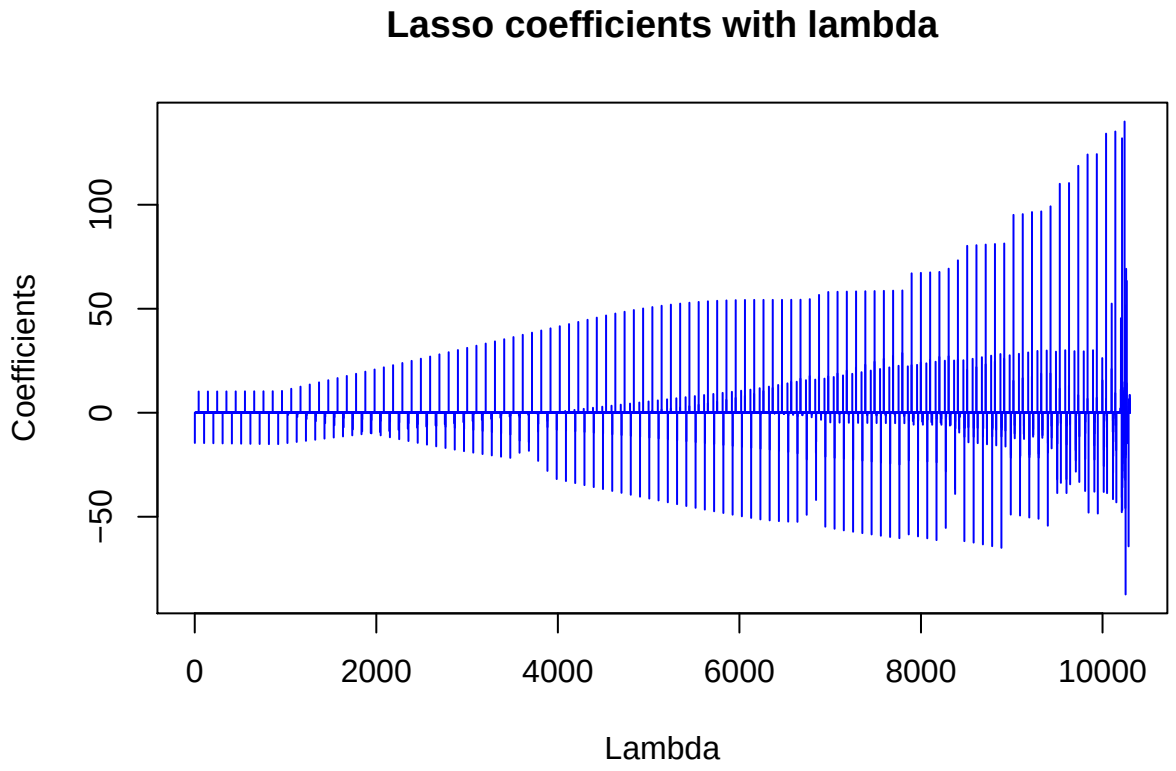
```r
y <- modelData$Fat
x <- model.matrix(Fat~., modelData)[,-1]
y <- modelData$Fat
x <- model.matrix(Fat~., modelData)[,-1]
lambdas <- 10^seq(3, -2, by = -.1)

cv_fit <- cv.glmnet(x, y, alpha = 0, lambda = lambdas)

#Lambda minimum value
lambda_best_ridge <- cv_fit$lambda.min
#The best lambda value is 0.01
#The lowest point in the curve indicates the optimal lambda which is 0.01 in our case,
#and it's the log value of lambda that best minimised the error in cross-validation

best_ridge <- glmnet(x, y, alpha = 0, lambda = lambdas)

#Plotting the coefficients for the best ridge model

plot(coef(best_ridge), main="Ridge coefficients with lambda",type = "h", col="blue", ylab="Coefficients
```



#The coefficients increases when the value of lambda increases and they decrease when lambda decreases, so, coefficients have a positive relationship with lambda.

#6) Fitting lasso regression

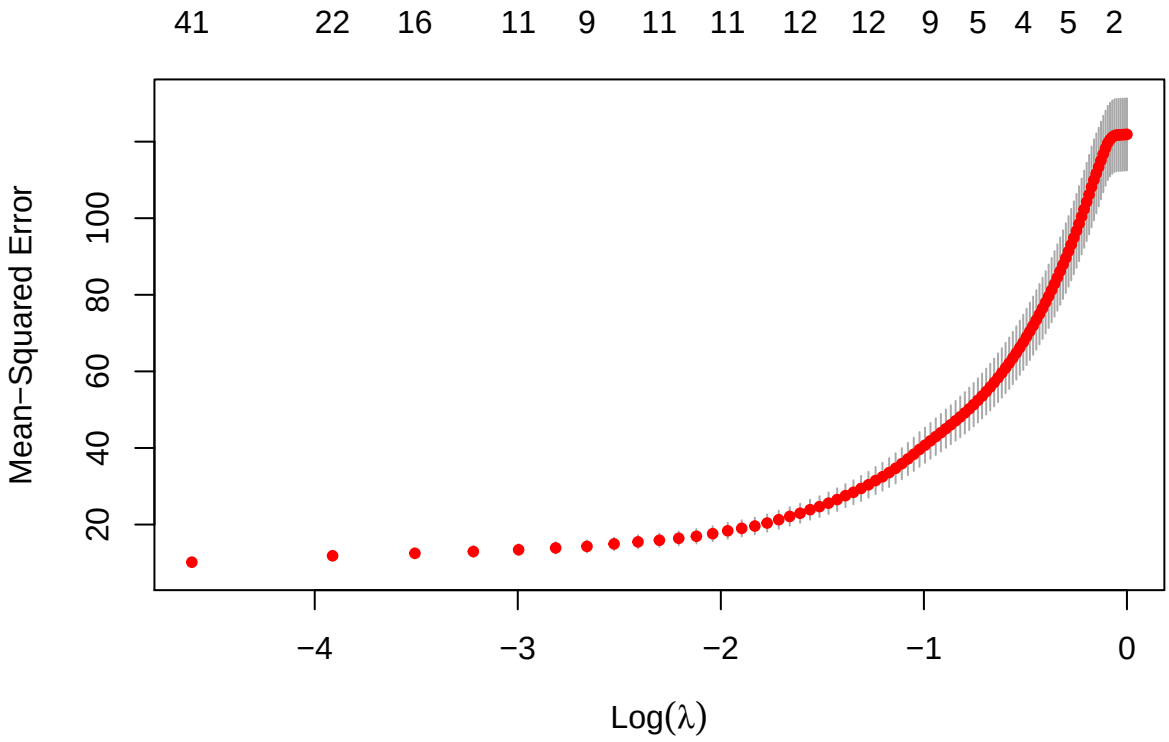
```r
yLasso <- modelData$Fat
xLasso <- model.matrix(Fat~., modelData)[,-1]
lambdasLasso <- seq(0, 1, by = 0.01)

#The best lambda value is 0.01
#The lowest point in the curve indicates the optimal lambda which is 0.01 in our case,
#and it's the log value of lambda that best minimised the error in cross-validation

best_lasso <- glmnet(xLasso, yLasso, alpha = 1, lambda = lambdasLasso)
#Plotting the coefficients for the best model

plot(coef(best_lasso), main="Lasso coefficients with lambda",type = "h", col="blue", ylab="Coefficients
```



#In lasso regression many parameters have become zero when lambda is minimum, thus, coefficients are closed or equal to zero, when lambda increases the coefficient increases accordingly.

#7) The best laso model

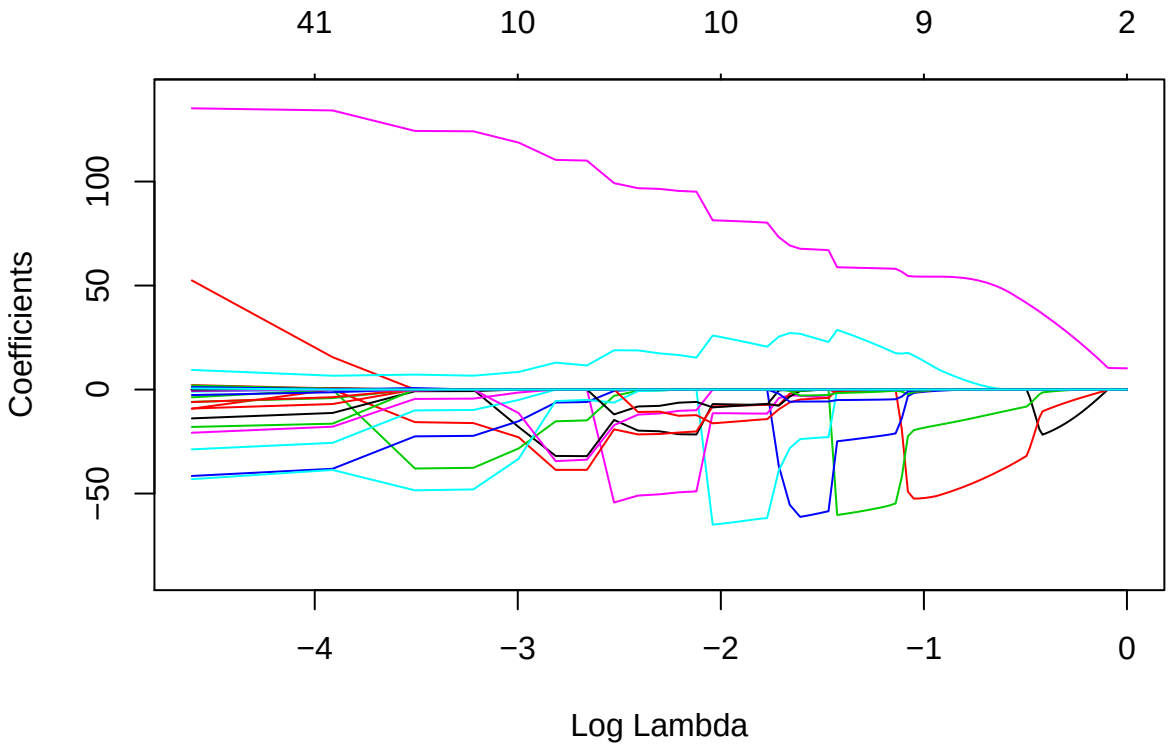
```r
cv.out = cv.glmnet(x, y, alpha = 1,family = "gaussian", lambda = lambdasLasso)
plot(cv.out)
```



#The best lasso model is when lambda is minimal, in our case lambda.min is equal to 0, and all 100 variables are selected by lasso. In this case lasso model simply gives the least squares fit.

#Plotting the coefficients with respoect to lambda

```
plot(best_lasso,xvar="lambda",label=TRUE)
```



#from the plot we can see that lasso has restricted some coefficients to be exactly zero

#8) when comparing the result from the linear model variable selection and lasso variable selection, linear model selected 63 variables whereas lasso chose all the variables.

Code Appendix

```
packages <- c("ggplot2", "plotly","readxl","kknn")
options(tinytex.verbose = TRUE)
knitr::opts_chunk$set(echo = TRUE)
packages <- c("ggplot2", "plotly","readxl","kknn")
options(tinytex.verbose = TRUE)
RNGkind(sample.kind = "Rounding")
data <- readxl::read_excel("D:/Desktop/Machine Learning/Machine Learning/lab01/spambase.xlsx")
data$Spam <- as.factor(data$Spam)

n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data[id,]
test=data[-id,]


logiModel <- glm(Spam ~ . , family = "binomial", data = train)
```

```
glm.probs.train <- predict(logiModel, type = "response")
glm.probs <- predict(logiModel,newdata = test, type = "response")
#making prediction
glm.pred <- ifelse(glm.probs > 0.5, 1, 0)
glm.pred.train <- ifelse(glm.probs.train > 0.5, 1, 0)
#The confusion matrices
table(glm.pred.train,train$Spam)
table(glm.pred,test$Spam)
testMisClassification <- (1 - mean(glm.pred == test$Spam))
testMisClassification


glmPredtest <- ifelse(glm.probs > 0.8, 1L, 0L)
glmPredtestTrain <- ifelse(glm.probs.train > 0.8, 1L, 0L)

#Confustion matrix
table(glmPredtestTrain,train$Spam)
table(glmPredtest,test$Spam)

#computing missclassification

testMisClassification08 <- (1 - mean(glmPredtest == test$Spam))
testMisClassification08

library(kknn)
kknnResult <- kknn(Spam ~ ., train = train, test = test, k=30, distance = 2, kernel = "optimal")
kknnPredict <- fitted(kknnResult)
table(kknnPredict, test$Spam)
#Calculating misclassification when k = 30
#Misclassification for test data
miscTest <- (1 - mean(kknnPredict == test$Spam))
miscTest
#When K = 1
kknnResult1 <- kknn(Spam ~ ., train = train, test = train, k=1)
kknnResultTest1 <- kknn(Spam ~ ., train = train, test = test, k=1)
kknnPredictTrain1 <- fitted(kknnResult1)
kknnPredictTest1 <- fitted(kknnResultTest1)
table(kknnPredictTrain1, train$Spam)
table(kknnPredictTest1, test$Spam)
#Calculating misclassification when k = 30
#Misclassification for training data
miscTrain1 <- (1 - mean(kknnPredictTrain1 == train$Spam))
miscTrain1
#Misclassification for test data
miscTest1 <- (1 - mean(kknnPredictTest1 == test$Spam))
miscTest1


#linear regression
mylin=function(X,Y, Xpred){
  Xpred1=cbind(1,Xpred)
  X = cbind(1,X)
  beta <- (solve(t(X)%*%X))%*%t(X)%*%Y
```

```
  Res=Xpred1%*%beta
  return(Res)
}
myfold <- 0
myCV=function(X,Y,Nfolds){
  n=length(Y)
  p=ncol(X)
  set.seed(12345)
  ind=sample(n,n)
  X1=X[ind,]
  Y1=Y[ind]
  sF=floor(n/Nfolds)
  MSE=numeric(2^p-1)
  Nfeat=numeric(2^p-1)
  Features=list()
  curr=0
  X<- X1
  Y<- Y1
  #X<-X[sample(nrow(X)),]

  #we assume 5 features.
  folds <- cut(seq(1,nrow(X)),breaks=5,labels=FALSE)
  #print(folds)
  #print(nrow(X))
  for (f1 in 0:1)
    for (f2 in 0:1)
      for(f3 in 0:1)
        for(f4 in 0:1)
          for(f5 in 0:1){
            model= c(f1,f2,f3,f4,f5)
            if (sum(model)==0) next()
            SSE=0
            #Creating five folds for the parameters


            #Looping through all folds
            for (k in 1:Nfolds){
              id <- which(folds==k)

              train <- X[-id,which(model==1)]

              test <- X[id,which(model==1)]

              Ypred <- mylin(train, Y[-id] ,test)
              Yp <- Y[id]
              SSE=SSE+sum((Ypred-Yp)^2)
            }
            curr=curr+1
            MSE[curr]=SSE/n
            Nfeat[curr]=sum(model)
            Features[[curr]]=model

          }
```

```
  plot(Nfeat, MSE, xlab = "Number of Features",ylab = "Mean Square Error")
  i=which.min(MSE)
  return(list(CV=MSE[i], Features=Features[[i]]))
}

myCV(as.matrix(swiss[,2:6]), swiss[[1]], 5)
library(MASS)
library(glmnet)
library(ggplot2)
data <- readxl::read_excel("D:/Desktop/Machine Learning/Machine Learning/lab01/tecator.xlsx")
#Plotting Moisture versus Protein
ggplot(data,aes(x=Protein,y =Moisture )) +geom_point() +geom_smooth(method = "lm")
#getting AIC function that takes training and testing data
getAic <- function(train, test){
  aicVector <- vector()
  trainingMse <- vector()
  testingMse <- vector()
  for (i in 1:6){
  linearModel <- lm(formula = Moisture ~ poly(Protein, degree = i, raw = TRUE), data = train)
  aicVector <- c(aicVector,AIC(linearModel))
  #Calculating the training MSE
  trainingMse <- c(trainingMse, mean(sum(linearModel$residuals^2)))
  #Calculating the testing MSE
  testingMse <- c(testingMse, mean((test$Moisture - predict.lm(linearModel, test)) ^ 2))
  }
  result <- data.frame(trainingMse,testingMse,c(1:6))
  return(result)
}
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data[id,]
test=data[-id,]

#Sending the data to get AICs for different models and plotting the MSEs

mses <- getAic(train,test)
ggplot(mses)+
  geom_line(aes(x = mses$c.1.6., y = mses$trainingMse, color = 'Training MSE')) +
  geom_point(aes(x = mses$c.1.6., y = mses$trainingMse, color = 'Training MSE')) +
  geom_line(aes(x = mses$c.1.6., y = mses$testingMse, color = 'Testing MSE')) +
  geom_point(aes(x = mses$c.1.6., y = mses$testingMse, color = 'Testing MSE')) +
  scale_color_discrete(name = 'Datset') +
  ggtitle('Training MSE & Testing MSE') +
  xlab('Degree (Model Complexity') +
  ylab('MSE')
modelData <- data[,-c(ncol(data),ncol(data)-1)]
linearModelAic <- lm(formula = Fat ~ . , data = modelData)
aicValue <- stepAIC(linearModelAic,direction = "backward")
y <- modelData$Fat
x <- model.matrix(Fat~., modelData)[,-1]
y <- modelData$Fat
```

```r
x <- model.matrix(Fat~., modelData)[,-1]
lambdas <- 10^seq(3, -2, by = -.1)

cv_fit <- cv.glmnet(x, y, alpha = 0, lambda = lambdas)

#Lambda minimum value
lambda_best_ridge <- cv_fit$lambda.min
#The best lambda value is 0.01
#The lowest point in the curve indicates the optimal lambda which is 0.01 in our case,
#and it's the log value of lambda that best minimised the error in cross-validation

best_ridge <- glmnet(x, y, alpha = 0, lambda = lambdas)

#Plotting the coefficients for the best ridge model

plot(coef(best_ridge), main="Ridge coefficients with lambda",type = "h", col="blue", ylab="Coefficients

yLasso <- modelData$Fat
xLasso <- model.matrix(Fat~., modelData)[,-1]
lambdasLasso <- seq(0, 1, by = 0.01)

#The best lambda value is 0.01
#The lowest point in the curve indicates the optimal lambda which is 0.01 in our case,
#and it's the log value of lambda that best minimised the error in cross-validation

best_lasso <- glmnet(xLasso, yLasso, alpha = 1, lambda = lambdasLasso)
#Plotting the coefficients for the best model

plot(coef(best_lasso), main="Lasso coefficients with lambda",type = "h", col="blue", ylab="Coefficients
cv.out = cv.glmnet(x, y, alpha = 1,family = "gaussian", lambda = lambdasLasso)
plot(cv.out)


plot(best_lasso,xvar="lambda",label=TRUE)
```