Introduction to Machine Learning - Lab 1

Ugurcan Lacin

11/3/2017

Assignment 1

In order to classify emails whether they are spam or not, K-NN Algorithm is used in this report. For that purpose, data set is imported into R and divided it into training and test sets (50% 50%). Thus, training set can be used to train K-NN implementation, then test set will be applied to test the consistency of the model.

Part 1

```r
# Read data
library(gdata)
data <- read.xls("/home/ugur/git/ML_Lab1/spambase.xlsx")

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

K-NN the algorithm simply looks at the distance to the neighbors around the specified k point. Cosine similarity is used to as distance measure here, which for two vectors X and Y defined as:

$$c(X,Y) = \frac{X^TY}{\sqrt{\sum_i X_i^2}\sqrt{\sum_i Y_i^2}}$$

After that, this measurement is used to calculate distance by subtracting it from 1.

$$d(X,Y) = 1 - c(X,Y)$$

After finding the distances, the list is sorted and get first k th elements, and calculate probability by using them. As a result of this function, the probability returns that email is junk or not.

Part 2

```
knearest=function(data,k,newdata) {
  n1=dim(data)[1]
  n2=dim(newdata)[1]
  p=dim(data)[2]
  Prob=numeric(n2)
  X=as.matrix(data[,-p])
  Xn=as.matrix(newdata[-p])

  X_hat=X/matrix(sqrt(rowSums(X^2)), nrow=n1, ncol=p-1)
  Y_hat=Xn/matrix(sqrt(rowSums(Xn^2)), nrow=n1, ncol=p-1)

  C = X_hat %*% t(Y_hat)

  D = 1 - C

  for (i in 1:n2 ){
    neighbor <- order(D[,i])[1:k]
    spam <- as.vector(data$Spam[neighbor])
    Prob[i] <- sum(spam)/k
  }
  return(Prob)
}
```

## Part 3 confusionMatrix is a function that calculates confusion matrix and missclassification rate for given parameters, such as, test data, probabilities and decision rate. The function decides email is spam or now by comparing decision rate which is 0.5 as default. If it is greater than 0.5, it is spam. Then, it produces confusion matrix for us to see results as a table. And it calculates missclassification rate which means

$$missclassification = 1 - correct$$

```
confusionMatrix <- function(test = test,Prob,rate = 0.5){
  svar <- data.frame(Pred = 0, Prob)
  svar$Pred[Prob > rate] <- 1
  conf_matr <- table(test$Spam, svar$Pred)
  print("Confusion Matrix:")
  print(conf_matr)
  print(paste("Misclassification rate is ",(conf_matr[2,1] + conf_matr[1,2])/(n/2)))
}
```

Knearest function will be used to classify training data with K = 5, which is default value for the function.

```r
resultKnearest5 <- knearest(train,5,test)
confusionMatrix(test,resultKnearest5)
```

```
## [1] "Confusion Matrix:"
##
##       0   1
##   0 695 242
##   1 193 240
## [1] "Misclassification rate is  0.317518248175182"
```

# Part 4 Now, K = 1 is used to evaluate from another approach.

```r
resultKnearest1 <- knearest(train,1,test)
confusionMatrix(test,resultKnearest1)
```

```
## [1] "Confusion Matrix:"
##
##       0   1
##   0 639 298
##   1 178 255
## [1] "Misclassification rate is  0.347445255474453"
```

As seen, when K value is decreased it increases missclassification rate which means wrong predictions are increased. The following conclusion can be drawn from this that being careful when choosing k. For example, choosing a value of 1 would only take into account the nearest neighbor. However, when the value of 5 is chosen, there is more consistent result because the nearest 5 neighbors are considered. Because it means a better interpretation of neighbors. But increasing this value too much may not give a good result as well.

Part 5

Now, kknn() function is used which is under kknn package, and same steps are done like K=5 and K=1, then results will be compared with previous ones. First kknn package is imported to R.

```r
library(kknn)
```

Then, kknn function is run with same parameters.

```r
kknnPackageResult5 <- kknn(Spam~., train, test, distance = 5)
confusionMatrix(test,fitted(kknnPackageResult5))
```

```
## [1] "Confusion Matrix:"
##
##       0   1
##   0 703 234
##   1 222 211
## [1] "Misclassification rate is  0.332846715328467"
```

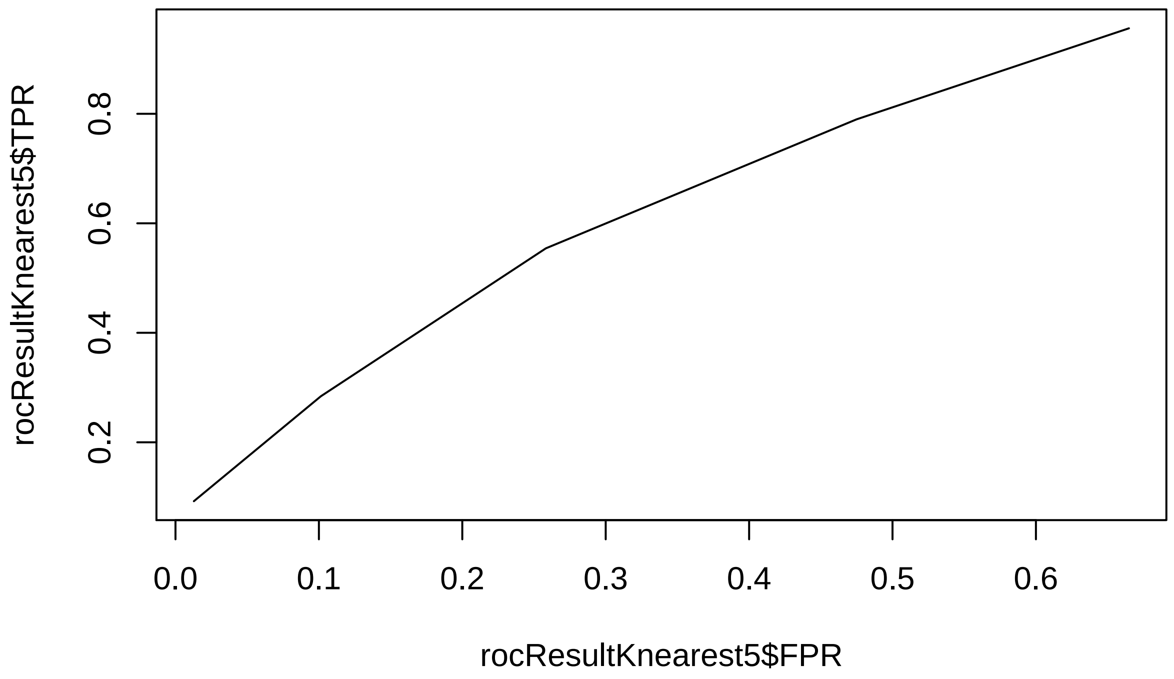
Based on the results received, the evaluation which was made earlier are confirmed here. Because when the distance value is given as 5, a better result is obtained.

Part 6

Now, There will be comparison for knearest() and kknn() functions with K=5 and will classify the test data by using following principle:

$$\hat{Y} = 1 \quad if \quad p(Y=1|X) > \pi, \quad otherwise \quad \hat{Y} = 0, \quad where \quad \pi = 0.05, 0.1, 0.15, ..., 0.95.$$

```
ROC=function(Y, Yfit, p){
  m=length(p)
  TPR=numeric(m)
  FPR=numeric(m)
  for(i in 1:m){
    t=table(Yfit>p[i], Y)
    TPR[i]=t[2,2]/(t[1,2]+t[2,2])
    FPR[i]=t[2,1]/(t[2,1]+t[1,1])
  }
  return (list(TPR=TPR,FPR=FPR))
}

rates <- seq(0.05, 0.95, by=0.05)
rocResultKnearest5 <- ROC(test$Spam,resultKnearest5,rates)

plot(x=rocResultKnearest5$FPR,y=rocResultKnearest5$TPR,type="l")
```

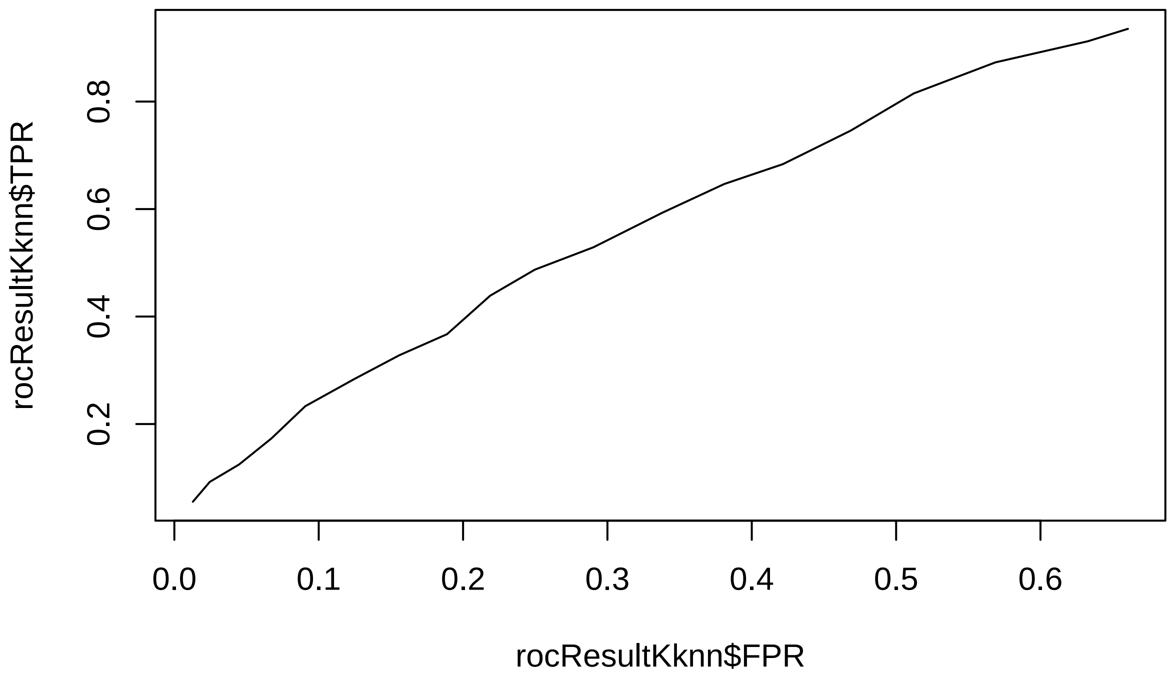


```
SpecificityKnearest <- 1 - rocResultKnearest5$FPR
SensitivityKnearest <- rocResultKnearest5$TPR
```

```
## [1] "Specificity"

##  [1] 0.3351121 0.3351121 0.3351121 0.5250800 0.5250800 0.5250800 0.5250800
##  [8] 0.7417289 0.7417289 0.7417289 0.7417289 0.8986126 0.8986126 0.8986126
## [15] 0.8986126 0.9871932 0.9871932 0.9871932 0.9871932

## [1] "Sensitivity"

##  [1] 0.95612009 0.95612009 0.95612009 0.78983834 0.78983834 0.78983834
##  [7] 0.78983834 0.55427252 0.55427252 0.55427252 0.55427252 0.28406467
## [13] 0.28406467 0.28406467 0.28406467 0.09237875 0.09237875 0.09237875
## [19] 0.09237875
```

First knearest() function is considered, and looked at kknn package result

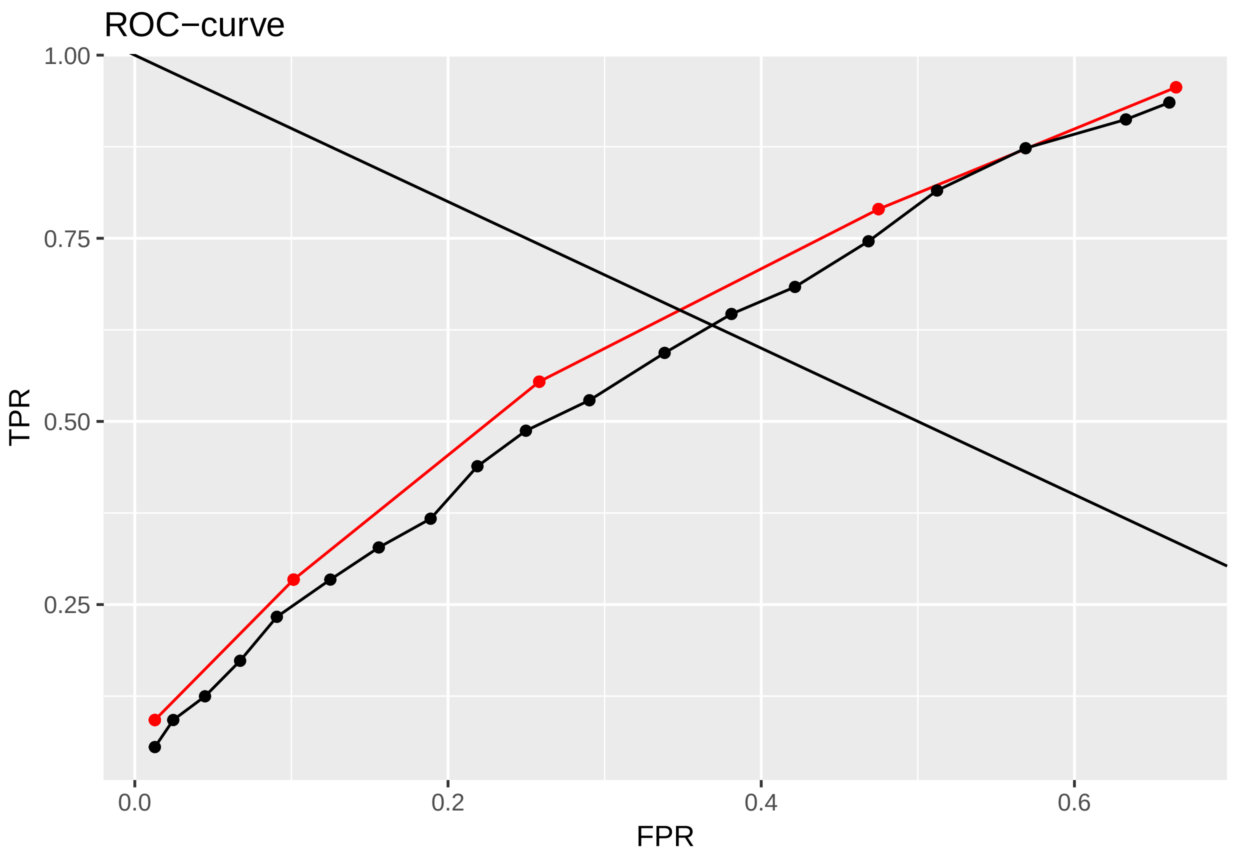
```
rocResultKknn <- ROC(test$Spam,fitted(kknnPackageResult5),rates)
plot(x=rocResultKknn$FPR,y=rocResultKknn$TPR,type="l")
```



```
SpecificityKknn <- 1 - rocResultKknn$FPR
SensitivityKknn <- rocResultKknn$TPR
```

```
## [1] "Specificity"

##  [1] 0.3393810 0.3671291 0.4311633 0.4877268 0.5314835 0.5784418 0.6189968
##  [8] 0.6616862 0.7097118 0.7502668 0.7812166 0.8110993 0.8441836 0.8751334
## [15] 0.9092850 0.9327641 0.9551761 0.9754536 0.9871932

## [1] "Sensitivity"

##  [1] 0.93533487 0.91224018 0.87297921 0.81524249 0.74595843 0.68360277
##  [7] 0.64665127 0.59353349 0.52886836 0.48729792 0.43879908 0.36720554
## [13] 0.32794457 0.28406467 0.23325635 0.17321016 0.12471132 0.09237875
## [19] 0.05542725
```

```
dataFrameKnearest <- data.frame(FPR = rocResultKnearest5$FPR,TPR = rocResultKnearest5$TPR)
dataFrameKknn <- data.frame(FPR = rocResultKknn$FPR,TPR = rocResultKknn$TPR)

library(ggplot2)
ggplot() +
  geom_point(data = dataFrameKnearest, aes(x = FPR, y = TPR), color = "red") +
  geom_line(data = dataFrameKnearest, aes(x = FPR, y = TPR), color = "red") +
  geom_point(data = dataFrameKknn, aes(x = FPR, y = TPR)) +
  geom_line(data = dataFrameKknn, aes(x = FPR, y = TPR)) +
  geom_abline(slope = -1, intercept = 1) +
  xlab("FPR") +
  ylab("TPR") +
  ggtitle("ROC-curve")
```



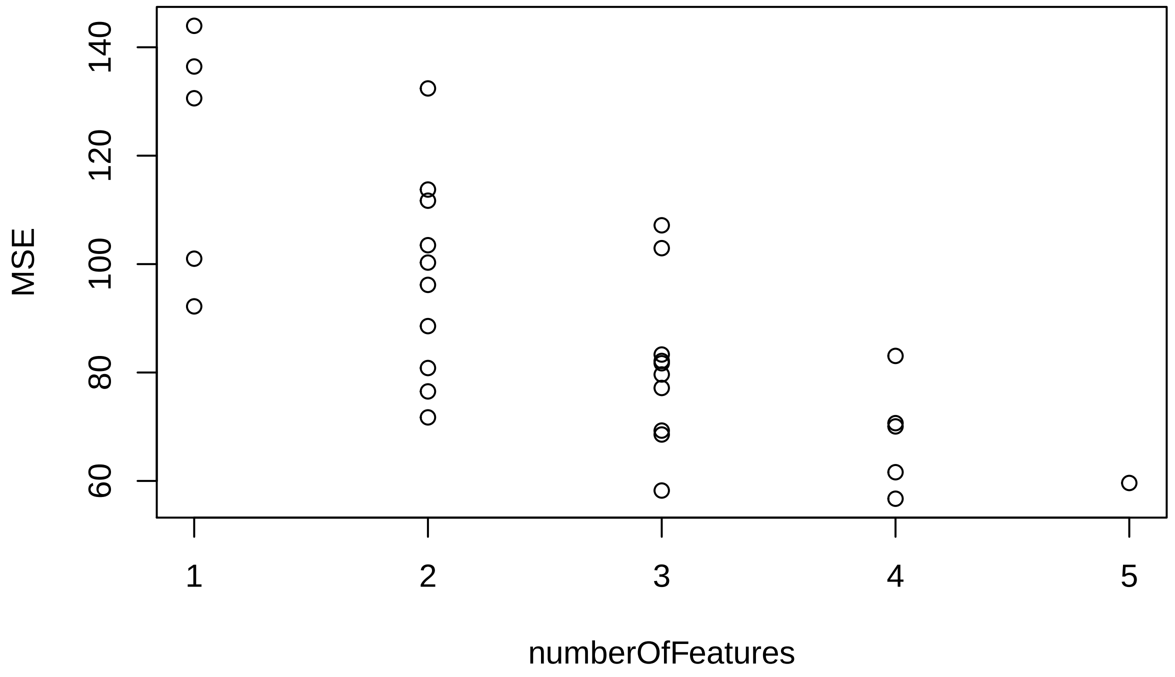
The following conclusion can be drawn from this, knearest() is better than kknn package. The reason can be explained for this by analyzing the graph. It can be seen that the knearest function, indicated by the red line, has more space than the package of the kknn under the curve. For ROC-curve analysis, high space under the curve means better model.

Assignment 3

mylin functions basically calculates linear model for given X and Y matrices and find beta values. After that, it multiply beta and test X matrix and return it.

Part 1 and 2

```
#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)
}

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

  #we assume 5 features.

  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

            for (k in 1:Nfolds) {
              startIndice <- (k * sF) - sF + 1
              endIndice <- 0
              if (k == Nfolds) {
                mod <- nrow(X1) %% Nfolds
                endIndice <- (k * sF) + mod
              } else{
                endIndice <- k * sF
              }
              selectedFeatures <- which(model == 1)
              Xvalidate <- as.matrix(X1[,selectedFeatures])
              Xvalidate <- as.matrix(Xvalidate[startIndice:endIndice,])
              Yvalidate <- as.matrix(Y1[startIndice:endIndice])

              Xtrain <- X1[-c(startIndice:endIndice), selectedFeatures]
              Ytrain <- Y1[-c(startIndice:endIndice)]
              Ypred <- mylin(X = Xtrain, Y = Ytrain, Xpred = Xvalidate)

              SSE = SSE + sum((Ypred - Yvalidate) ^ 2)
            }
            curr = curr + 1
            MSE[curr] = SSE / n
            Nfeat[curr] = sum(model)
            Features[[curr]] = model

          }
  numberOfFeatures <-lapply(Features,FUN = function(x) {return(length(which(x == 1)))})
  plot(x = numberOfFeatures, y = MSE)
  i = which.min(MSE)
  Features=Features[[i]]
  Features2 <- which(Features == 1)
  return(list(CV=MSE[i],
              Features=colnames(X1)[Features2],
              Features = Features))
}

res <- myCV(as.matrix(swiss[, 2:6]), swiss[[1]], 5)
```



```
print(res)
```

```
## $CV
## [1] 56.72245
```

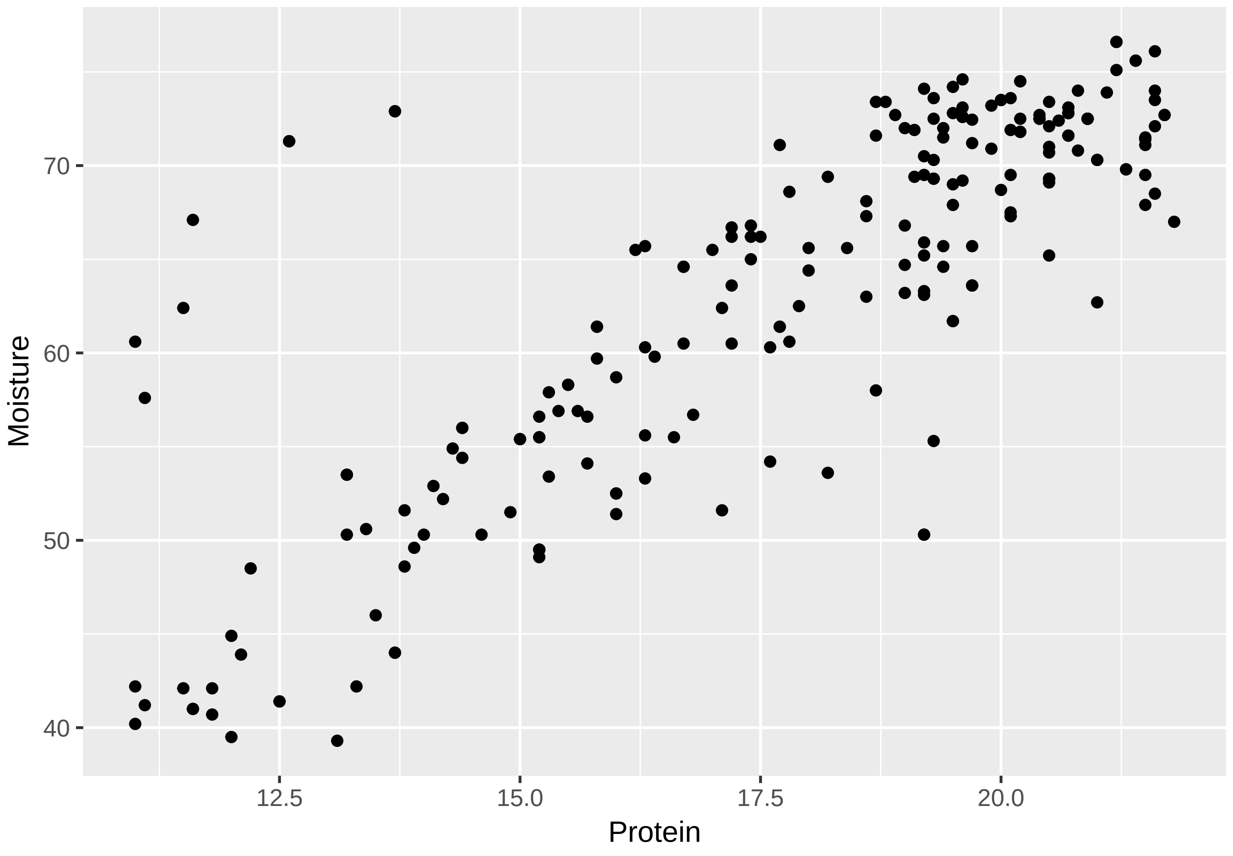
```
##
## $Features
## [1] "Agriculture"      "Education"        "Catholic"
## [4] "Infant.Mortality"
##
## $Features
## [1] 1 0 1 1 1
```

In conclusion, A graph can be seen as X = numberOfFeatures and Y = MSE. According to graph, Following explanation can be said, the MSE tends to decrease as the number of features increases. But that does not mean that the highest number of features gives the lowest MSE rate. The results can be considered to see which features bring the best model. According to this result, It is not possible to say the Examination feature influences the success of the model well.

Assignment 4

Part 1

```r
library(gdata)
data <- read.xls("/home/ugur/git/ML_Lab1/tecator.xlsx")

library(ggplot2)
ggplot(data,aes(x=Protein,y=Moisture)) + geom_point()
```



According to graph, linear model can be applied to describe this data even though there are some outliers.

Part 2

```r
model <- function(i, data){
    m <- lm(Moisture ~ poly(Protein, i, raw = TRUE), data)
    return(m)
}
summary(model(6, data))
```

```
##
## Call:
## lm(formula = Moisture ~ poly(Protein, i, raw = TRUE), data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -17.6554  -3.1664   0.4363   2.9123  21.7608
##
## Coefficients:
##                                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)                   -1.022e+03  1.380e+04  -0.074    0.941
## poly(Protein, i, raw = TRUE)1  4.581e+02  5.279e+03   0.087    0.931
## poly(Protein, i, raw = TRUE)2 -8.008e+01  8.324e+02  -0.096    0.923
## poly(Protein, i, raw = TRUE)3  7.314e+00  6.930e+01   0.106    0.916
## poly(Protein, i, raw = TRUE)4 -3.691e-01  3.214e+00  -0.115    0.909
## poly(Protein, i, raw = TRUE)5  9.811e-03  7.873e-02   0.125    0.901
## poly(Protein, i, raw = TRUE)6 -1.076e-04  7.963e-04  -0.135    0.893
##
## Residual standard error: 5.752 on 208 degrees of freedom
## Multiple R-squared:  0.6721, Adjusted R-squared:  0.6627
## F-statistic: 71.06 on 6 and 208 DF,  p-value: < 2.2e-16
```

MSE is a suitable way of comparing the models in this case because all models have the same loss function. MSE is computed from average SSE of each model. So, it captures variance and bias. By helping of MSE, variance can be followed and interpretation can be made for the model's favor.

Part 3

Data is divided into training and test sets as 50% 50%

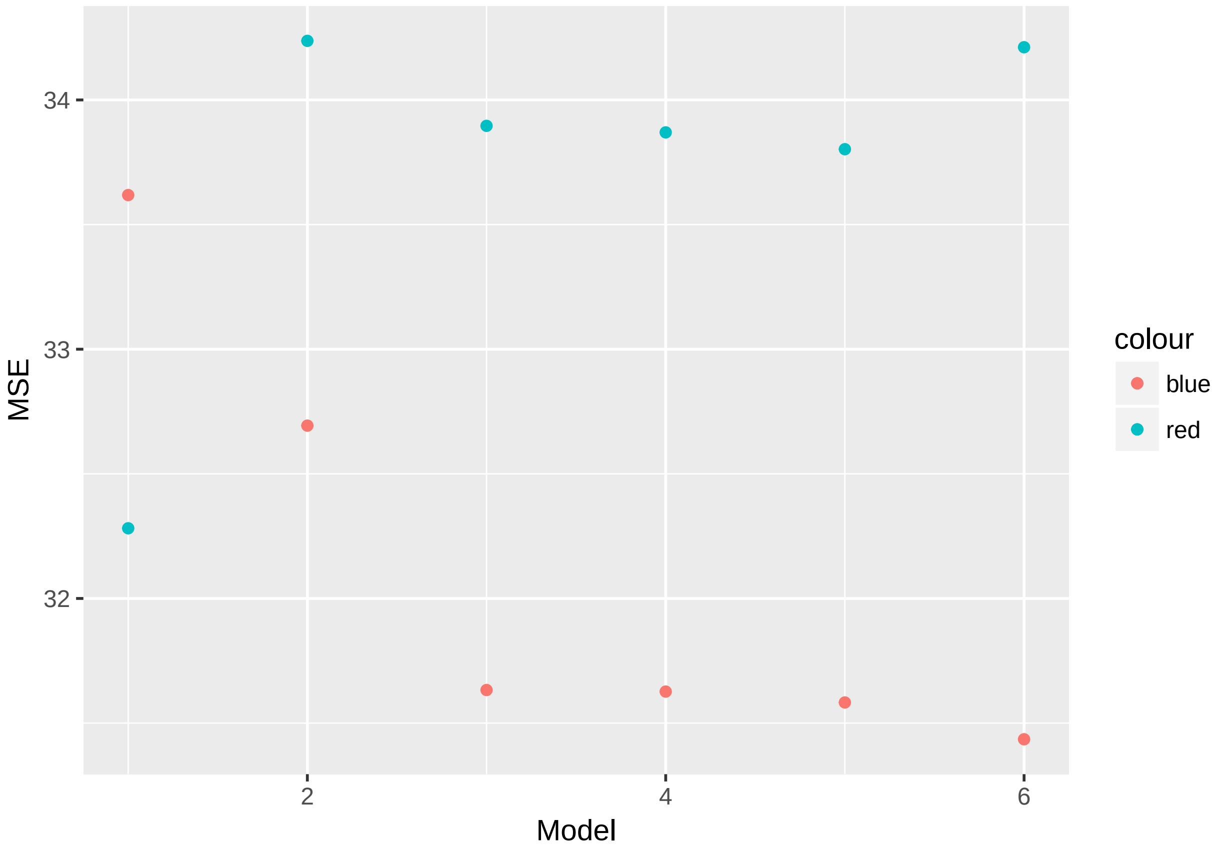
```r
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data[id,]
test=data[-id,]
```

This function takes train and test sets. It calculates MSE for both data sets by using model function from 1 to 6 as i value. After calculation, it returns results.

```r
analyzeMSE <- function(train, test){
  output <- vector()
  for(i in 1:6){
    m <- model(i, train)
    pred <- predict(m, test)
    MSE_train <- mean(residuals(m)^2)
    MSE_test <- mean((pred - test$Moisture)^2)

    output <- rbind(output,c(MSE_train, MSE_test))
  }
  output <- cbind(output, c(1:6))
  colnames(output) <- c("MSE_train", "MSE_test", "model")
  return(output)
}
```

To analyze train and test data, analyzeMSE is applied for that purpose. After getting MSE results, They are plotted to see clearly.

```
MSE <- analyzeMSE(train, test)
dfMSE <- data.frame(MSE)

library(ggplot2)
ggplot(data = dfMSE) +
  geom_point(mapping = aes(x = dfMSE$model, y = dfMSE$MSE_train,colour ='blue')) +
  geom_point(mapping = aes(x = dfMSE$model, y = dfMSE$MSE_test, colour ='red')) +
  ylab("MSE") +
  xlab("Model")
```



According to graph, high bias and low variance are observed in the first model. However, as the iteration increases, the states of these two terms change inversely. In the last iteration, low bias and high variance are observed. This means that the model is actually exposed to overfitting. In that case, first model looks better model among others.

Part 4

```
library(MASS)
df <- data
df4 <- df[, 2:102]
linearModel <- lm(Fat ~., df4)
aic <- stepAIC(linearModel, direction = "both")
```

stepAIC method is applied to perform variable selection. Thus, more important features that will affect the result come to forward.

```
length(aic$coefficients)
```

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

64 features were selected according to the result

Part 5

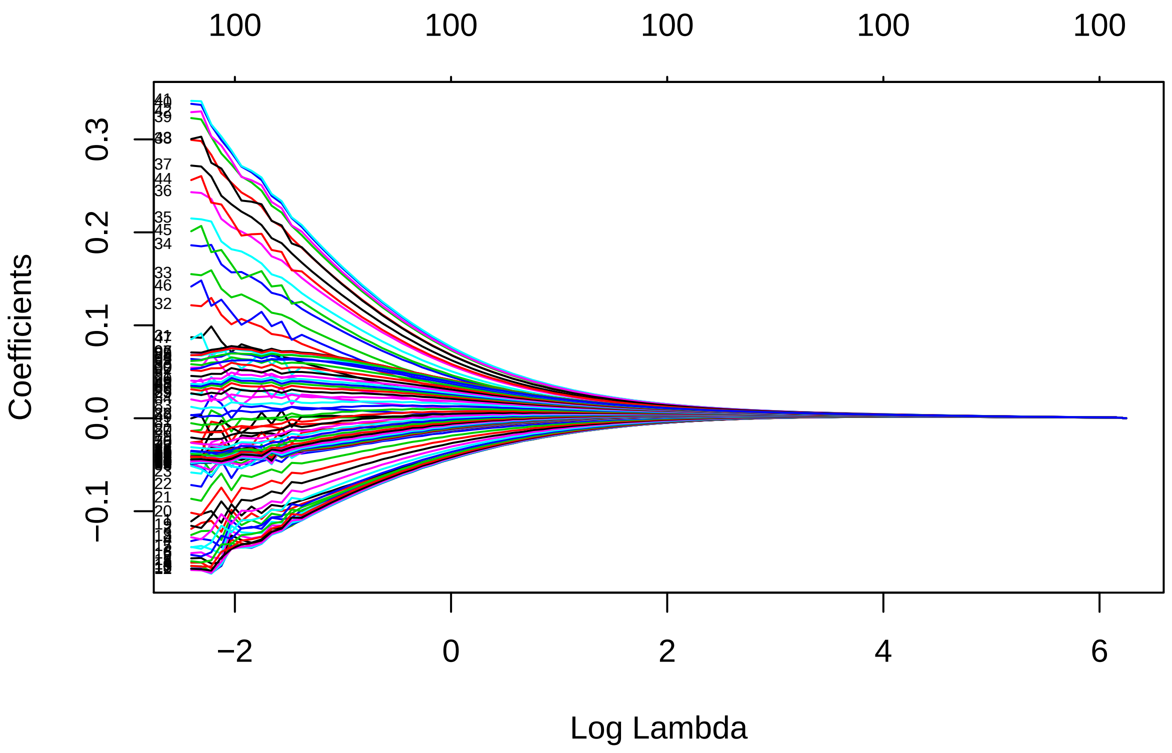
There are still more features to be considered. Unfortunately, in most cases, having more features does not mean that the accuracy of the model will be higher. In addition, overfitting is a significant problem and regularization techniques decreases overfitting possibility.

Regularization is a model that its loss function contains another part to minimize variables' affects over dependent variable.

The ridge regression has following part to minimize the loss function:

$$\lambda * \sum w_j^2$$

```
library(glmnet)
ridge <- glmnet(x = as.matrix(scale(df4[,1:100])), y = as.matrix(scale(df4[, 101])), alpha = 0, family =
plot(ridge, xvar = "lambda", label = TRUE)
```



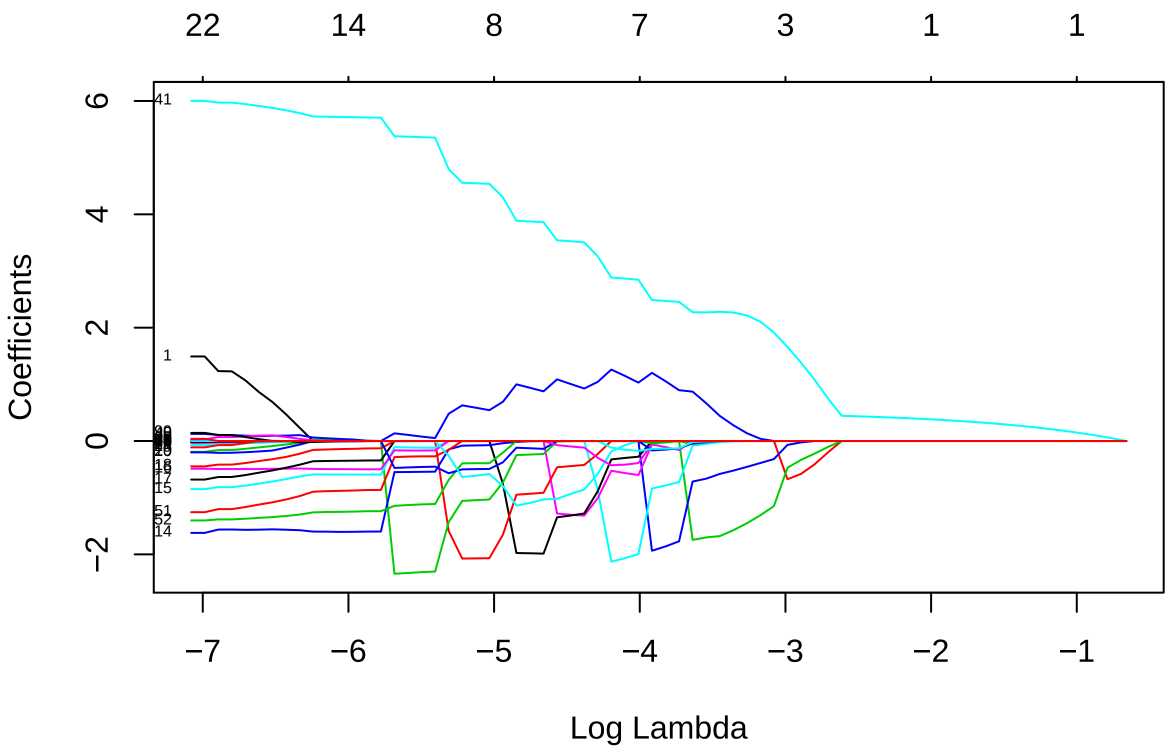
According to graph, it is observed that as the lambda value increases, the weights of the properties approach to zero. This can be explained as follows: Features still affect the result, however, their impact on result is less comparing to before. This is a kind of feature selection result.

Part 6

There are another solution for regularization techniques which is called Lasso. In that method, the following part are used to minimize loss function. Unless ridge regression, Lasso method sets certain features as zero. Thus, if features are not good enough to affect result, they are excluded from the equation by helping of Lasso.

$$\lambda * \sum_{j=1}^{p} |w_j|$$

```
lasso <- glmnet(x = as.matrix(scale(df4[,1:100])), y = as.matrix(scale(df4[, 101])), alpha = 1, family =
plot(lasso, xvar = "lambda", label = TRUE)
```



According the graph, it can be observed that as the lambda value increases, unlike the ridge method, some of the weights are set as zero.

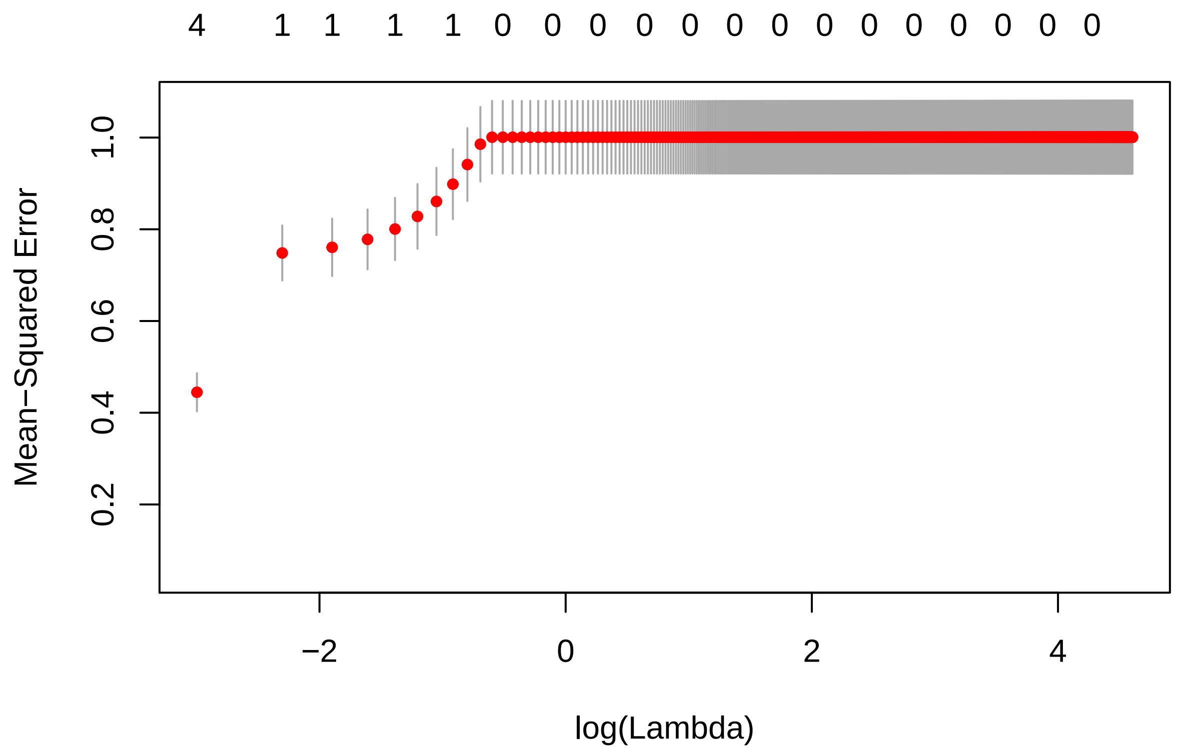
Part 7

```
lamSeq <- seq(0,100,by = 0.05)
CVlasso <- cv.glmnet(x = as.matrix(scale(df4[,1:100])), y = as.matrix(scale(df4[, 101])), alpha = 1, fa
CVlasso$lambda.min
```

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

A lambda sequence is created from 0 to 100 by increasing 0.05. This sequence is used in cross validation for Lasso method. As a result of this step, best lambda value is 0.

```
plot(CVlasso)
```



According to graph, as the logarithm of lambda value increases, mean squared error increases polynominaly, and after specific MSE it stays same.

```r
coef(CVlasso, s = "lambda.min")
```

```
## 101 x 1 sparse Matrix of class "dgCMatrix"
##                         1
## (Intercept) -3.197402e-16
## Channel1     6.051928e-01
## Channel2     9.053259e-01
## Channel3     8.506143e-01
## Channel4     6.121928e-01
## Channel5     3.955075e-01
## Channel6    -3.709255e-01
## Channel7    -4.410980e-01
## Channel8    -5.233351e+00
## Channel9     1.546898e+00
## Channel10    1.344944e+00
## Channel11    1.165100e+00
## Channel12    8.993070e-01
## Channel13    6.149744e-01
## Channel14    2.960214e-01
## Channel15   -3.931042e-02
## Channel16   -2.683172e-01
## Channel17   -6.166294e-01
## Channel18   -1.016090e+00
## Channel19   -1.199002e+00
## Channel20   -1.290918e+00
## Channel21   -1.007067e+00
## Channel22   -5.836188e-01
## Channel23    4.799909e-03
## Channel24    7.069609e-01
## Channel25    1.170280e+00
## Channel26    1.142498e+00
## Channel27    6.231290e-01
## Channel28   -1.370865e-01
## Channel29   -6.537139e-01
## Channel30   -1.181218e+00
## Channel31   -1.613548e+00
## Channel32   -1.788563e+00
## Channel33   -1.619834e+00
## Channel34   -1.200793e-01
## Channel35    2.259240e+00
## Channel36    1.830519e+00
## Channel37    1.472045e+00
## Channel38    4.743348e+00
## Channel39   -7.283795e-01
## Channel40   -3.198187e+00
## Channel41   -4.463598e+00
## Channel42    5.290052e+00
## Channel43    3.001371e+00
## Channel44    2.536639e+00
## Channel45    1.311825e+00
## Channel46   -2.626145e-01
## Channel47   -1.658885e+00
## Channel48   -2.710111e+00
## Channel49   -2.388818e+00
## Channel50   -1.925492e+00
## Channel51   -1.784189e+00
## Channel52   -1.422805e+00
## Channel53   -6.467356e-01
## Channel54    3.937032e-01
## Channel55    1.276204e+00
## Channel56    1.865015e+00
## Channel57    2.028338e+00
## Channel58    1.628950e+00
## Channel59    9.132295e-01
## Channel60    1.024625e+00
## Channel61    5.437716e-01
## Channel62    3.530916e-01
## Channel63   -2.539445e-01
## Channel64   -5.038915e-01
## Channel65   -6.626905e-01
## Channel66   -5.597164e-01
## Channel67   -9.844516e-02
## Channel68    9.740085e-02
## Channel69   -3.225478e-01
## Channel70   -7.660165e-01
## Channel71   -6.192042e-01
## Channel72   -2.609384e-01
## Channel73   -5.638186e-01
## Channel74   -9.206391e-01
## Channel75   -6.735937e-01
## Channel76    4.712080e-02
## Channel77   -9.313536e-02
## Channel78   -2.111476e-01
## Channel79   -2.516879e-01
## Channel80   -1.433479e-01
## Channel81   -2.613789e-01
## Channel82   -1.941777e-01
## Channel83   -1.553290e-01
## Channel84   -7.941321e-02
## Channel85    1.258228e-01
## Channel86    2.064944e-01
## Channel87    3.027398e-01
## Channel88    2.329870e-01
## Channel89    1.199809e-01
## Channel90   -4.295216e-02
## Channel91   -1.808773e-01
## Channel92   -1.008281e-01
## Channel93   -5.438867e-02
## Channel94    6.444539e-02
## Channel95    2.151574e-01
## Channel96    3.143032e-01
## Channel97    4.950684e-01
## Channel98    5.334692e-01
## Channel99    4.791448e-01
## Channel100   2.452897e-01
```

According to coefficients result, it is observed that if lambda is equal to zero it does not affect model improvement. As mentioned before, following equation is used to minimize loss function in Lasso.

$$\lambda * \sum_{j=1}^{p} |w_j|$$

When the equation is taken into consideration, the result gives the zero value when the lambda value equals zero.

$$0 * \sum_{j=1}^{p} |w_j| = 0$$

It can be interpreted as there is no need for feature selection.

Part 8

All three models present different aproaches to fitting same data. The stepAIC selects models based on Akaike Information Criterion and it selected model with 63 variables. While LASSO model with optimal λ selected all possible parameters (100). This indicates that in Lasso regresssion the lowest MSE is reached when no penalty is included ($\lambda = 0$).

Appendix

```r
knitr::opts_chunk$set(echo = TRUE)
# Read data
library(gdata)
```

```r
data <- read.xls("/home/ugur/git/ML_Lab1/spambase.xlsx")

n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data[id,]
test=data[-id,]
knearest=function(data,k,newdata) {
  n1=dim(data)[1]
  n2=dim(newdata)[1]
  p=dim(data)[2]
  Prob=numeric(n2)
  X=as.matrix(data[,-p])
  Xn=as.matrix(newdata[-p])

  X_hat=X/matrix(sqrt(rowSums(X^2)), nrow=n1, ncol=p-1)
  Y_hat=Xn/matrix(sqrt(rowSums(Xn^2)), nrow=n1, ncol=p-1)

  C = X_hat %*% t(Y_hat)

  D = 1 - C

  for (i in 1:n2 ){
    neighbor <- order(D[,i])[1:k]
    spam <- as.vector(data$Spam[neighbor])
    Prob[i] <- sum(spam)/k
  }
  return(Prob)
}

confusionMatrix <- function(test = test,Prob,rate = 0.5){
  svar <- data.frame(Pred = 0, Prob)
  svar$Pred[Prob > rate] <- 1
  conf_matr <- table(test$Spam, svar$Pred)
  print("Confusion Matrix:")
  print(conf_matr)
  print(paste("Misclassification rate is ",(conf_matr[2,1] + conf_matr[1,2])/(n/2)))
}
resultKnearest5 <- knearest(train,5,test)
confusionMatrix(test,resultKnearest5)
resultKnearest1 <- knearest(train,1,test)
confusionMatrix(test,resultKnearest1)
library(kknn)
kknnPackageResult5 <- kknn(Spam~., train, test, distance = 5)
confusionMatrix(test,fitted(kknnPackageResult5))
ROC=function(Y, Yfit, p){
  m=length(p)
  TPR=numeric(m)
  FPR=numeric(m)
  for(i in 1:m){
    t=table(Yfit>p[i], Y)
    TPR[i]=t[2,2]/(t[1,2]+t[2,2])
    FPR[i]=t[2,1]/(t[2,1]+t[1,1])
  }
  return (list(TPR=TPR,FPR=FPR))
}

rates <- seq(0.05, 0.95, by=0.05)
rocResultKnearest5 <- ROC(test$Spam,resultKnearest5,rates)

plot(x=rocResultKnearest5$FPR,y=rocResultKnearest5$TPR,type="l")

SpecificityKnearest <- 1 - rocResultKnearest5$FPR
SensitivityKnearest <- rocResultKnearest5$TPR

print("Specificity")
print(SpecificityKnearest)
print("Sensitivity")
print(SensitivityKnearest)
rocResultKknn <- ROC(test$Spam,fitted(kknnPackageResult5),rates)
plot(x=rocResultKknn$FPR,y=rocResultKknn$TPR,type="l")
SpecificityKknn <- 1 - rocResultKknn$FPR
SensitivityKknn <- rocResultKknn$TPR
print("Specificity")
print(SpecificityKknn)
print("Sensitivity")
print(SensitivityKknn)
dataFrameKnearest <- data.frame(FPR = rocResultKnearest5$FPR,TPR = rocResultKnearest5$TPR)
dataFrameKknn <- data.frame(FPR = rocResultKknn$FPR,TPR = rocResultKknn$TPR)

library(ggplot2)
ggplot() +
  geom_point(data = dataFrameKnearest, aes(x = FPR, y = TPR), color = "red") +
  geom_line(data = dataFrameKnearest, aes(x = FPR, y = TPR), color = "red") +
  geom_point(data = dataFrameKknn, aes(x = FPR, y = TPR)) +
  geom_line(data = dataFrameKknn, aes(x = FPR, y = TPR)) +
  geom_abline(slope = -1, intercept = 1) +
  xlab("FPR") +
  ylab("TPR") +
  ggtitle("ROC-curve")

#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)
}

myCV = function(X, Y, Nfolds) {
  n = length(Y)
  p = ncol(X)
  set.seed(12345)
  ind = sample(n, n)
  X1 = X[ind, ]
```

```r
Y1 = Y[ind]
sF = floor(n / Nfolds)
MSE = numeric(2 ^ p - 1)
Nfeat = numeric(2 ^ p - 1)
Features = list()
curr = 0

#we assume 5 features.

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

          for (k in 1:Nfolds) {
            startIndice <- (k * sF) - sF + 1
            endIndice <- 0
            if (k == Nfolds) {
              mod <- nrow(X1) %% Nfolds
              endIndice <- (k * sF) + mod
            } else{
              endIndice <- k * sF
            }

            selectedFeatures <- which(model == 1)
            Xvalidate <- as.matrix(X1[,selectedFeatures])
            Xvalidate <- as.matrix(Xvalidate[startIndice:endIndice,])
            Yvalidate <- as.matrix(Y1[startIndice:endIndice])

            Xtrain <- X1[-c(startIndice:endIndice), selectedFeatures]
            Ytrain <- Y1[-c(startIndice:endIndice)]
            Ypred <- mylin(X = Xtrain, Y = Ytrain, Xpred = Xvalidate)

            SSE = SSE + sum((Ypred - Yvalidate) ^ 2)
          }
          curr = curr + 1
          MSE[curr] = SSE / n
          Nfeat[curr] = sum(model)
          Features[[curr]] = model

        }
numberOfFeatures <-lapply(Features,FUN = function(x) {return(length(which(x == 1)))})
plot(x = numberOfFeatures, y = MSE)
i = which.min(MSE)
Features=Features[[i]]
Features2 <- which(Features == 1)
return(list(CV=MSE[i],
            Features=colnames(X1)[Features2],
              Features = Features))
}

res <- myCV(as.matrix(swiss[, 2:6]), swiss[[1]], 5)
print(res)
library(gdata)
data <- read.xls("/home/ugur/git/ML_Lab1/tecator.xlsx")

library(ggplot2)
ggplot(data,aes(x=Protein,y=Moisture)) + geom_point()
model <- function(i, data){
    m <- lm(Moisture ~ poly(Protein, i, raw = TRUE), data)
    return(m)
}
summary(model(6, data))
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data[id,]
test=data[-id,]
analyzeMSE <- function(train, test){
  output <- vector()
  for(i in 1:6){
    m <- model(i, train)
    pred <- predict(m, test)
    MSE_train <- mean(residuals(m)^2)
    MSE_test <- mean((pred - test$Moisture)^2)

    output <- rbind(output,c(MSE_train, MSE_test))
  }
  output <- cbind(output, c(1:6))
  colnames(output) <- c("MSE_train", "MSE_test", "model")
  return(output)
}
MSE <- analyzeMSE(train, test)
dfMSE <- data.frame(MSE)

library(ggplot2)
ggplot(data = dfMSE) +
  geom_point(mapping = aes(x = dfMSE$model, y = dfMSE$MSE_train,colour ='blue')) +
  geom_point(mapping = aes(x = dfMSE$model, y = dfMSE$MSE_test, colour ='red')) +
  ylab("MSE") +
  xlab("Model")
library(MASS)
df <- data
df4 <- df[, 2:102]
linearModel <- lm(Fat ~., df4)
aic <- stepAIC(linearModel, direction = "both")
length(aic$coefficients)
library(glmnet)
ridge <- glmnet(x = as.matrix(scale(df4[,1:100])), y = as.matrix(scale(df4[, 101])), alpha = 0, family
plot(ridge, xvar = "lambda", label = TRUE)
lasso <- glmnet(x = as.matrix(scale(df4[,1:100])), y = as.matrix(scale(df4[, 101])), alpha = 1, family
plot(lasso, xvar = "lambda", label = TRUE)
lamSeq <- seq(0,100,by = 0.05)
CVlasso <- cv.glmnet(x = as.matrix(scale(df4[,1:100])), y = as.matrix(scale(df4[, 101])), alpha = 1, fa
CVlasso$lambda.min
plot(CVlasso)
coef(CVlasso, s = "lambda.min")
```