tle: “Lab1_ML”
thor: “vinbe289”
te: “November 26, 2018”
tput: html_document

Assignment1

1.1

```
library(readxl)
spam_data <- read_xlsx("spambase.xlsx")
n=dim(spam_data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=spam_data[id,]
test=spam_data[-id,]
```

1.2

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

```
##
## Call:
## glm(formula = Spam ~ ., family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5205  -0.4402  -0.0005   0.6584   3.6196
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.508e+00  2.011e-01   7.499 6.44e-14 ***
## Word1       -7.192e-01  5.015e-01  -1.434 0.151520
## Word2        3.994e-02  3.014e-01   0.133 0.894580
## Word3       -3.529e-01  1.839e-01  -1.919 0.055019 .
## Word4        1.370e-01  1.117e-01   1.226 0.220230
## Word5        1.221e-01  1.413e-01   0.864 0.387400
## Word6        2.887e-01  4.153e-01   0.695 0.486888
## Word7       -2.948e-01  3.348e-01  -0.880 0.378676
## Word8       -1.034e-01  3.510e-01  -0.295 0.768337
## Word9       -1.085e-01  4.053e-01  -0.268 0.788983
## Word10      -3.091e-02  1.668e-01  -0.185 0.852991
## Word11      -6.088e-01  6.790e-01  -0.897 0.369902
## Word12       1.614e-01  1.073e-01   1.505 0.132378
## Word13       7.811e-01  3.572e-01   2.187 0.028771 *
```

```
## Word14      -4.819e-01  3.003e-01  -1.605 0.108598
## Word15      -1.305e-01  3.884e-01  -0.336 0.736861
## Word16       3.171e-01  2.332e-01   1.360 0.173891
## Word17      -9.201e-02  2.823e-01  -0.326 0.744479
## Word18       2.330e-02  2.322e-01   0.100 0.920074
## Word19       3.694e-04  5.746e-02   0.006 0.994871
## Word20       1.688e-02  3.181e-01   0.053 0.957687
## Word21      -2.821e-02  1.072e-01  -0.263 0.792524
## Word22      -4.767e-01  3.200e-01  -1.490 0.136337
## Word23       2.541e-01  3.413e-01   0.745 0.456525
## Word24      -2.483e-01  6.255e-01  -0.397 0.691372
## Word25      -7.828e-02  5.852e-02  -1.338 0.181027
## Word26       3.733e-03  1.358e-01   0.027 0.978071
## Word27      -2.238e-01  1.077e-01  -2.078 0.037749 *
## Word28       1.320e-01  1.880e-01   0.702 0.482776
## Word29      -8.131e-02  9.119e-02  -0.892 0.372564
## Word30      -1.815e+00  6.195e-01  -2.930 0.003391 **
## Word31      -4.694e+00  1.853e+00  -2.533 0.011296 *
## Word32      -1.194e+02  1.513e+04  -0.008 0.993703
## Word33      -2.899e+00  6.794e-01  -4.268 1.98e-05 ***
## Word34      -3.710e+00  4.352e+00  -0.852 0.394004
## Word35      -7.033e+00  1.996e+00  -3.522 0.000428 ***
## Word36      -1.678e+00  3.810e-01  -4.404 1.06e-05 ***
## Word37      -8.583e-01  2.175e-01  -3.947 7.92e-05 ***
## Word38      -6.043e-01  1.279e+00  -0.472 0.636575
## Word39      -1.877e+00  4.282e-01  -4.384 1.16e-05 ***
## Word40       7.393e-02  3.400e-01   0.217 0.827885
## Word41      -3.326e+02  1.656e+04  -0.020 0.983978
## Word42      -5.352e+00  1.302e+00  -4.109 3.98e-05 ***
## Word43      -2.592e+00  7.353e-01  -3.525 0.000423 ***
## Word44      -2.931e+00  6.601e-01  -4.441 8.96e-06 ***
## Word45      -1.141e+00  1.681e-01  -6.785 1.16e-11 ***
## Word46      -3.288e+00  5.178e-01  -6.350 2.15e-10 ***
## Word47      -3.741e+00  2.030e+00  -1.843 0.065399 .
## Word48      -4.390e+00  1.473e+00  -2.980 0.002878 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1696.82  on 1369  degrees of freedom
## Residual deviance:  928.54  on 1321  degrees of freedom
## AIC: 1026.5
##
## Number of Fisher Scoring iterations: 23
```

```
y_predicted_train <- predict(glm_model_train,train,type = "response")
class_train  <- ifelse(y_predicted_train > 0.50, "Spam", "Not Spam")
confusion_matrix_train <- table(train$Spam, class_train)
confusion_matrix_train
```

```
##    class_train
##     Not Spam Spam
##   0      803  142
```

```
##   1       81  344
```

```
1 - (sum(diag(confusion_matrix_train))/sum(confusion_matrix_train))
```

```
## [1] 0.1627737
```

The following observation is made for the training dataset. There are two classes in the cunfusion matrix i.e. Not spam=0, and Spam=1. The confusion matrix shows the number of actual Spam:425, Not spam:945 mails and predicted number of Spam:486 and Not spam:884 mails.There are 344 True positive, 803 True negative, 142 False predicted positive and 81 false predicted negative values.Misclassification rate or error rate is 16.27%.

```
y_predicted_test <- predict(glm_model_train,test,type = "response")
class_test <- ifelse(y_predicted_test > 0.50, "Spam", "Not Spam")
confusion_matrix_test <- table(test$Spam, class_test)
confusion_matrix_test
```

```
##    class_test
##     Not Spam Spam
##   0      791  146
##   1       97  336
```

```
1 - (sum(diag(confusion_matrix_test))/sum(confusion_matrix_test))
```

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

The following observation is made for the test dataset. There are two classes in the cunfusion matrix i.e. Not spam=0, and Spam=1. The confusion matrix shows the number of actual Spam:433, Not spam:937 mails and predicted number of Spam:482 and Not spam:888 mails.There are 336 True positive, 791 True negative, 146 False predicted positive and 97 false predicted negative values.Misclassification rate or error rate is 17.73%.

1.3

```
class_train2 <- ifelse(y_predicted_train > 0.90, "Spam", "Not Spam")
confusion_matrix_train2 <- table(train$Spam, class_train2)
confusion_matrix_train2
```

```
##    class_train2
##     Not Spam Spam
##   0      944    1
##   1      419    6
```

```
1 - (sum(diag(confusion_matrix_train2))/sum(confusion_matrix_train2))
```

```
## [1] 0.3065693
```

```
class_test2 <- ifelse(y_predicted_train > 0.90, "Spam", "Not Spam")
confusion_matrix_test2 <- table(test$Spam, class_test2)
confusion_matrix_test2
```

```
##    class_test2
##     Not Spam Spam
##   0      933    4
##   1      430    3
```

```
1 - (sum(diag(confusion_matrix_test2))/sum(confusion_matrix_test2))
```

```
## [1] 0.3167883
```

With the change in value of probability, the misclassification rate has increased from 16.27% to 30.65% for train data set and 17.73% to 31.67% for test data set, and the number of spam mails predicted has reduced. So it can be seen that though the misclassification rate increases the proability of classifying a mail which is not a spam as spam, which is a serious error, is reduced.

1.4

```
library(kknn)

kknn_model <- kknn(Spam ~ .,train,train,k=30)
y_predicted_kknn <- predict(kknn_model)
class_kknn <- ifelse(y_predicted_kknn > 0.50, "Spam", "Not Spam")
confusion_matrix_kknn <- table(train$Spam, class_kknn)
confusion_matrix_kknn
```

```
##    class_kknn
##     Not Spam Spam
##   0      807  138
##   1       98  327
```

```
1 - (sum(diag(confusion_matrix_kknn))/sum(confusion_matrix_kknn))
```

```
## [1] 0.1722628
```

```
kknn_model1.1 <- kknn(Spam ~ .,train,test,k=30)
y_predicted_kknn1.1 <- predict(kknn_model1.1)
class_kknn1.1 <- ifelse(y_predicted_kknn1.1 > 0.50, "Spam", "Not Spam")
confusion_matrix_kknn2 <- table(test$Spam, class_kknn1.1)
confusion_matrix_kknn2
```

```
##    class_kknn1.1
##     Not Spam Spam
##   0      672  265
##   1      187  246
```

```
1 - ((sum(diag(confusion_matrix_kknn2))/sum(confusion_matrix_kknn2)))
```

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

The misclassification rate for train data set is 17.22% and for test data set it is 32.99% whereas in step 2 it was 16.27% and 17.73% for train and test data set respectively.

1.5

```
kknn_model2 <- kknn(Spam ~ .,train,train,k=1)
y_predicted_kknn2 <- predict(kknn_model2)
class_kknn2 <- ifelse(y_predicted_kknn2 > 0.50, "Spam", "Not Spam")
confusion_matrix_kknn3 <- table(train$Spam, class_kknn2)
confusion_matrix_kknn3
```

```
##    class_kknn2
##     Not Spam Spam
##   0      945    0
##   1        0  425
```

```
1 - (sum(diag(confusion_matrix_kknn3))/sum(confusion_matrix_kknn3))
```

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

```
kknn_model2.1 <- kknn(Spam ~ .,train,test,k=1)
y_predicted_kknn2.1 <- predict(kknn_model2.1)
class_kknn2.1 <- ifelse(y_predicted_kknn2.1 > 0.50, "Spam", "Not Spam")
confusion_matrix_kknn4 <- table(test$Spam, class_kknn2.1)
confusion_matrix_kknn4
```

```
##    class_kknn2.1
##     Not Spam Spam
##   0      640  297
##   1      177  256
```

```
1 - ((sum(diag(confusion_matrix_kknn4))/sum(confusion_matrix_kknn4)))
```

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

The decrease in k value leads to the increase in misclassification rate. The misclassification rate for train data set is 0% and for test data set it is 34.59% whereas in step 4 it was 17.22% and 32.99% for train and test data set respectively. With the 1-nearest neighbor model, each observation of the training dataset is potentially the center of an area predicting the outcome, with the other neighbors potentially predicting the other possible outcomes and hence making it highly complex. With k=30, the areas predicting each outcome will be more smooth as its the majority of the 30 nearest neighbors that predict the outcome of any observation. There are less number of areas which are larger and less complex.

Assignment 3

```
library(MASS)
library(ggplot2)

data <- swiss #assigning the dataset of interest to variable data
y <- as.vector(data[,1]) #defining the dependent variable by mentioning corresponding column number in
x <- as.matrix(data[,c(2:6)]) #defining the predictor or independent variables
Nfolds <- 5 #number of folds needed


#function to calculate weights for each column
fun_weight <- function(X,Y)
{
  w <- ginv(t(X)%*%X)%*%t(X)%*%Y
}
n <- dim(data)[1] #number of rows in the dataset
set.seed(12345)
sample_n <- sample(1:n) #shuffle the indeces (or rownumbers)
ids <- list() #empty list to store the indeces divided into Nfolds groups
cv_score <- c() #empty vector to store cv scores

#function to calculate cv score
cv_func <- function(X,Y,Nfolds)
{
  start <- 1 #starts from the first index
  #splitting indeces into Nfold groups & performing Nfolds cross validation
  for(i in 1:Nfolds)
    {
      #creating Nfolds-1 groups with equal number of indeces in them
      if(i<Nfolds)
      {
        end <- start+(as.integer(n/Nfolds)-1)
        ids[[i]] <- sample_n[start:end]
        start <- end+1
      }
      #creating the Nfolds'th group with the remaining indeces
      else if(i==Nfolds)
      {
        end <- n
        ids[[i]] <- sample_n[start:end]
      }

      #splitting the data into train and test data for independent(Y) and dependent(X) variables
      X_test <- X[as.vector(ids[[i]]),]
      X_train <- X[-as.vector(ids[[i]]),]
      Y_test <- Y[as.vector(ids[[i]])]
      Y_train <- Y[-as.vector(ids[[i]])]

      weights <- as.matrix(fun_weight(X=X_train,Y=Y_train))
      y_cap <- X_test%*%weights #y estimate for each column(independent variable) in X
      loss <- y_cap-Y_test #calculating loss for each observation in column Y(dependent variable)
      cv <- sum(loss*loss)/length(Y_test) #cv score for each fold
```

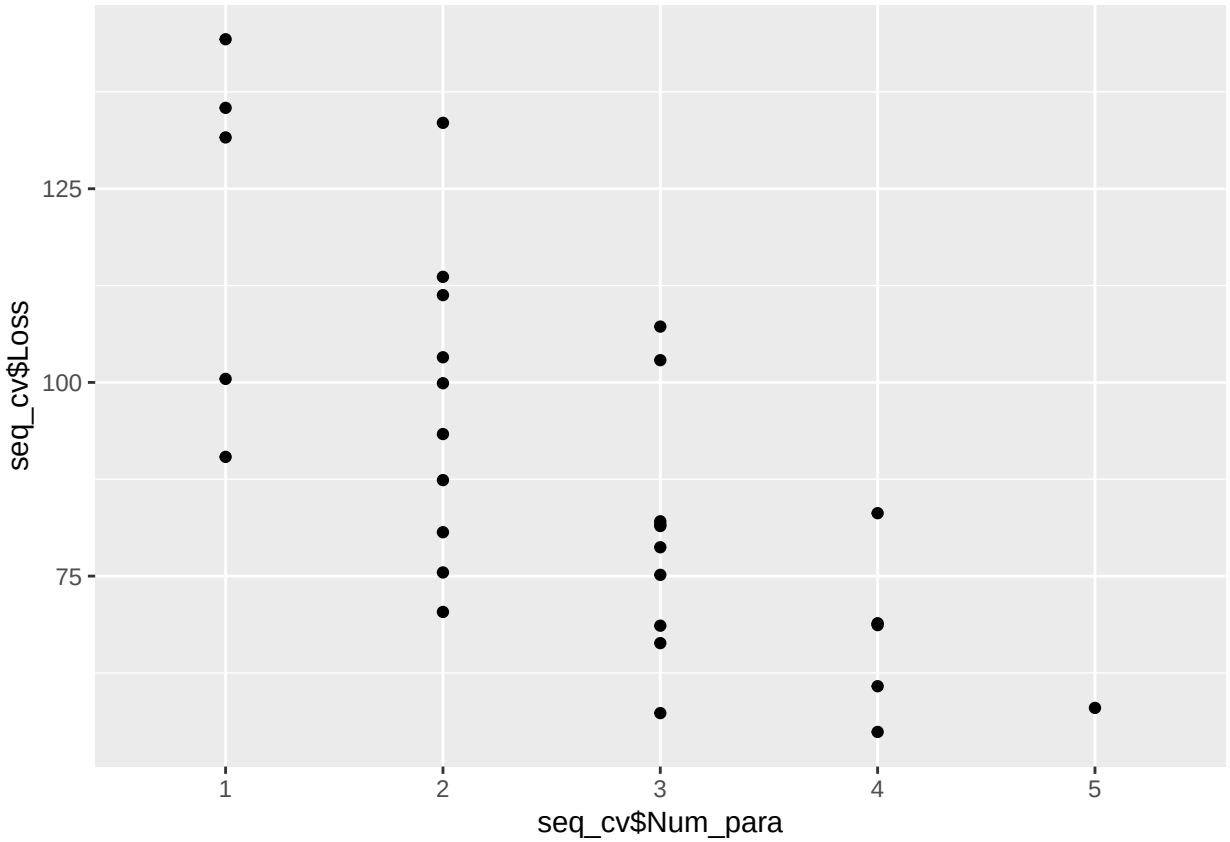
```
      cv_score[i] <- cv
  }
  average_cv <- sum(cv_score)/Nfolds #average of cv score for Nfolds
}

#producing different combinations of column names(numbers or independent variables) of X(matrix of inde
seq_cv <- matrix(0, nrow = 0, ncol = 3)
for(k in 1:ncol(x))
{
  combs <- combn(1:ncol(x),k)
  for(j in 1:ncol(combs))
  {
    x_new <- as.matrix(x[,combs[,j]])
    x_new <- cbind(x_new, 1) #adding the incercept
    seq <- paste(combs[,j], collapse = ",") #converting the sequence into string
    avg_cv_score <- cv_func(X = x_new,Y = y,Nfolds)
    seq_cv <- rbind(seq_cv, c(seq,avg_cv_score, k))
  }
}

seq_cv = as.data.frame(seq_cv)
colnames(seq_cv) = c("Seq", "Loss","Num_para")
seq_cv$Loss = as.numeric(as.character(seq_cv$Loss))
seq_cv$Seq = as.character(seq_cv$Seq)

cv_func(X= x,Y= y,Nfolds)

cat("Optimal subset of features: ",seq_cv$Seq[which.min(seq_cv$Loss)])
```

```
## Optimal subset of features:  1,3,4,5
```

```
cat("Cross validation score: ",min(seq_cv$Loss))
```

```
## Cross validation score:  54.88725
```

```
ggplot()+geom_point(data= seq_cv,aes(x= seq_cv$Num_para,y= seq_cv$Loss))
```

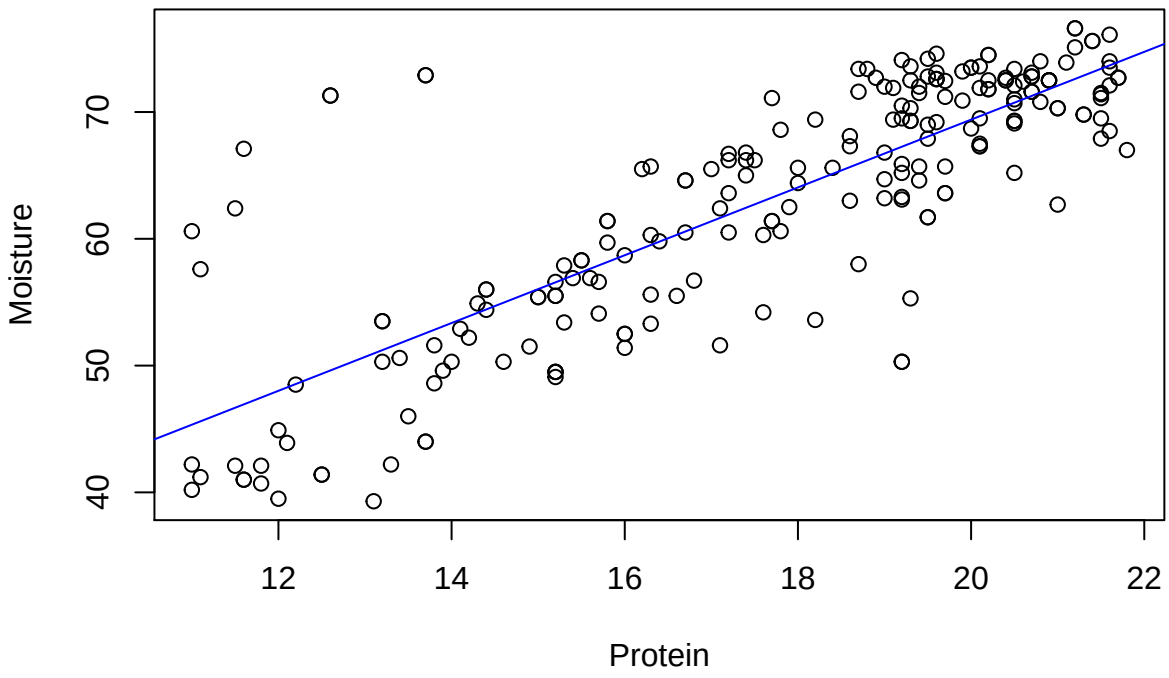


It can be seen that as the number of features considered in the features subset increases the CV score decreases. A linear model fits better, will have a good fit with lesser loss, when the number of independent variables or predictors, to predict the values of dependent variable, increases for a particular combination of variables out of the available independent variables. According to the result the optimal subset of features contained the following features: Agriculture, Education, Catholic, Infant.mortality. It is reasonable that these specific features have largest impact on the target, since all these variables are related to Fertility in some way.

Assignment 4

4.1

```
library(readxl)
tecator <- read_xlsx("tecator.xlsx")
plot(Moisture~Protein, data = tecator)
model <- lm(Moisture~Protein, data = tecator)
abline(model, col="blue")
```



These data can be described well by a linear model.

4.2

p(y|x,w)=N(w$_0$ + $\sum_{i=1}^{n} w_i x^i$,σ^2)

i = 1,2,3,4,...n

x is protein

y is the expected moisture

It is appropriate to use the Mean Squared Error criterion when fitting this model to the training data as the model is fitted using the Least Sqaures Method, where the fitted line is chosen such that the vertical distances from the data are the least.

4.3

```
library(dplyr)
library(plotly)
library(ggplot2)

tecator$Protein2<-(tecator$Protein)^2
tecator$Protein3<-(tecator$Protein)^3
```

```r
tecator$Protein4<-(tecator$Protein)^4
tecator$Protein5<-(tecator$Protein)^5
tecator$Protein6<-(tecator$Protein)^6

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

model1 = lm(Moisture~Protein,data=train)
train_MSE1 = mean(model1$residuals^2)
valid_1 = predict(model1, valid)
valid_MSE11 = mean((valid_1-valid$Moisture)^2)

model2 = lm(Moisture~Protein2+Protein,data=train)
train_MSE2 = mean(model2$residuals^2)
valid_2 = predict(model2, valid)
valid_MSE22 = mean((valid_2-valid$Moisture)^2)

model3 = lm(Moisture~Protein3+Protein2+Protein,data=train)
train_MSE3 = mean(model3$residuals^2)
valid_3 = predict(model3, valid)
valid_MSE33 = mean((valid_3-valid$Moisture)^2)

model4 = lm(Moisture~Protein4+Protein3+Protein2+Protein,data=train)
train_MSE4 = mean(model4$residuals^2)
valid_4 = predict(model4, valid)
valid_MSE44 = mean((valid_4-valid$Moisture)^2)

model5 = lm(Moisture~Protein5+Protein4+Protein3+Protein2+Protein,data=train)
train_MSE5 = mean(model5$residuals^2)
valid_5 = predict(model5, valid)
valid_MSE55 = mean((valid_5-valid$Moisture)^2)

model6 = lm(Moisture~Protein6+Protein5+Protein4+Protein3+Protein2+Protein,data=train)
train_MSE6 = mean(model6$residuals^2)
valid_6 = predict(model6, valid)
valid_MSE66 = mean((valid_6-valid$Moisture)^2)

x <- list(title = "Model ids")
y <- list(title ="Mean Squared Error")

train_mse<-c(train_MSE1,train_MSE2,train_MSE3,train_MSE4,train_MSE5,train_MSE6)
valid_mse<-c(valid_MSE11,valid_MSE22,valid_MSE33,valid_MSE44,valid_MSE55,valid_MSE66)
mse_values<-c(train_mse, valid_mse)
plot_mse<-data.frame(id=1:6,mse_values,datasetfrom=c("train","train","train","train","train","train","va

p1 <- ggplot(plot_mse, aes(y=plot_mse$mse_values, x=plot_mse$id, color=plot_mse$datasetfrom)) + geom_lin
  ggtitle("Model id vs Mean Squared Error")
p1
```

According to the plot, the best fit for validation/test data is the model with i=1 and for training data it is i=6. As the complexity of the model fitted for training data increases as a result of increase in degrees of polynomial, the model fits better to training data and leads to overfitting. With the overfitting of training data the bias decreases and the variance increases which is evident in the plot for validation data error values.

4.4

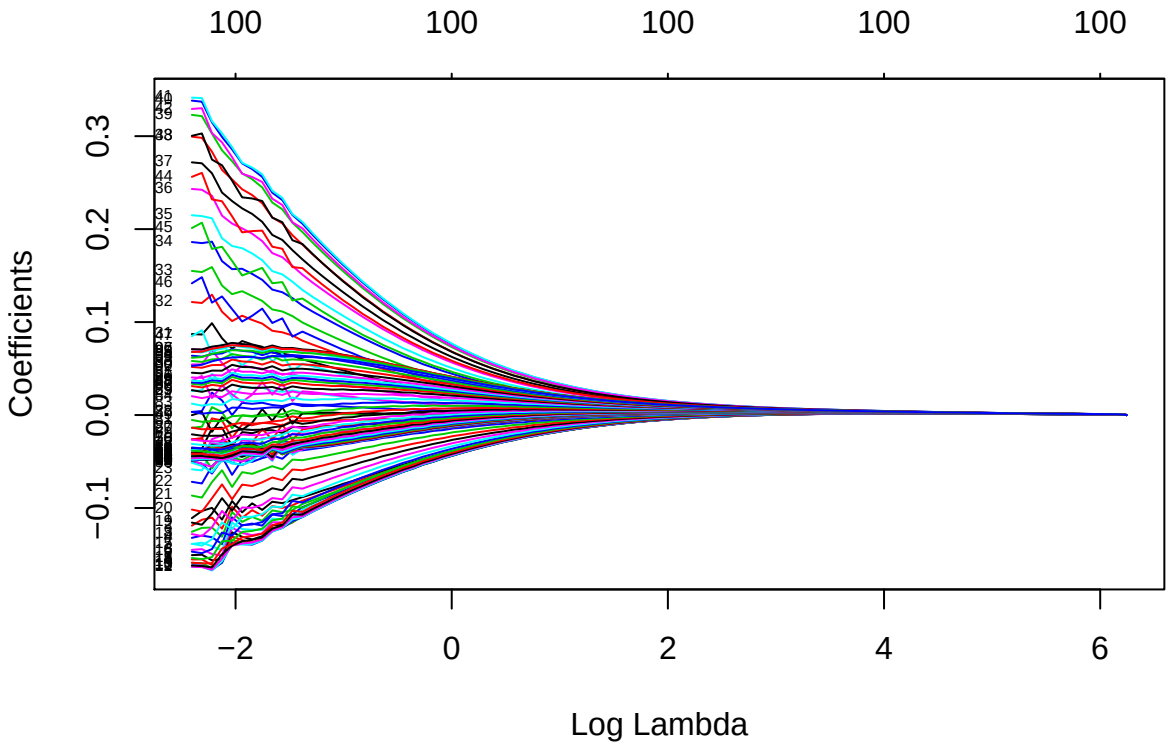
```
library(MASS)

tecator_2<-tecator[2:102]
model_fit <- lm(Fat~.,data=data.frame(tecator_2))

s <- stepAIC(model_fit, direction="both", trace = FALSE)

length(s$coefficients)
```

```
## [1] 64
```

Number of variables selected: 64

4.5

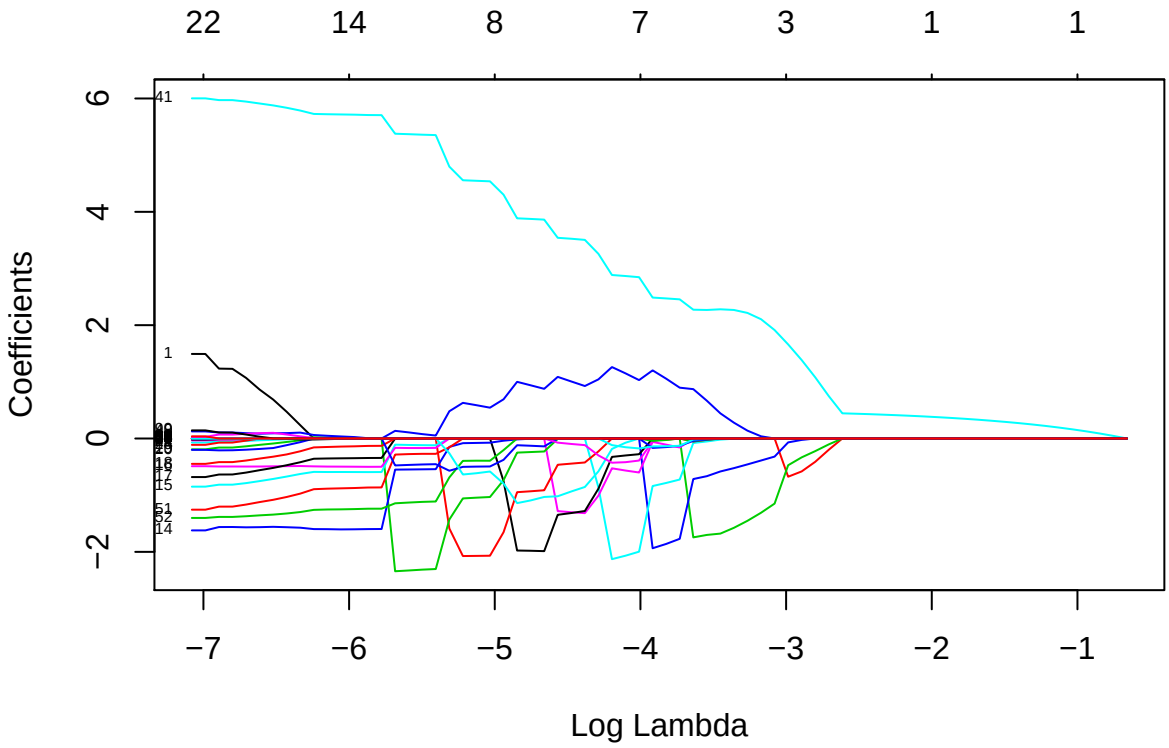
```
library(readxl)
library(glmnet)

covariates <- scale(tecator_2[,1:100])
response <- scale(tecator_2[,101])
model_4.51 <- glmnet(as.matrix(covariates), response, alpha=0,family="gaussian")
plot(model_4.51, xvar="lambda", label=TRUE)
```



The coefficients are converging to zero as the value of Log Lambda is increasing. The coefficients almost converge to 0 when the value of log lambda becomes 4.

4.6

```
model_4.61=glmnet(as.matrix(covariates), response, alpha=1,family="gaussian")
plot(model_4.61, xvar="lambda", label=TRUE)
```



It can be seen from the above plots that the coefficients of certain variables which do not account for the change in dependent variable are reduced gradually to 0 but the variables are not ignored completely when ridge regression is applied. Whereas in the case of Lasso regression the coefficients of certain variables which do not account for the change in dependent variable are made 0 and completely ignored. There is no gradual decrease in the value of coefficients, the change is drastic.

4.7

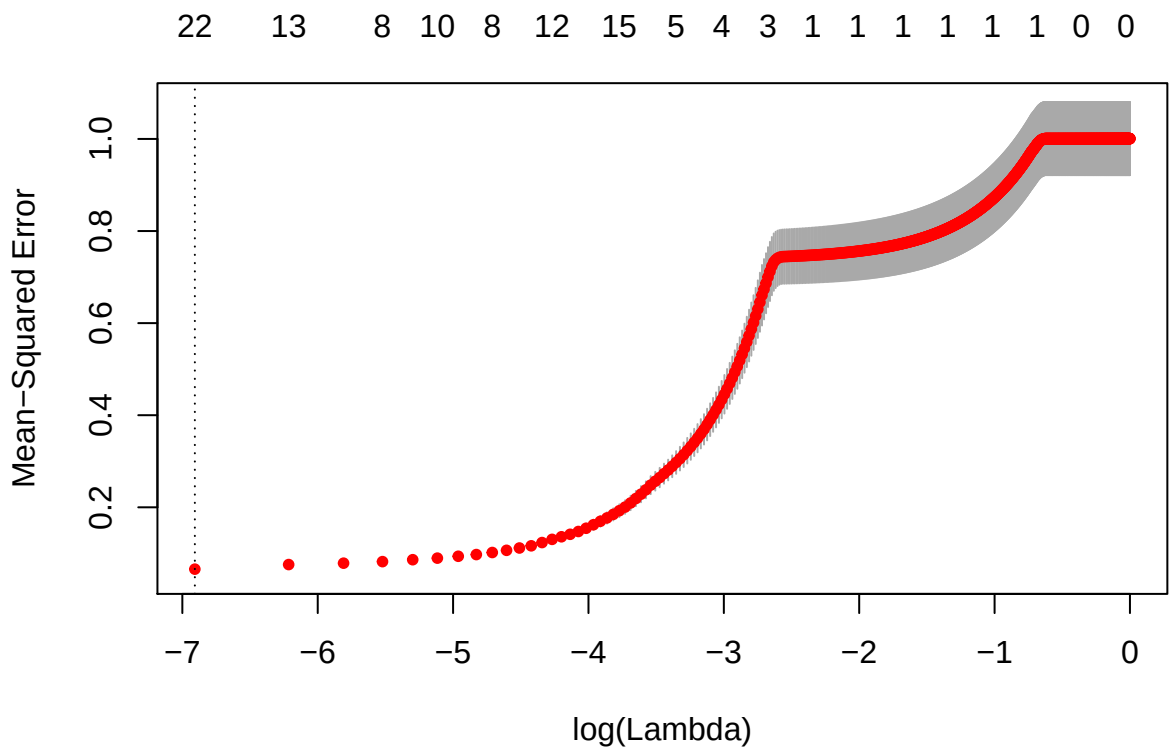
```
model_4.71 <- cv.glmnet(as.matrix(covariates), response, alpha=1,family="gaussian",lambda=seq(0,1,0.001
y <- tecator_2[,101]
ynew=predict(model_4.71, newx=as.matrix(tecator_2[, 1:100]), type="response")

#Coefficient of determination
sum((ynew-mean(y))^2)/sum((y-mean(y))^2)
```

```
## [1] NA
```

```
sum((ynew-y)^2)
```

```
## [1] 90141.18
```

```
plot(model_4.71)
```




```
coef(model_4.71, s="lambda.min")
```

```
## 101 x 1 sparse Matrix of class "dgCMatrix"
##                          1
## (Intercept)  3.423729e-15
## Channel1     2.389874e+00
## Channel2     1.988995e-06
## Channel3     1.193300e-06
## Channel4    -3.161787e-06
## Channel5    -5.893738e-06
## Channel6     3.484921e-01
## Channel7    -7.283755e-02
## Channel8    -6.794308e-02
## Channel9     1.422145e-01
## Channel10   -7.818637e-02
## Channel11   -9.791948e-03
## Channel12   -6.749591e-02
## Channel13    8.343128e-03
## Channel14   -1.159804e-02
## Channel15   -1.105447e+00
## Channel16   -1.617016e-01
## Channel17   -9.932520e-01
## Channel18   -7.298657e-01
```

```
## Channel19   -2.581660e+00
## Channel20   -1.919222e-01
## Channel21   -1.431399e-01
## Channel22   -4.840860e-02
## Channel23   -3.651574e-02
## Channel24   -1.864789e-02
## Channel25    7.570411e-06
## Channel26    1.491368e-05
## Channel27    1.536979e-05
## Channel28    1.409084e-05
## Channel29    1.370188e-05
## Channel30    6.678097e-06
## Channel31   -6.166308e-06
## Channel32   -1.319432e-05
## Channel33   -8.494726e-06
## Channel34    1.139730e-05
## Channel35    3.363313e-05
## Channel36    5.203122e-05
## Channel37    6.456039e-05
## Channel38    6.255091e-05
## Channel39    4.143816e-05
## Channel40   -2.162560e-01
## Channel41    6.734801e+00
## Channel42   -1.260505e-05
## Channel43   -1.435354e-05
## Channel44   -2.446549e-05
## Channel45   -3.925484e-05
## Channel46   -5.191104e-05
## Channel47   -5.284406e-05
## Channel48   -2.916115e-05
## Channel49    1.534122e-01
## Channel50   -1.216789e-01
## Channel51   -1.601908e+00
## Channel52   -1.271089e+00
## Channel53   -7.792935e-02
## Channel54   -1.232931e-01
## Channel55   -8.068550e-02
## Channel56    1.832617e-02
## Channel57    1.809941e-01
## Channel58   -5.985316e-03
## Channel59   -4.819876e-02
## Channel60    6.648840e-02
## Channel61    1.663522e-01
## Channel62   -1.320815e-01
## Channel63   -9.210347e-02
## Channel64   -5.989582e-02
## Channel65   -9.942104e-05
## Channel66   -9.768134e-05
## Channel67   -8.517783e-05
## Channel68   -7.996782e-05
## Channel69   -8.946121e-05
## Channel70   -9.530787e-05
## Channel71   -8.319355e-05
## Channel72   -6.480279e-05
```

```
## Channel73   -6.565899e-05
## Channel74   -7.081182e-05
## Channel75   -5.436526e-05
## Channel76   -2.522736e-05
## Channel77   -1.878956e-05
## Channel78   -1.046285e-05
## Channel79   -4.135561e-06
## Channel80    7.920261e-06
## Channel81    1.224882e-05
## Channel82    1.636592e-05
## Channel83    1.766812e-05
## Channel84    1.797741e-05
## Channel85    2.643758e-05
## Channel86    3.403708e-05
## Channel87    4.460767e-05
## Channel88    5.152972e-05
## Channel89    5.712904e-05
## Channel90    5.687547e-05
## Channel91    5.839429e-05
## Channel92    6.316667e-05
## Channel93    6.450139e-05
## Channel94    6.473916e-05
## Channel95    6.403011e-05
## Channel96   -1.350519e-02
## Channel97    1.275633e-01
## Channel98    9.754263e-02
## Channel99    6.085942e-02
## Channel100   1.261776e-01
```

```
model_4.71$lambda.min
```

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

It can be seen from the plot that there is an increase in MSE with the increase in log(lambda) i.e. lambda itself. For the MSE to be minimum the lambda should be minimum which is 0 in this case. This corresponds to the maximum number of available variables for predictions tobe used, which is 100 variables in this case.

4.8

In step 4 64 variables were chosen, in step 7 all the variables were chosen to be predictors.