Lab 1 Introduction to Machine Learning

Alessia De Biase

16th November 2017

Assignment 1. Spam classification with nearest neighbors

The **spambase.xlsx** data file containts a total of 2740 email manually classified as Spam and not Spam according to the frequency of various words present in the text of the emails. The aim of this assignment is to use the classification algorithm of K-nearest neighbors to build a model that can be used as spam filter.

The algorithm has been implemented from scratch with the function *knearest(data,k,newdata)* that uses *data* as training data to learn the model and gives as output the predicted class probabilities for *newdata*.

- The first step to build the method is to split the data into training and test sets:

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

- The function uses a distance measure called cosine similarity to find the distance between the elements of the two dataset so the steps *i*,*ii*,*iii* solve the formula

$$c(X,Y) = \frac{X^T Y}{\sqrt{(\sum_i X_i^2)}\sqrt{(\sum_i Y_i^2)}} \tag{1}$$

  and the distance is obtained by $d(X,Y) = 1 - c(X,Y)$. After computing the distance, next step is to find the probabilities to be spam or not counting how many among the k closest elements are spam dividing by the number of k.

```
#2)
#implementing the K-nearest neighbors method with the knearest function

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])

  #i)
  X=X/matrix(sqrt(rowSums(X^2)), nrow=n1, ncol=p-1)

  #ii)
  Xn=Xn/matrix(sqrt(rowSums(Xn^2)), nrow=n2, ncol=p-1)

  #iii)
  C=X%*%t(Xn)
```

```
  #iv) find the distance
  D=1-C

  for (i in 1:n2 ){

    or<-order(D[,i])[1:k]
    Prob[i]<-sum(train[or,49])/k #probability to be spam or not

  }
  return(Prob)
}
```

- k can be choosen by the user and it represents the number of nearest neighbors to evaluate in the classification process. After choosing k the function can be applied to the training and the test set and the *confusion matrix* can be computed. This matrix is useful to understand how well our model classifies, how many mistakes and what kind of misclassification errors it produces. The classification principle used in this first case is: $\hat{Y} = 1$ if $p(Y = 1|X) > 0.5$, otherwise $\hat{Y} = 0$.

When k=5 the confusion matrix for the **testing data** looks like:

	True-NoSpam	True-Spam
Pred-NoSpam	695	193
Pred-Spam	242	240

and the misclassification rate is 32%.

When k=5 the confusion matrix for the **training data** looks like:

	True-NoSpam	True-Spam
Pred-NoSpam	787	119
Pred-Spam	158	306

and the misclassification rate is 20%.

As we may expect the misclassification rate for the training data is lower than the one for the testing data, this is because the model has been created by the training data so we have an overfitting problem.

When k=1 the confusion matrix for the **testing data** looks like:

	True-NoSpam	True-Spam
Pred-NoSpam	639	178
Pred-Spam	298	255

and the misclassification rate is 35%.

When k=1 the confusion matrix for the **training data** looks like:

	True-NoSpam	True-Spam
Pred-NoSpam	939	2
Pred-Spam	6	423

and the misclassification rate is 1%.

The case of k=1 is an extreme case: the elements are classified according to the class of the closest one. The misclassification rate for the testing set is higher than the one from the training set with k=1, the training set with k=5 and the testing set when k=5. When k=1 the classification of the training set has the lowest misclassification rate but the one of the test set has the highest rate, this is because the effect of the noise on the classification is too large.

- After using the *knearest()* function to classify the data, the *kknn()* function from the *kknn* library has been used.

In this case the confusion matrix for the **testing data** is the following:

	True-NoSpam	True-Spam
Pred-NoSpam	640	177
Pred-Spam	297	256

and the misclassification rate is 35%.

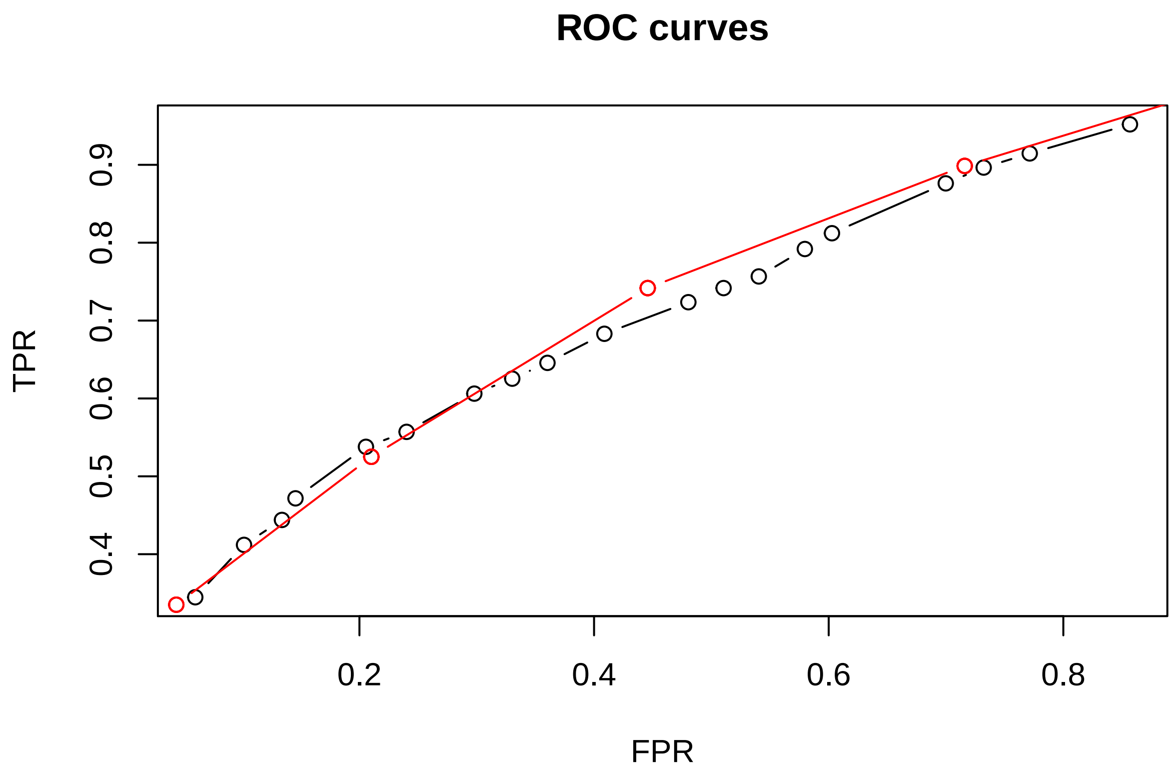
The misclassification rates of each method are reported in the following tab:

K=5 with knearest()	K=1 with knearest()	K=5 with kknn()
31.75%	34.74%	34.6%

What we can notice from the tabs is that the best classifier is the one built from scratch with k=5. Its misclassification error is lower than others, less Spam emails are classified and way less True NoSpam email have been predicted as Spam. The function made from scratch uses the cosine similarity as distance function, the *kknn()* function, instead, uses a random selection of distance. The worst classifier is the one with k=1 and this is not surprising for what we stated before. Between the *kknn()* and the *knearest()* function the one which classifies better is the second one because it uses a better distance function. We will have the proof also by the area of the ROC curve in the following step.

- Now both the *knearest()* and the *kknn()* functions are used but the classification principle is different: $\hat{Y} = 1$ if $p(Y = 1|X) > \pi$, otherwise $\hat{Y} = 0$ where $\pi = 0.05, 0.1, 0, 15, ...0.9, 0.95$.

To better compare the two methods the ROC curves have been plotted. The ROC curve is created by plotting the true positive rate against the false positive rate. The best classifier has the highest area under its ROC curve compared to others. The graph below represents the two ROC curves from the two different methods we used before: the red one is the one of the K-nearest neighbors made from scratch and the black one is from the R function *kknn()*. The red line is the best ROC curve even if they look very similar. In the graph we can notice that while the *kknn()* function has as output all different values for TPR and FPR, *knearest()* has many repeated values of probabilities that's why we can see less connecting points.


ROC curves
TPR
0.4
0.5
0.6
0.7
0.8
0.9
0.2
0.4
0.6
0.8
FPR

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

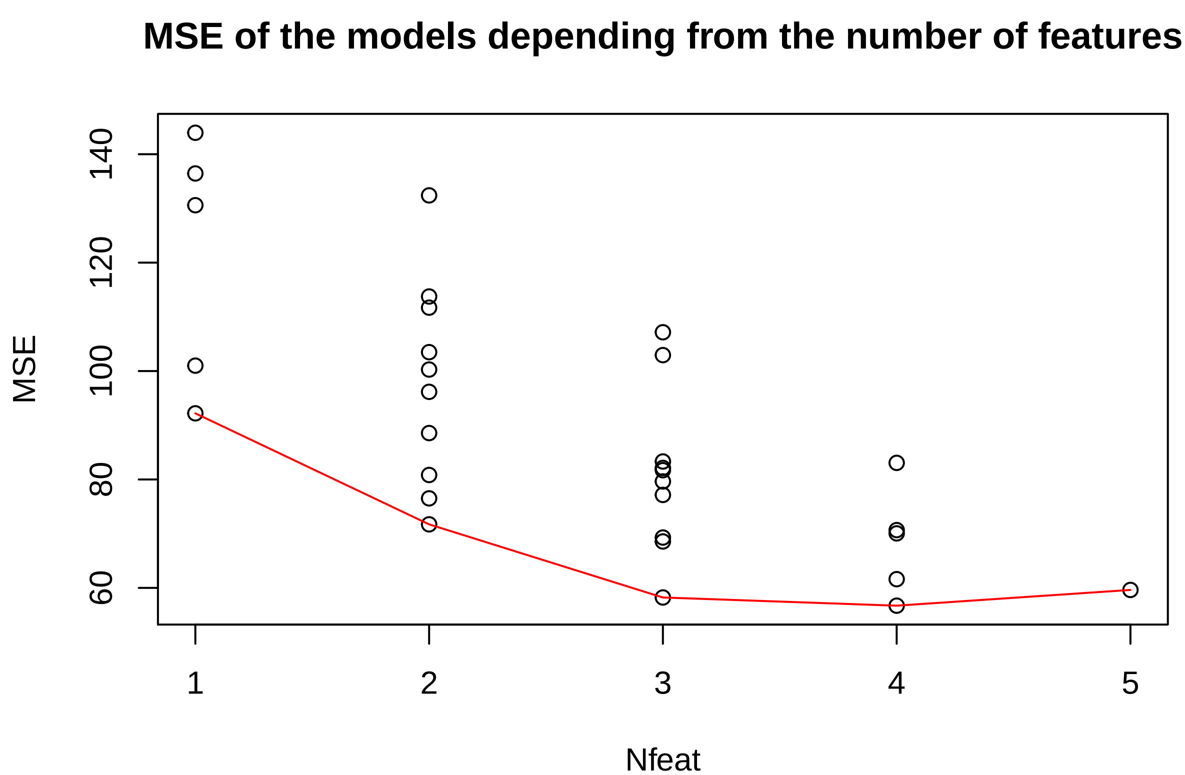
In this assignment it's been implemented a function that performs feature selection (best subset selection) in linear regression by using k-fold cross-validation.

In the function we assume to have 5 features, we want to find the best possible subset of features so the model with the lowest MSE. For each model to find the SSE we compute k-fold cross validation. The cross validation method randomly partion the data in k folds of equal dimension (if not specified the dimension of each group), performs the analysis on one subset and validates the analysis on the other subset. To reduce the variance this kind of process is repeated k times and the validation results are averaged.

```
#cross validation with feature selection
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) #number of observations in a folder
  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() #case of vector with all 0
            SSE=0

            for (k in 1:Nfolds){ #for each fold

              index_k<-(1:sF)+((k-1)*sF) #testing subset

              if(k==Nfolds) index_k<-(sF*4+1):n #last subset

              Xp<-X1[-index_k,which(model==1)] #training X
              Yp<-Y1[-index_k] #training X

              X_test<-X1[index_k,which(model==1)] #testing X
              Y_test<-Y1[index_k] #testing Y

              SSE=SSE+sum((Y_test-mylin(Xp,Yp,X_test))^2)

            }

            curr=curr+1 #number of model we are currently
```

```
            MSE[curr]=SSE/n
            Nfeat[curr]=sum(model)
            Features[[curr]]=model

          }

  min_MSE<-vector()
  for(j in 1:Nfolds){
    ind <-which(Nfeat==j)
    min_MSE[j]<-min(MSE[ind])
  }

  plot(Nfeat,MSE,main="MSE of the models depending from the number of features")
  lines(min_MSE,col="red")

  i=which.min(MSE)

  return(list(CV=MSE[i], Features=Features[[i]])) #return the model with lowest MSE
}
```

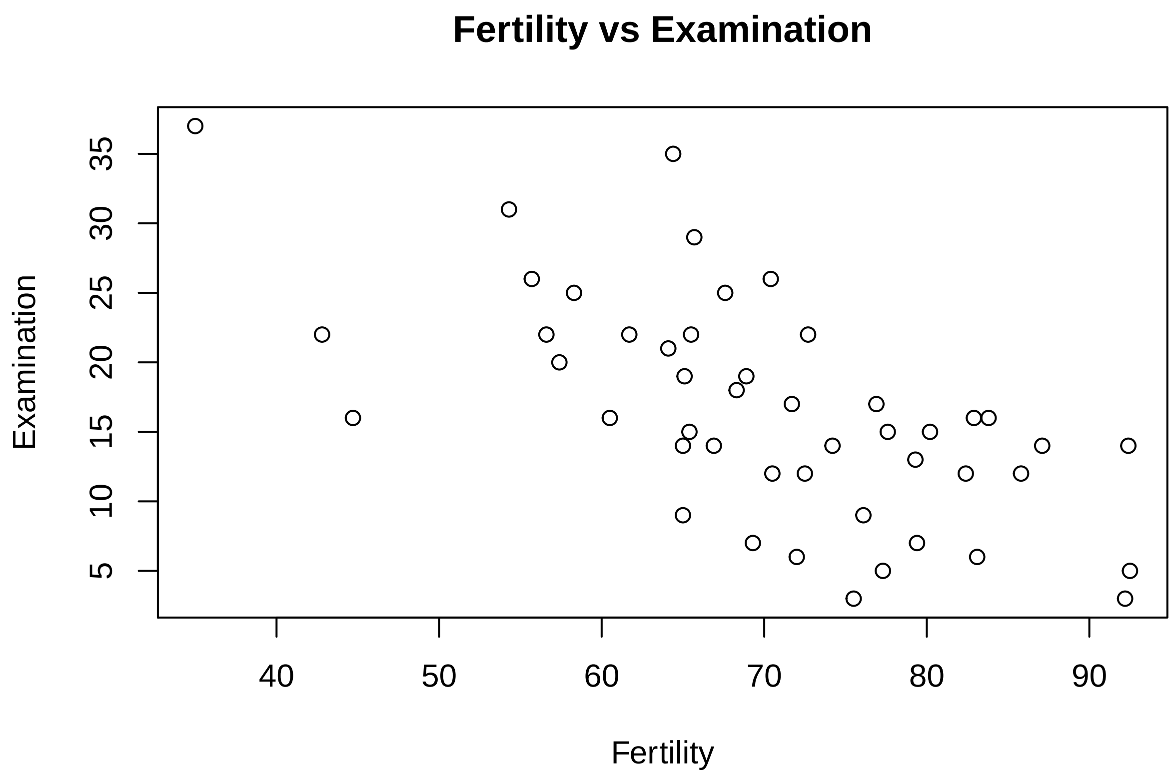
After implementing the function we test it on the set *swiss* where *fertility* is the Y and all other variables are columns of the matrix X. Nfolds is choosen to be equal to 5. The plot obtained shows that the model with lowest MSE is the one with four features: *Agricolture, Education, Catholic and Infant. Mortality.*




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

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

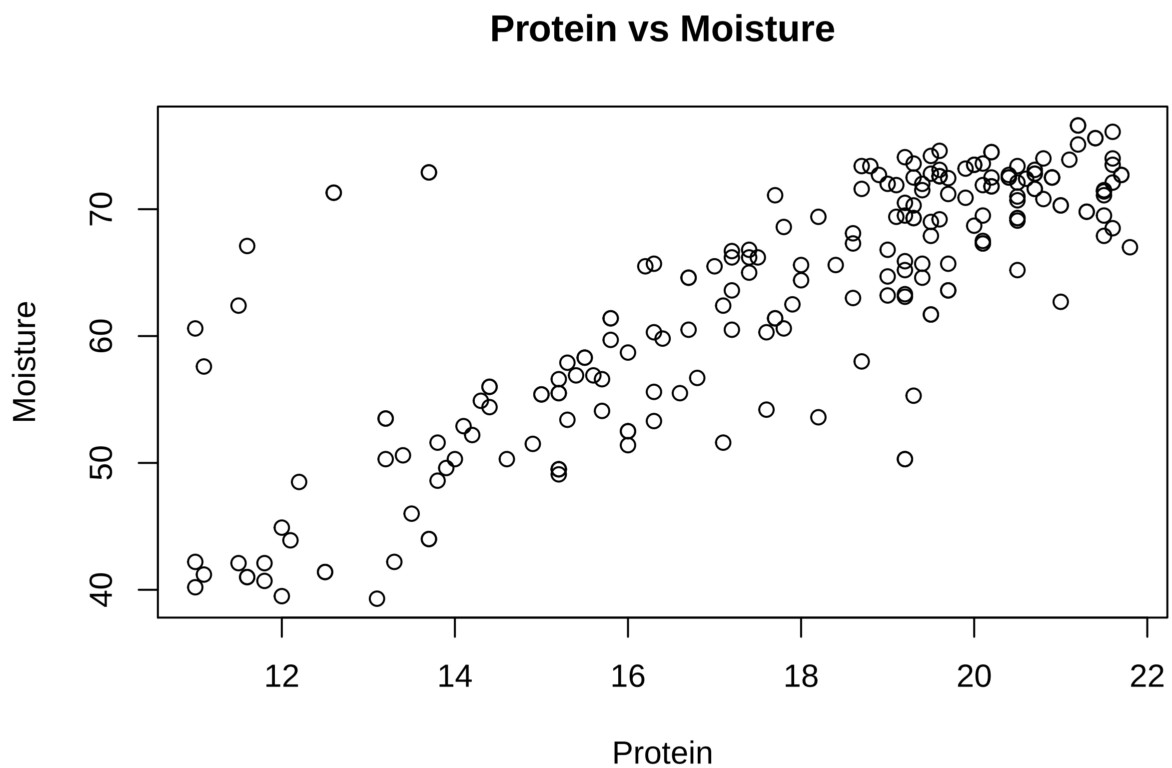
It's reasonable not to consider the feature Examination in the model infact, as we can see from the following plot, the target variable *Fertility* doesn't really seam to have a kind of relation with *Examination*:



Assignment 4. Linear regression and regularization

The **tecator.xlsx** data file containts the results of study aimed to investigate whether a near infrared absorbance spectrum can be used to predict the far content of samples of meat.

1. In the first part we only focus on two variables from the dataset: *Moisture* and *Protein*. The following plot describes how those variables are related.



As we can see they have a strong correlation that at first sight may look like linear even though some outliers appear in the top-left corner.

2. M_i is the model in which *Moisture* is normally distributed and P is *Protein*. The model is represented as it follows:

$$M_1 = \beta_{01} + \beta_{11}P$$

$$M_2 = \beta_{02} + \beta_{12}P + \beta_{22}P^2$$

$$M_3 = \beta_{03} + \beta_{13}P + \beta_{23}P^2 + \beta_{33}P^3$$

...

We know that the expected *Moisture* is a polynomial function of *Protein* so the probabilistic model that describes M_i is

$$M_i \sim N(\mu_i, \sigma_i^2) \tag{2}$$

where

$$\mu_i = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_i \end{pmatrix} \begin{pmatrix} 1 \, \mu(P) \, \mu(P^2) \, \ldots \, \mu(P^i) \end{pmatrix} \tag{3}$$

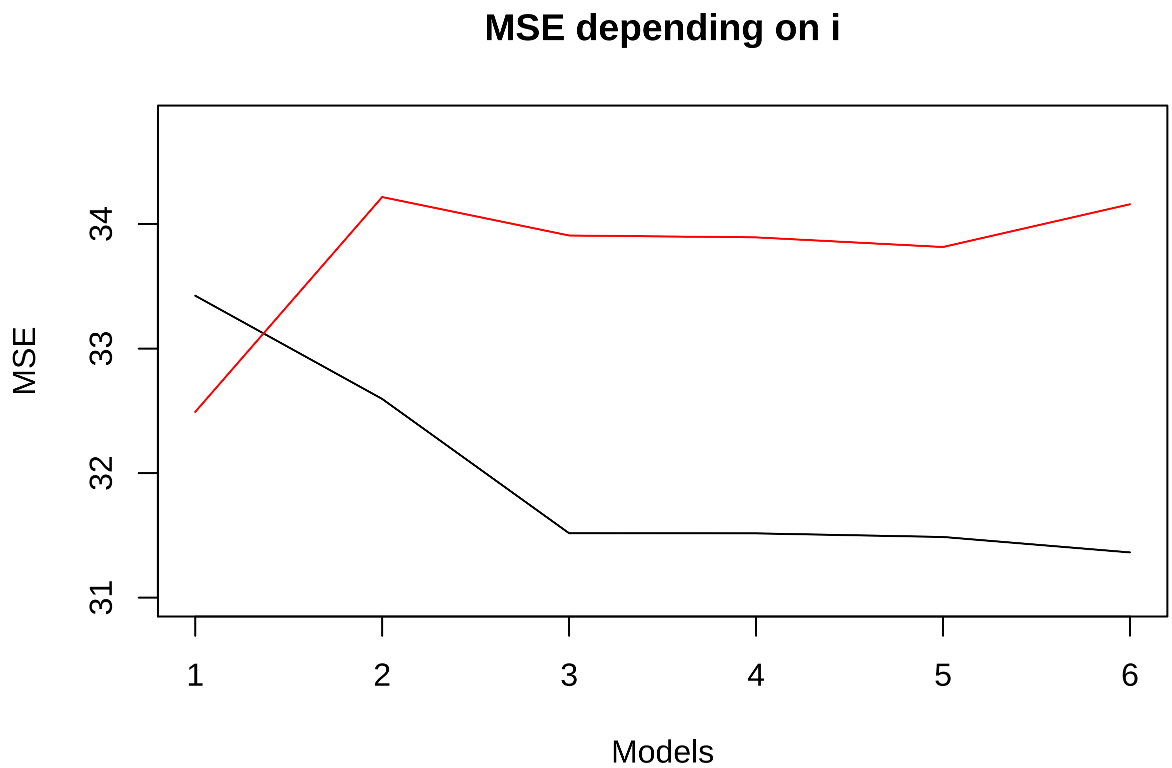
and

$$\sigma_i^2 = \begin{pmatrix} \beta_1^2 \\ \vdots \\ \beta_i^2 \end{pmatrix} \begin{pmatrix} \sigma^2(P) \, \sigma^2(P^2) \, \ldots \, \sigma^2(P^i) \end{pmatrix} \tag{4}$$

3. After identifying the model, we randomly divide the data into training and validation sets and we fit the models M_i with $i = 1 \ldots 6$. For each model we record training and validation MSE and we plot them to show how they depend on i.

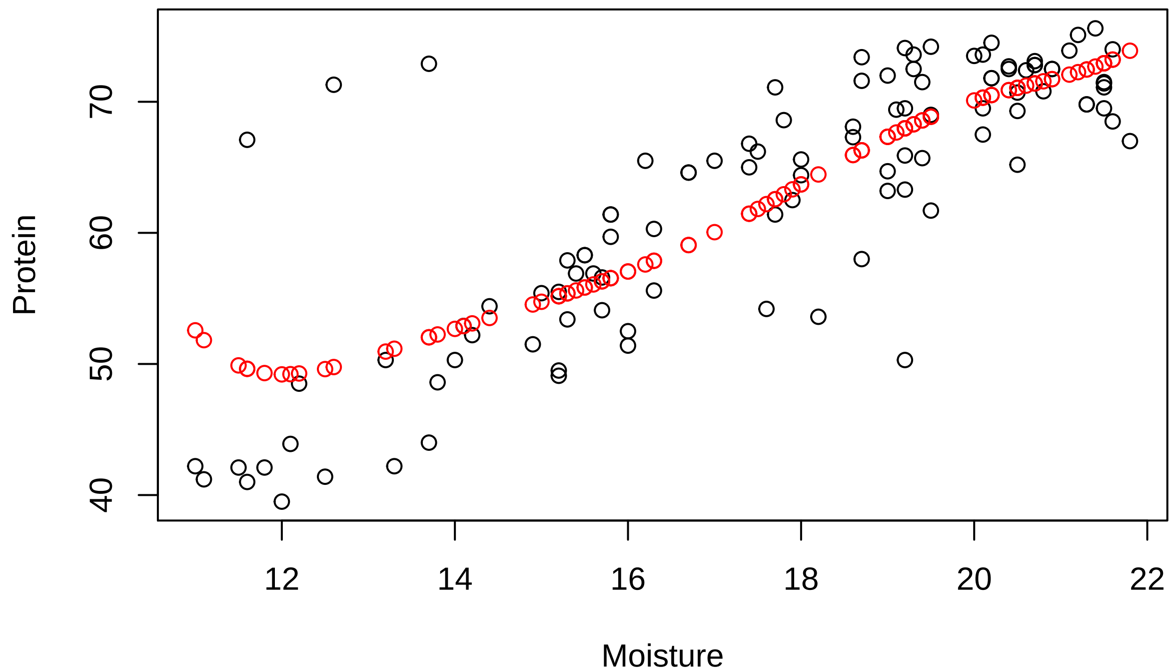
```
#3)

set.seed(12345)
X<-cbind(data1$Protein,data1$Moisture)

index=sample(dim(X)[1],round(dim(X)[1]*.5)) #50-50%

train<-X[index,]
test<-X[-index,]
```

```
#fit the model and find MSE of the training and of the test set

for(i in 2:7){
  fit<-lm(Yp ~. , data=phi_train[,1:i])
  Y_hat=predict(fit,newdata = phi_test[,1:i])
  MSE_tr[i-1]<-sum(resid(fit)^2)/n_train
  MSE_te[i-1]<-sum((Y_hat-phi_test[,1])^2)/n_test
}

#MSE plot

plot(1:6,MSE_tr,type="l",main="MSE depending on i",ylim=c(31,34.8),ylab="MSE",xlab="Models")
lines(MSE_te,col="red")
```



In the plot the **black line** represents the MSE values from the training set and the **red line** represents the MSE values from the test set. According to the graph the best model for the test set is the linear one, fitting the training set, instead, the best model seams to be the more complex one. The black lines shows how increasing the number of features we reduce the bias, we reduce the MSE but we have a problem of overfitting and the variance increases. The red line represents the MSE of the test set when i increases. How we can see the M_6 model is not the best model for this subset, the best seams to be the linear model, the simplest one. The bias-variance tradeoff underlines how hard it is to choose a model that both well fit the training and the test set accurately.

The following plot represents the more complex model(M_6 in red) fitted on the test set:



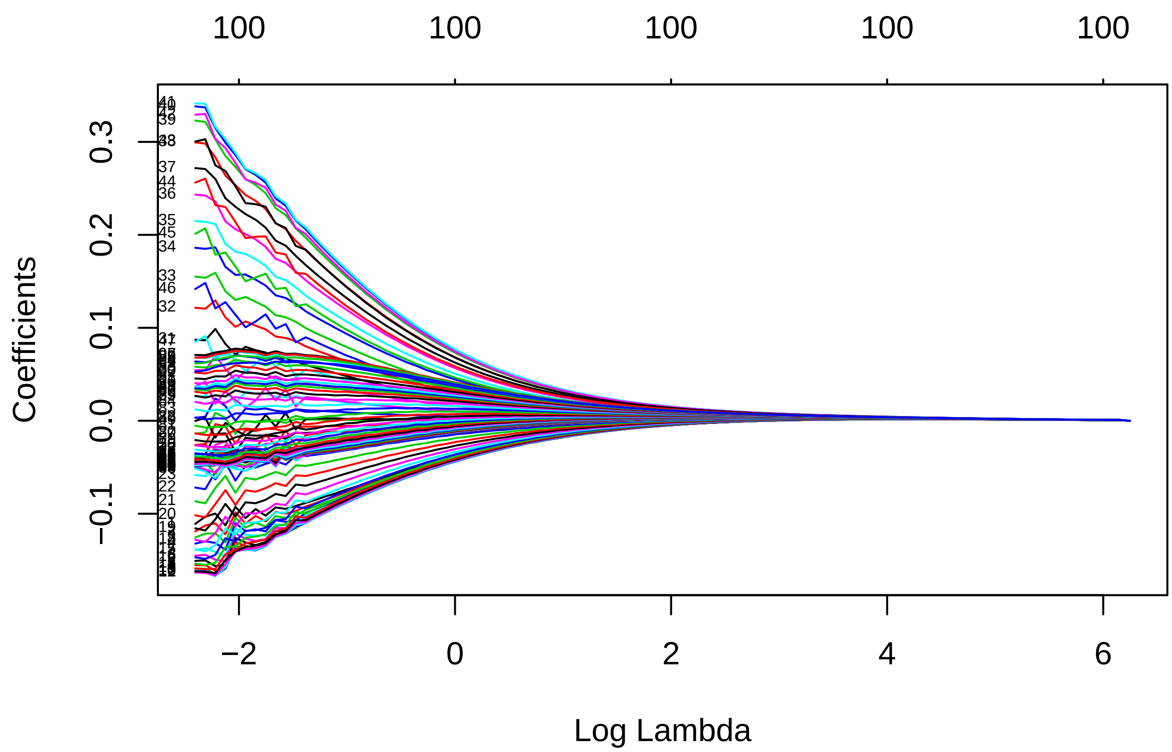
4. The dataset also contains other variables such as *Fat* and 100 *Channels* spectrum of absorbance records. In this second part of the assignment the new response variable is *Fat* and *Channel1-Channel100* are the new predictors. We use the *stepAIC()* function on the linear regression model obtained from the new response variable and the new predictors to compute variables selection. The best model selected has 63 features and it's the Final model of the following output.

```
## Stepwise Model Path
## Analysis of Deviance Table
##
## Initial Model:
## Y_new ~ Channel1 + Channel2 + Channel3 + Channel4 + Channel5 +
##     Channel6 + Channel7 + Channel8 + Channel9 + Channel10 + Channel11 +
##     Channel12 + Channel13 + Channel14 + Channel15 + Channel16 +
##     Channel17 + Channel18 + Channel19 + Channel20 + Channel21 +
##     Channel22 + Channel23 + Channel24 + Channel25 + Channel26 +
##     Channel27 + Channel28 + Channel29 + Channel30 + Channel31 +
##     Channel32 + Channel33 + Channel34 + Channel35 + Channel36 +
##     Channel37 + Channel38 + Channel39 + Channel40 + Channel41 +
##     Channel42 + Channel43 + Channel44 + Channel45 + Channel46 +
##     Channel47 + Channel48 + Channel49 + Channel50 + Channel51 +
##     Channel52 + Channel53 + Channel54 + Channel55 + Channel56 +
##     Channel57 + Channel58 + Channel59 + Channel60 + Channel61 +
##     Channel62 + Channel63 + Channel64 + Channel65 + Channel66 +
##     Channel67 + Channel68 + Channel69 + Channel70 + Channel71 +
##     Channel72 + Channel73 + Channel74 + Channel75 + Channel76 +
##     Channel77 + Channel78 + Channel79 + Channel80 + Channel81 +
##     Channel82 + Channel83 + Channel84 + Channel85 + Channel86 +
```

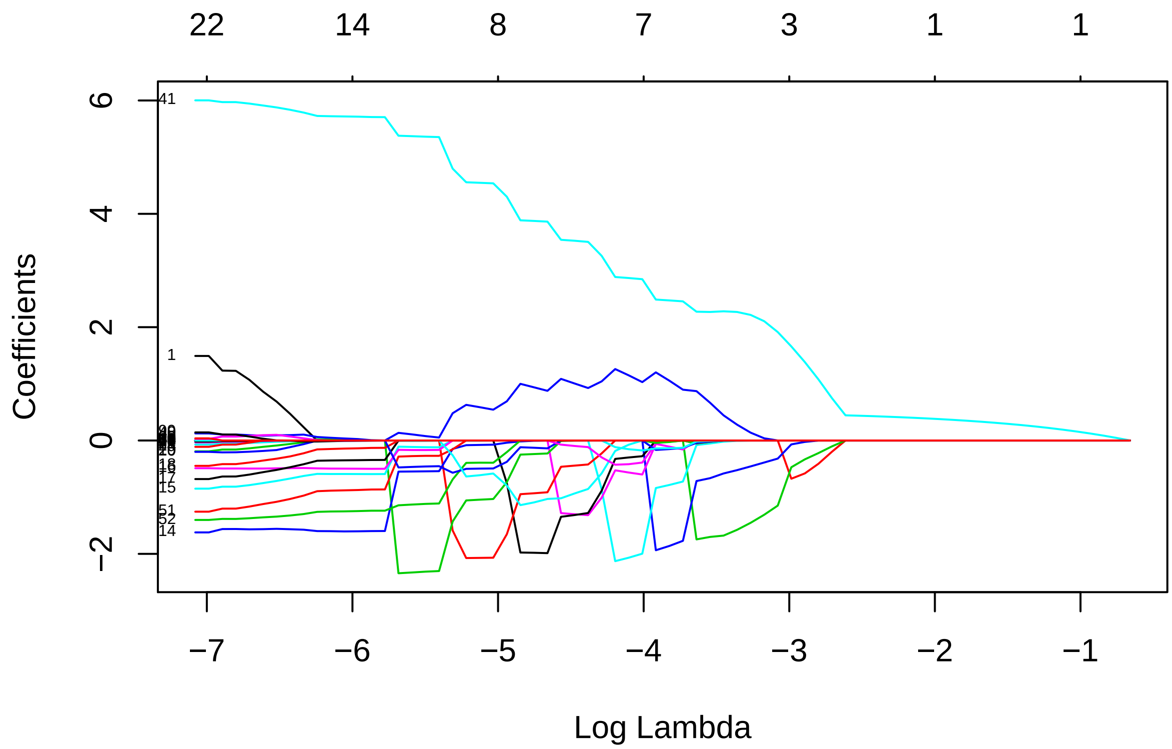
```
##     Channel87 + Channel88 + Channel89 + Channel90 + Channel91 +
##     Channel92 + Channel93 + Channel94 + Channel95 + Channel96 +
##     Channel97 + Channel98 + Channel99 + Channel100
##
## Final Model:
## Y_new ~ Channel1 + Channel2 + Channel4 + Channel5 + Channel7 +
##     Channel8 + Channel11 + Channel12 + Channel13 + Channel14 +
##     Channel15 + Channel17 + Channel19 + Channel20 + Channel22 +
##     Channel24 + Channel25 + Channel26 + Channel28 + Channel29 +
##     Channel30 + Channel32 + Channel34 + Channel36 + Channel37 +
##     Channel39 + Channel40 + Channel41 + Channel42 + Channel45 +
##     Channel46 + Channel47 + Channel48 + Channel50 + Channel51 +
##     Channel52 + Channel54 + Channel55 + Channel56 + Channel59 +
##     Channel60 + Channel61 + Channel63 + Channel64 + Channel65 +
##     Channel67 + Channel68 + Channel69 + Channel71 + Channel73 +
##     Channel74 + Channel78 + Channel79 + Channel80 + Channel81 +
##     Channel84 + Channel85 + Channel87 + Channel88 + Channel92 +
##     Channel94 + Channel98 + Channel99
##
##
##            Step Df     Deviance Resid. Df Resid. Dev       AIC
## 1                                     114   169.8123 151.27203
## 2   - Channel70  1 5.580758e-05       115   169.8124 149.27210
## 3   - Channel89  1 6.338934e-04       116   169.8130 147.27290
## 4   - Channel66  1 4.350148e-04       117   169.8135 145.27345
## 5  - Channel100  1 9.526559e-04       118   169.8144 143.27466
## 6   - Channel57  1 1.512331e-03       119   169.8159 141.27657
## 7   - Channel38  1 4.235150e-03       120   169.8202 139.28193
## 8   - Channel58  1 7.141818e-03       121   169.8273 137.29098
## 9   - Channel53  1 2.509829e-02       122   169.8524 135.32275
## 10   - Channel9  1 3.771904e-02       123   169.8901 133.37049
## 11  - Channel91  1 3.178511e-02       124   169.9219 131.41071
## 12  - Channel77  1 5.501288e-02       125   169.9769 129.48030
## 13  - Channel49  1 9.282875e-02       126   170.0698 127.59769
## 14  - Channel33  1 1.137405e-01       127   170.1835 125.74143
## 15  - Channel96  1 1.838591e-01       128   170.3674 123.97358
## 16  - Channel93  1 1.204802e-01       129   170.4878 122.12557
## 17  - Channel82  1 2.012906e-01       130   170.6891 120.37927
## 18  - Channel86  1 2.608049e-01       131   170.9499 118.70753
## 19  - Channel72  1 3.340581e-01       132   171.2840 117.12725
## 20  - Channel35  1 4.539629e-01       133   171.7380 115.69633
## 21  - Channel43  1 3.667681e-01       134   172.1047 114.15500
## 22  - Channel44  1 3.686336e-01       135   172.4734 112.61502
## 23  - Channel90  1 4.430432e-01       136   172.9164 111.16659
## 24  - Channel83  1 4.636039e-01       137   173.3800 109.74225
## 25   - Channel3  1 4.495464e-01       138   173.8295 108.29899
## 26  - Channel23  1 4.393963e-01       139   174.2689 106.84177
## 27   - Channel6  1 6.745513e-01       140   174.9435 105.67238
## 28  - Channel62  1 6.873639e-01       141   175.6309 104.51547
## 29  - Channel10  1 6.770690e-01       142   176.3079 103.34272
## 30  - Channel18  1 5.551316e-01       143   176.8631 102.01861
## 31  - Channel27  1 8.012085e-01       144   177.6643 100.99038
## 32  - Channel16  1 8.124404e-01       145   178.4767  99.97132
## 33  - Channel21  1 9.726859e-01       146   179.4494  99.13987
```

```
## 34  - Channel95  1 8.809590e-01       147   180.3304  98.19277
## 35  - Channel97  1 6.630855e-01       148   180.9934  96.98189
## 36  - Channel76  1 1.451145e+00       149   182.4446  96.69882
## 37  - Channel75  1 7.506552e-01       150   183.1952  95.58160
## 38  - Channel31  1 1.682931e+00       151   184.8782  95.54769
```

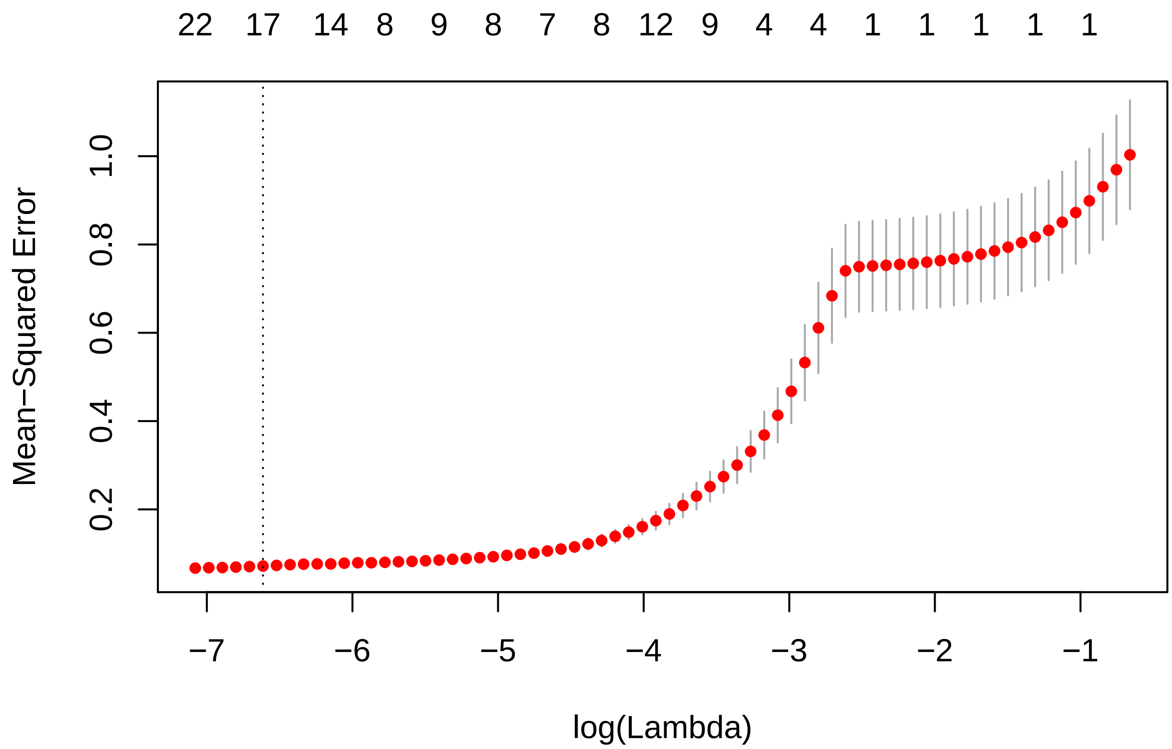
5. We fit now a Ridge Regression model with the same predictor and response variables. The following plot shows how model coefficients depend on the log of the penalty factor λ. The starting point of the graph represent $\lambda = 0$ so the coefficients are not effected by any penalty. As soon as λ grows the penalty increases and the estimated coefficients get closer to zero. A good choice of λ identifies a reasonable trade-off between model fitting and number of variables to include in the model.



6. At this point we fit Lasso regression to our data. The plot that we obtain in this case is totally different from the one of ridge regression, this is because Lasso regression when λ increases makes some coefficients be equal to zero computing variable selection. What we can see in the graph, for exaple, is that the only variable that survives when λ grows is the 41st, most of the others goes to zero very quickly for low values of lambda so they are not so significant to predict the response variable.



7. To find the optimal LASSO model we use cross-validation. The optimal λ found is 0.001341 and the dotted line in the plot below tells us that the best model has 17 non zero coefficients. As we can see from the red dots the MSE increases when lambda increases and the number of features decreases. The CV scores have low standard errors for small lambda and higher for a small number of features and bigger lambda.



The variables selected are the one with non zero coefficient in the following output.

```
## 101 x 1 sparse Matrix of class "dgCMatrix"
##                         1
## (Intercept)  2.464658e-15
## Channel1     8.635295e-01
## Channel2     .
## Channel3     .
## Channel4     .
## Channel5     .
## Channel6     .
## Channel7     .
## Channel8     .
## Channel9     .
## Channel10    .
## Channel11    .
## Channel12    .
## Channel13    .
## Channel14   -1.564397e+00
## Channel15   -7.511834e-01
## Channel16   -4.945293e-01
## Channel17   -5.591383e-01
## Channel18   -3.536799e-01
## Channel19   -1.116970e-01
## Channel20   -1.842917e-01
## Channel21   -3.145644e-02
## Channel22   -8.373254e-03
```

```
## Channel23   -2.553959e-03
## Channel24    .
## Channel25    .
## Channel26    .
## Channel27    .
## Channel28    .
## Channel29    .
## Channel30    .
## Channel31    .
## Channel32    .
## Channel33    .
## Channel34    .
## Channel35    .
## Channel36    .
## Channel37    .
## Channel38    .
## Channel39    .
## Channel40    8.500263e-02
## Channel41    5.911785e+00
## Channel42    .
## Channel43    .
## Channel44    .
## Channel45    .
## Channel46    .
## Channel47    .
## Channel48    .
## Channel49    .
## Channel50    .
## Channel51   -1.121537e+00
## Channel52   -1.356066e+00
## Channel53    .
## Channel54    .
## Channel55    .
## Channel56    .
## Channel57    .
## Channel58    .
## Channel59    .
## Channel60    .
## Channel61    .
## Channel62    .
## Channel63    .
## Channel64    .
## Channel65    .
## Channel66    .
## Channel67    .
## Channel68    .
## Channel69    .
## Channel70    .
## Channel71    .
## Channel72    .
## Channel73    .
## Channel74    .
## Channel75    .
## Channel76    .
```

```
## Channel77    .
## Channel78    .
## Channel79    .
## Channel80    .
## Channel81    .
## Channel82    .
## Channel83    .
## Channel84    .
## Channel85    .
## Channel86    .
## Channel87    .
## Channel88    .
## Channel89    .
## Channel90    .
## Channel91    .
## Channel92    .
## Channel93    .
## Channel94    .
## Channel95    .
## Channel96    .
## Channel97    .
## Channel98    9.427340e-02
## Channel99    3.234302e-02
## Channel100   .
```

8. Comparing the results from 7. with the one from 4. we can see a big difference: the model obtained from the *stepAIC()* function has many more variables compared to the one obtained by *Cross validation and Lasso regression*. Lasso regression is the best solution in this case for variables selection because there are many variables and we want to extract the most relevant to predict the responde so the ones which have a strong correlation with the variable *Fat*. Using too many variables may cause overfitting so we prefer Lasso model with cross validation.

Appendix

```
knitr::opts_chunk$set(echo = TRUE)

#ASSIGNMENT 1.

library(readxl)
data<-read_excel("spambase.xlsx",1)
#1)
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data[id,]
test=data[-id,]
#2)
#implementing the K-nearest neighbors method with the knearest function

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])

  #i)
  X=X/matrix(sqrt(rowSums(X^2)), nrow=n1, ncol=p-1)

  #ii)
  Xn=Xn/matrix(sqrt(rowSums(Xn^2)), nrow=n2, ncol=p-1)

  #iii)
  C=X%*%t(Xn)

  #iv) find the distance
  D=1-C

  for (i in 1:n2 ){

    or<-order(D[,i])[1:k]
    Prob[i]<-sum(train[or,49])/k #probability to be spam or not

  }
  return(Prob)
}
#3)

#when K=5

#TEST DATA: find the Probabilities, the confusion matrix and the misclassification rate when k=5
knear_test <- knearest(train,5,test)
```

```r
Y_hat_5_test<-rep(0,dim(test)[1])
Y_hat_5_test[which(knear_test>.5)]<-1

table_test=table(Y_hat_5_test,test$Spam)
row.names(table_test)<-c("Pred-NoSpam","Pred-Spam")
colnames(table_test)<-c("True-NoSpam","True-Spam")

mis_rate_test<-(table_test[1,2]+table_test[2,1])/sum(table_test)*100

#TRAIN DATA: find the Probabilities, the confusion matrix and the misclassification rate when k=5
knear_train <- knearest(train,5,train)

Y_hat_5_train<-rep(0,dim(train)[1])
Y_hat_5_train[which(knear_train>.5)]<-1

table_train<-table(Y_hat_5_train,train$Spam)
row.names(table_train)<-c("Pred-NoSpam","Pred-Spam")
colnames(table_train)<-c("True-NoSpam","True-Spam")

mis_rate_train<-(table_train[1,2]+table_train[2,1])/sum(table_train)*100

library(knitr)
library(kableExtra)
kable(table_test)
kable(table_train)
#4)

#when k=1

#TEST DATA: find the Probabilities, the confusion matrix and the misclassification rate when k=5
knear_test2 <- knearest(train,1,test)

Y_hat_1_test<-rep(0,dim(test)[1])
Y_hat_1_test[which(knear_test2>.5)]<-1

table_test2=table(Y_hat_1_test,test$Spam)
row.names(table_test2)<-c("Pred-NoSpam","Pred-Spam")
colnames(table_test2)<-c("True-NoSpam","True-Spam")

mis_rate_test2<-(table_test2[1,2]+table_test2[2,1])/sum(table_test2)*100

#TRAIN DATA: find the Probabilities, the confusion matrix and the misclassification rate when k=5
knear_train2 <- knearest(train,1,train)

Y_hat_1_train<-rep(0,dim(train)[1])
Y_hat_1_train[which(knear_train2>.5)]<-1

table_train2=table(Y_hat_1_train,train$Spam)
row.names(table_train2)<-c("Pred-NoSpam","Pred-Spam")
colnames(table_train2)<-c("True-NoSpam","True-Spam")

mis_rate_train2<-(table_train2[1,2]+table_train2[2,1])/sum(table_train2)*100
```

```
kable(table_test2)
kable(table_train2)
#5)
library(kknn)
kknn5<-kknn(Spam ~ .,train, test,k=5)

Y_hat<-kknn5$fitted.values

t=table(Y_hat>.5,test$Spam)
row.names(t)<-c("Pred-NoSpam","Pred-Spam")
colnames(t)<-c("True-NoSpam","True-Spam")

mis_rate_k5=(t[1,2]+t[2,1])/sum(t)*100
kable(t)
misclas<-data.frame(paste(round(mis_rate_test,2),"%",sep=""),paste(round(mis_rate_test2,2),"%",sep=""),p
names(misclas)<-c("K=5 with knearest()","K=1 with knearest()","K=5 with kknn()")
kable(misclas)

#6)
p<-seq(0.05,.95,0.05)

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

roc_mykknn<-ROC(test$Spam,knear_test,p)
roc_kknn<-ROC(test$Spam,fitted(kknn5),p)

plot(roc_kknn$FPR,roc_kknn$TPR,type="b",xlab="FPR",ylab = "TPR",main="ROC curves")
lines(roc_mykknn$FPR,roc_mykknn$TPR, col="red",type="b")
#-------------------------------------#

#ASSIGNMENT 3
data<-swiss
X=data[-1,]
Y=data[1,]
Nfolds=5

#linear regression from scratch:
mylin=function(X,Y, Xpred){
  Xpred1=cbind(1,Xpred)
  X1=cbind(1,X)
  beta=solve(t(X1)%*%X1)%*%t(X1)%*%Y #beta are from the training
  Y_hat=Xpred1%*%beta #Xpred1 is from the testing
  return(Y_hat)
```

```
}
#cross validation with feature selection
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) #number of observations in a folder
  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() #case of vector with all 0
            SSE=0

            for (k in 1:Nfolds){ #for each fold

              index_k<-(1:sF)+((k-1)*sF) #testing subset

              if(k==Nfolds) index_k<-(sF*4+1):n #last subset

              Xp<-X1[-index_k,which(model==1)] #training X
              Yp<-Y1[-index_k] #training X

              X_test<-X1[index_k,which(model==1)] #testing X
              Y_test<-Y1[index_k] #testing Y

              SSE=SSE+sum((Y_test-mylin(Xp,Yp,X_test))^2)

            }

            curr=curr+1 #number of model we are currently
            MSE[curr]=SSE/n
            Nfeat[curr]=sum(model)
            Features[[curr]]=model

          }

  min_MSE<-vector()
  for(j in 1:Nfolds){
    ind <-which(Nfeat==j)
    min_MSE[j]<-min(MSE[ind])
```

```r
  }

  plot(Nfeat,MSE,main="MSE of the models depending from the number of features")
  lines(min_MSE,col="red")

  i=which.min(MSE)

  return(list(CV=MSE[i], Features=Features[[i]])) #return the model with lowest MSE
}

myCV(as.matrix(swiss[,2:6]), swiss[[1]], 5)
plot(swiss$Fertility,swiss$Examination,main="Fertility vs Examination",xlab = "Fertility",ylab="Examina

#------------------------------------#

#ASSIGNMENT 4.

#1)

library(readxl)
data1<-read_excel("tecator.xlsx",1)
plot(data1$Protein,data1$Moisture,main="Protein vs Moisture", xlab = "Protein",ylab = "Moisture")
#3)

set.seed(12345)
X<-cbind(data1$Protein,data1$Moisture)

index=sample(dim(X)[1],round(dim(X)[1]*.5)) #50-50%

train<-X[index,]
test<-X[-index,]
phi_train<-matrix(ncol=7,nrow=length(train[,1]))
phi_test<-matrix(ncol=7,nrow=length(test[,1]))

for(i in 2:7){
  phi_train[,i]<-train[,1]^(i-1)
}
phi_train[,1]<-train[,2]

for(i in 2:7){
  phi_test[,i]<-test[,1]^(i-1)
}
phi_test[,1]<-test[,2]


phi_train<-as.data.frame(phi_train)
phi_test<-as.data.frame(phi_test)

names(phi_train)<-c("Yp","X1","X2","X3","X4","X5","X6")
names(phi_test)<-c("Yp","X1","X2","X3","X4","X5","X6")

n_train=length(train[,1])
n_test=length(test[,1])
```

```
MSE_tr<-vector(length = 6)
MSE_te<-vector(length = 6)
#fit the model and find MSE of the training and of the test set

for(i in 2:7){
  fit<-lm(Yp ~. , data=phi_train[,1:i])
  Y_hat=predict(fit,newdata = phi_test[,1:i])
  MSE_tr[i-1]<-sum(resid(fit)^2)/n_train
  MSE_te[i-1]<-sum((Y_hat-phi_test[,1])^2)/n_test
}

#MSE plot

plot(1:6,MSE_tr,type="l",main="MSE depending on i",ylim=c(31,34.8),ylab="MSE",xlab="Models")
lines(MSE_te,col="red")
#Model with i=6
plot(phi_test[,2],phi_test[,1],xlab="Moisture",ylab="Protein",title="Regression model")
points(phi_test[,2],Y_hat,col="red")
#4)
library(MASS)

Y_new<-data1$Fat
X_new<-data1[,2:101]
database<-cbind(Y_new,X_new)

linear_model<-lm(Y_new~.,data=database)
AIC<-stepAIC(linear_model,trace=FALSE,direction = "both")

#the best model is
AIC$anova

#5)
library(glmnet)
X_new<-scale(X_new)
Y_new<-scale(Y_new)
ridge_model<-glmnet(X_new,Y_new,alpha = 0)

plot(ridge_model,xvar="lambda",label=TRUE)
#6)
lasso_model<-glmnet(X_new,Y_new,alpha = 1)

plot(lasso_model,xvar="lambda",label=TRUE)
#7)
set.seed(12345)

lambda<-c(lasso_model$lambda,0)
lasso_cv <- cv.glmnet(X_new,Y_new,alpha=1,lambda=lambda)
plot(lasso_cv)
coef(lasso_cv)
```