Lab 1 Introduction to Machine Learning

Alejandro García

28th 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%.

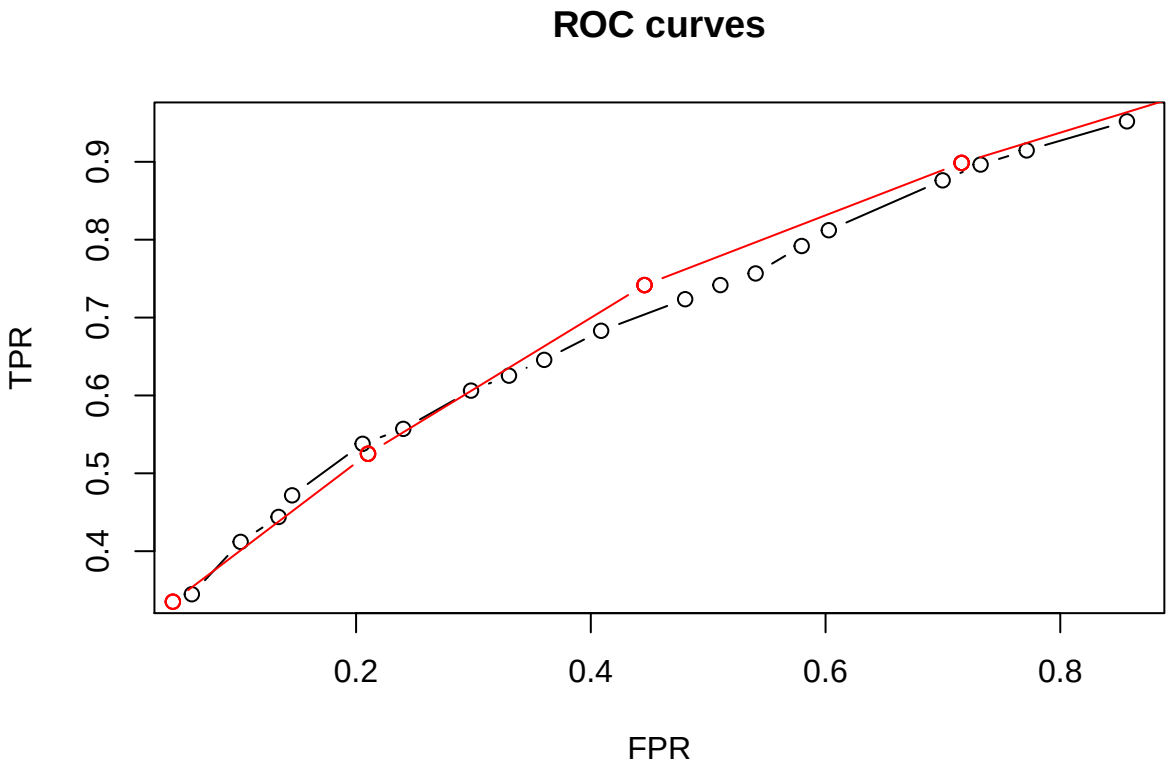
The misclassification rates of each method are reported in the following tab:

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

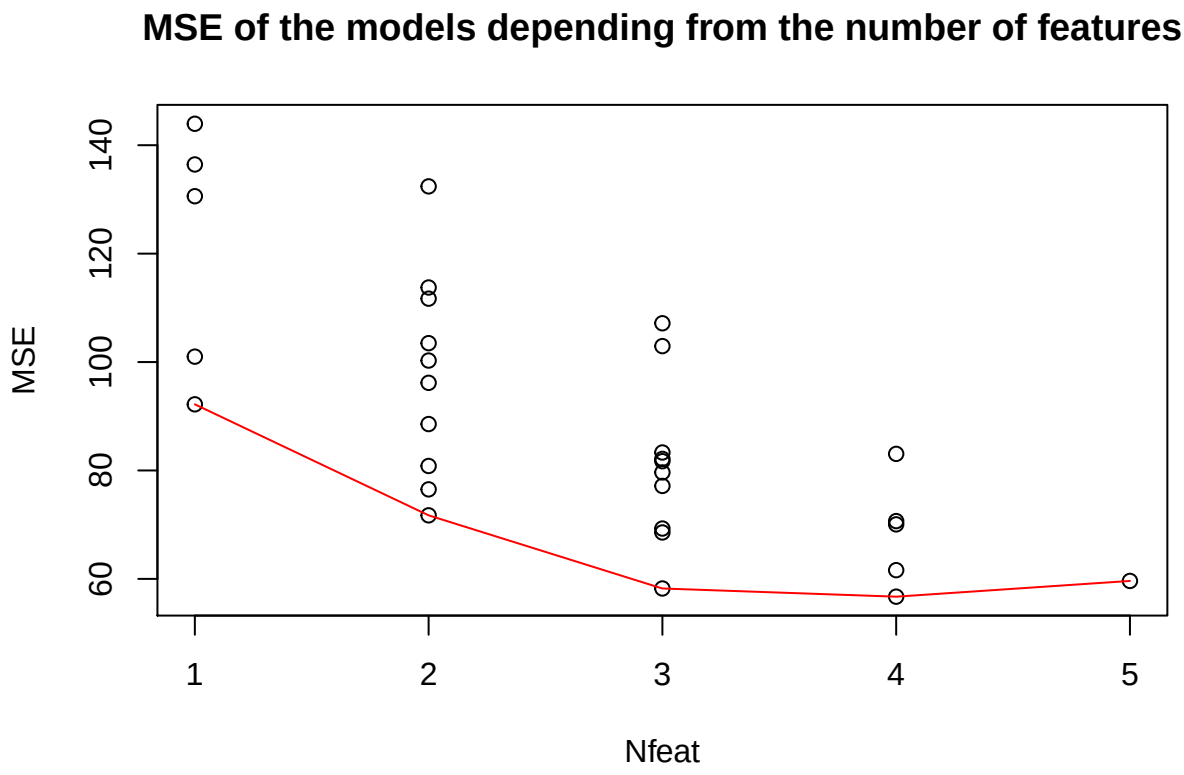
In this assignment it's been implemented a function that performs feature selection (best subset selection) in linear regression by using k-fold cross-validation.

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
}
```

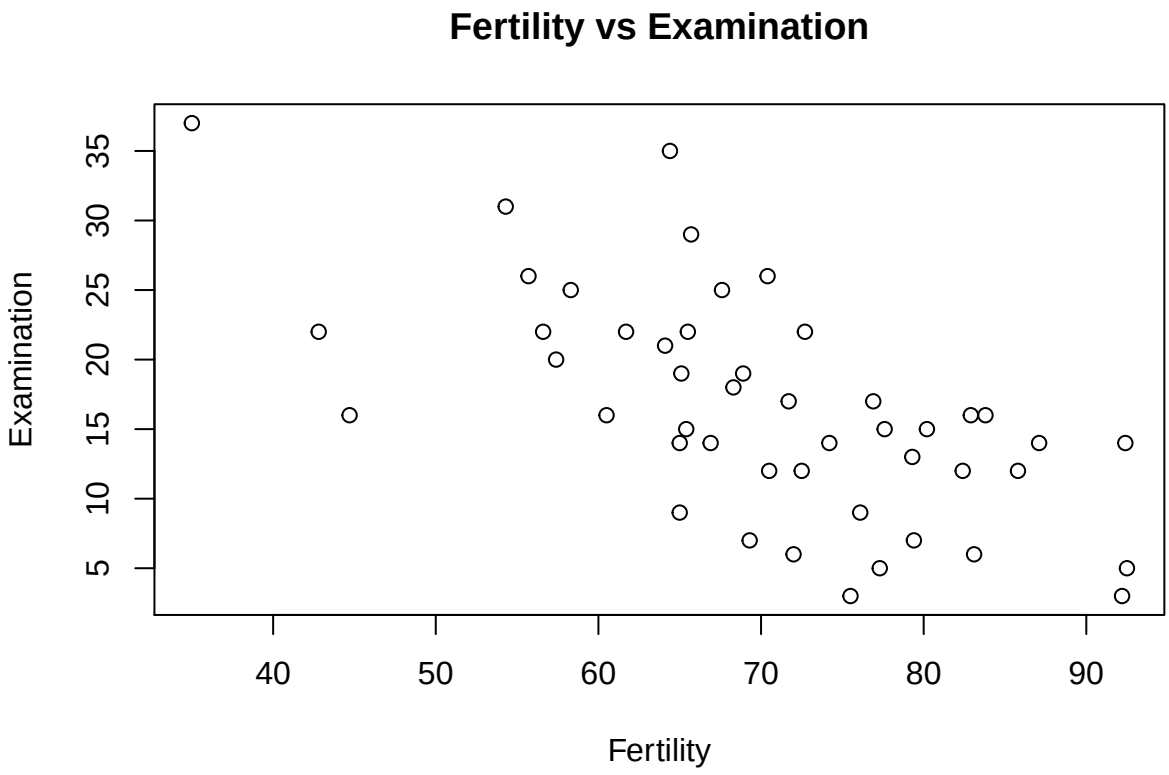
After implementing the function we test it on the set *swiss* where *fertility* is the Y and all other variables are columns of the matrix X. Nfolds is choosen to be equal to 5. The plot obtained shows that the model with lowest MSE is the one with four features: *Agricolture, Education, Catholic and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




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

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

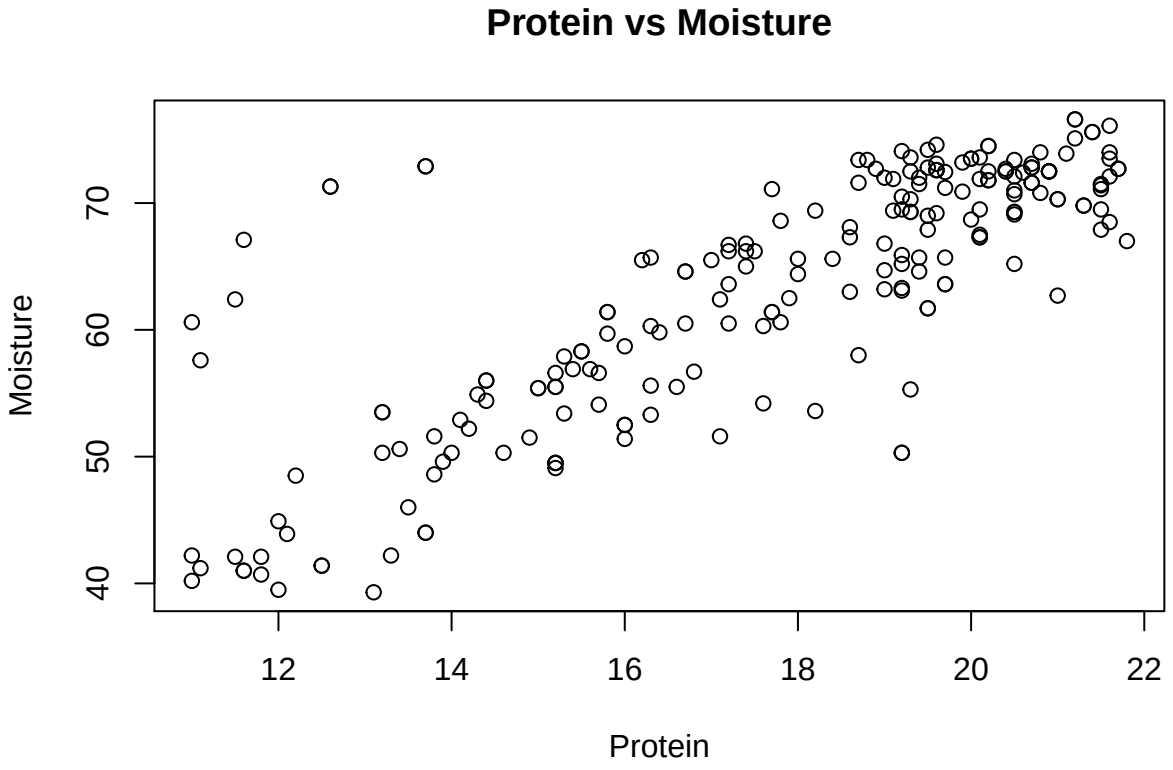
It's reasonable not to consider the feature Examination in the model infact, as we can see from the following plot, the target variable *Fertility* doesn't really seam to have a kind of relation with *Examination*:



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

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

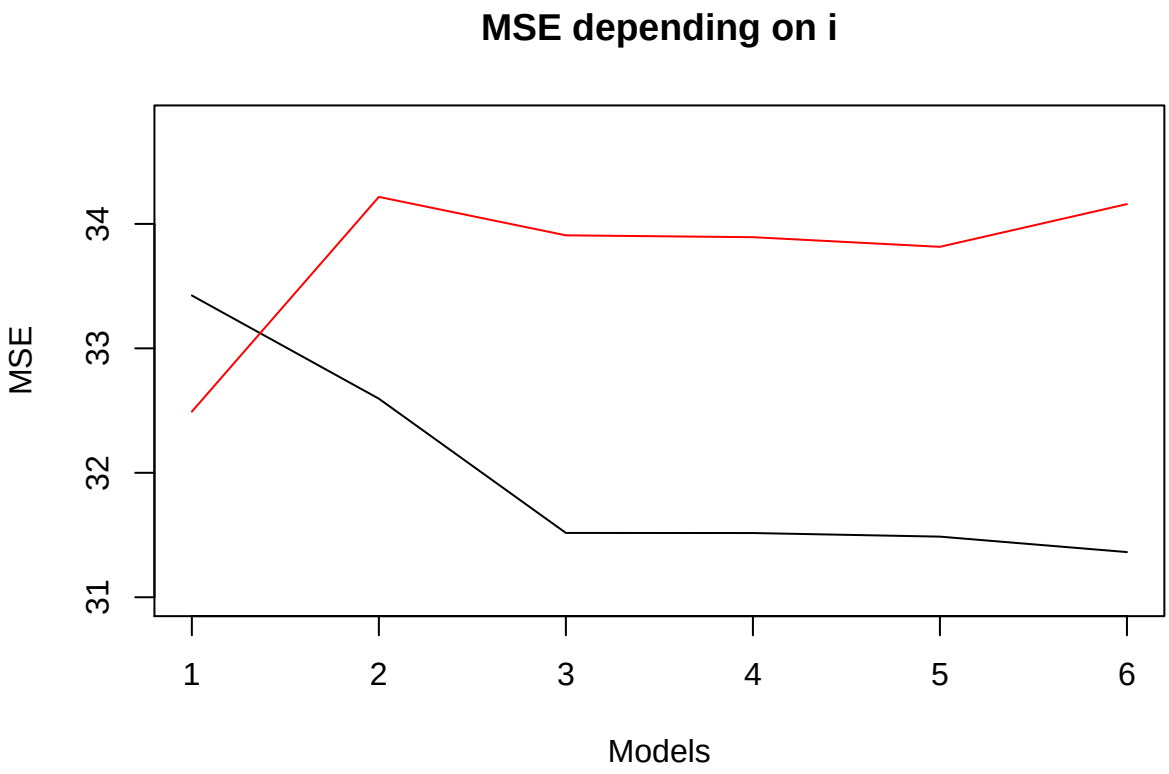
and

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

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.

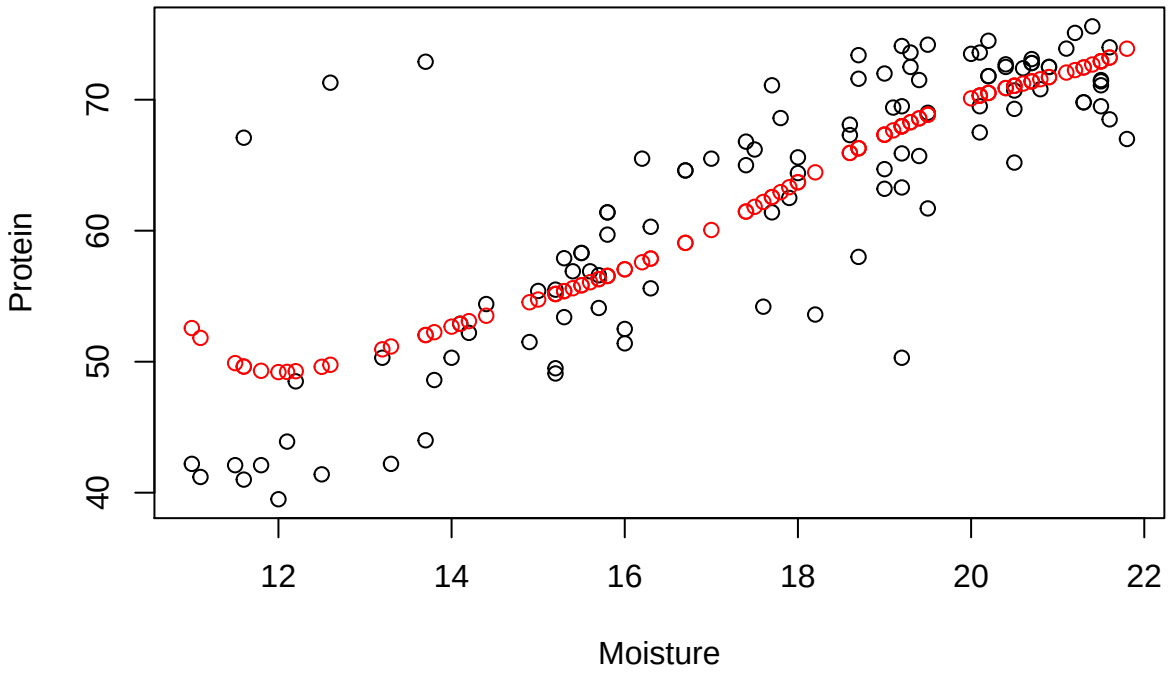
```
#3)

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:



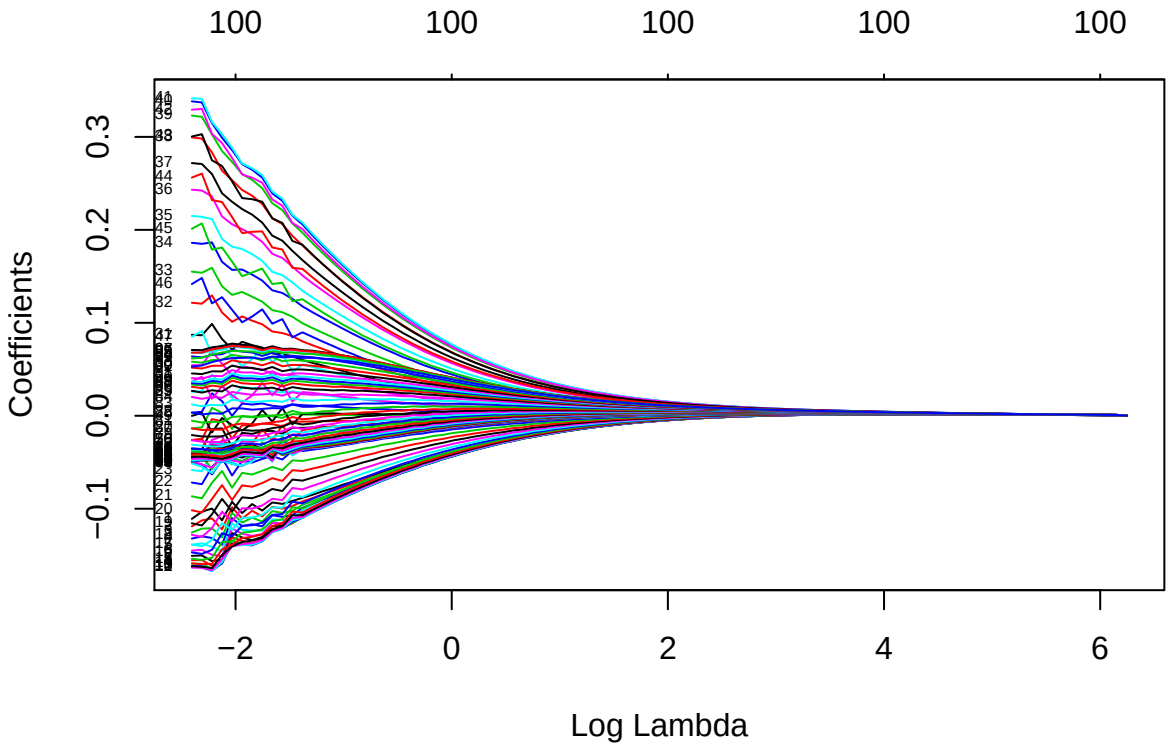
4. The dataset also contains other variables such as *Fat* and 100 *Channels* spectrum of absorbance records. In this second part of the assignment the new response variable is *Fat* and *Channel1-Channel100* are the new predictors. We use the *stepAIC()* function on the linear regression model obtained 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 +
```

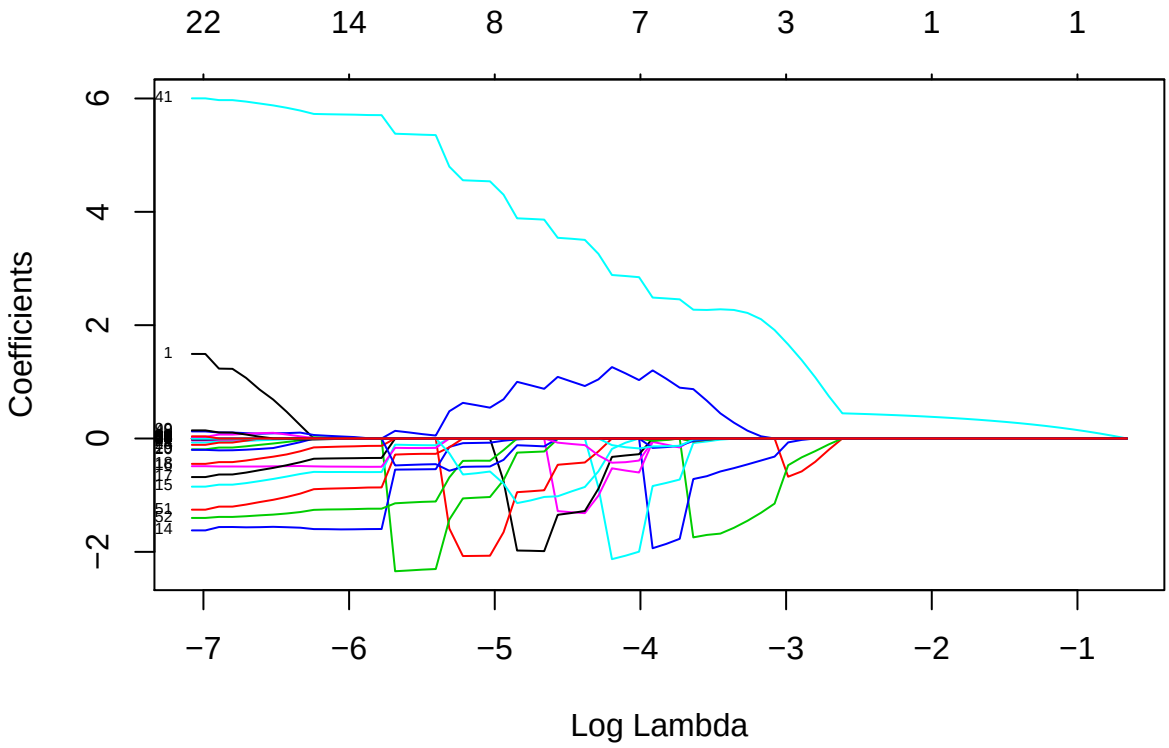
```
##     Channel87 + Channel88 + Channel89 + Channel90 + Channel91 +
##     Channel92 + Channel93 + Channel94 + Channel95 + Channel96 +
##     Channel97 + Channel98 + Channel99 + Channel100
##
## Final Model:
## Y_new ~ Channel1 + Channel2 + Channel4 + Channel5 + Channel7 +
##     Channel8 + Channel11 + 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
```

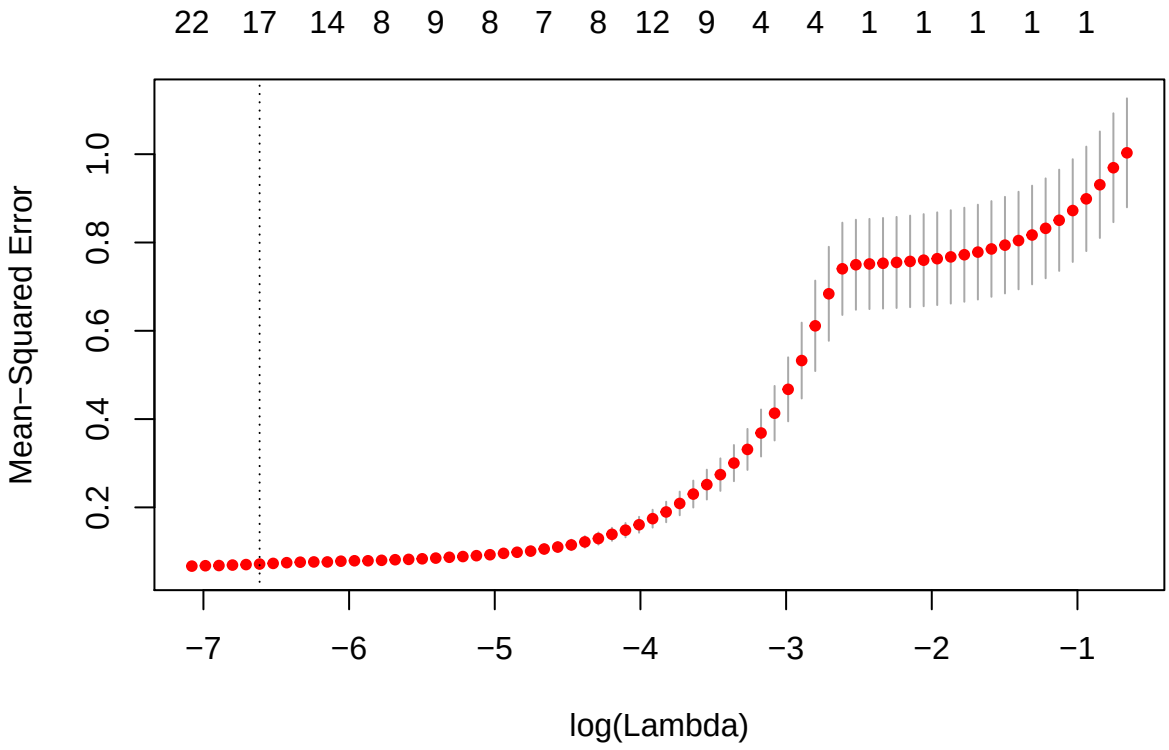
5. We fit now a Ridge Regression model with the same predictor and response variables. The following plot shows how model coefficients depend on the log of the penalty factor λ. The starting point of the graph represent $\lambda = 0$ so the coefficients are not effected by any penalty. As soon as λ grows 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 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. Finally, the plot doesn't show the λ equals to 0 because its logarithm is -infinite, however we can see that as the best one is 0 the best fit for the data is a linear regression.



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)
```

```
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])
```

```
  }

  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])
```

```r
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)
```

Lab1 Block 2

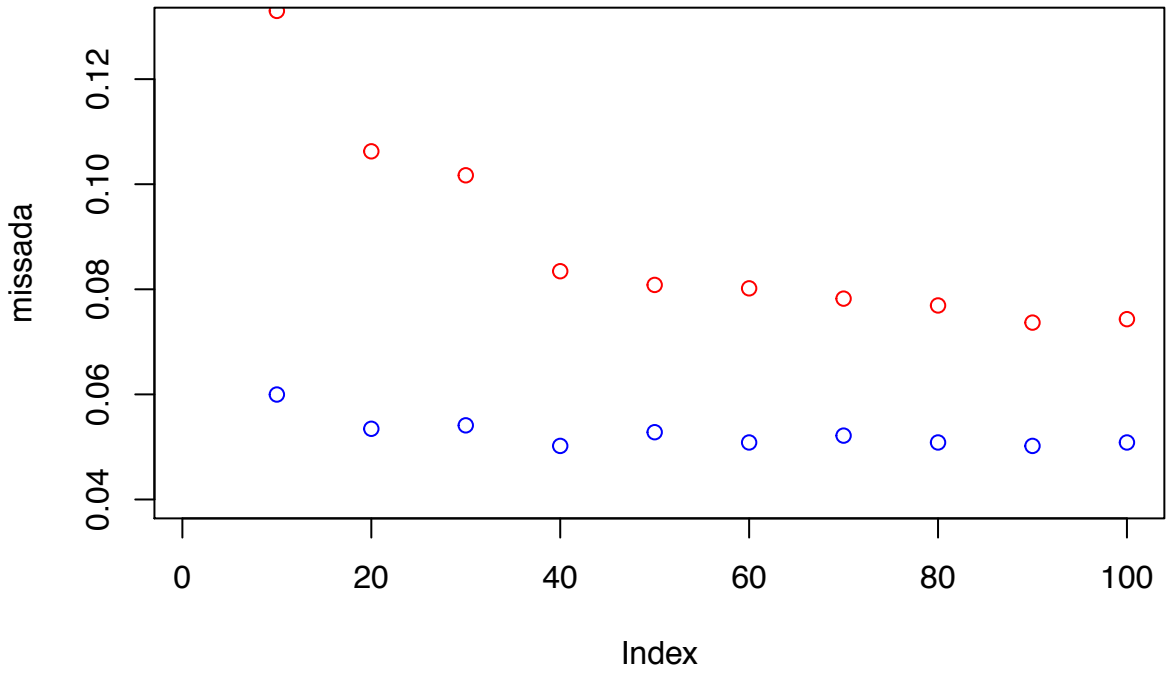
Alejandro Garcia, Alessia De Biase, Balaji Ramkumar

November 28, 2017

1. Ensemble Methods

In this assignment we are asked to evaluate Adaboost classification trees and random forests on the spam database, where Spam has been classified as (spam=0) when regular e-mails and (spam=1) when spam.

To evaluate these algorithms we separate the data into training data, with 2/3 of the data and 1/3 of the data for the hold-out test data. We also provide a plot showing the error rates when the number of trees are 10, 20,. . . ,100.

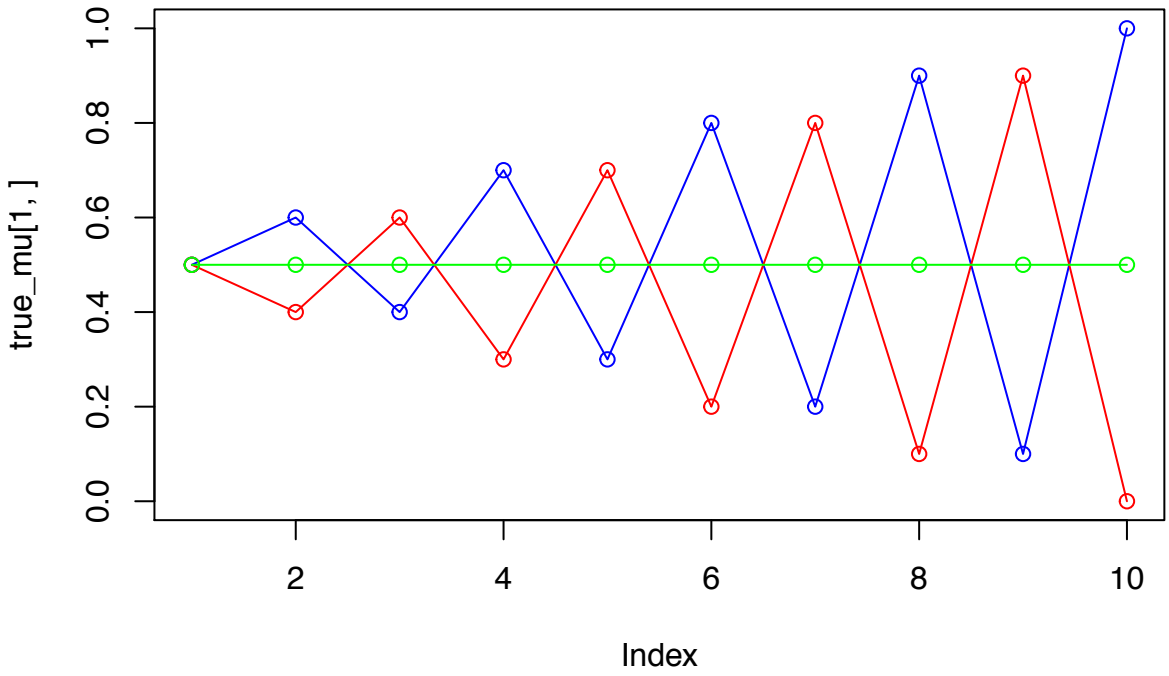


As we can see in the plot above, the best algorithm for this database is the random forest as the error rate is lower than adaboost. The best number of trees considering complexity and error rate is 20 trees for random forest, while the optimal one for adaboost classification trees is 40 trees.

2. Mixture models

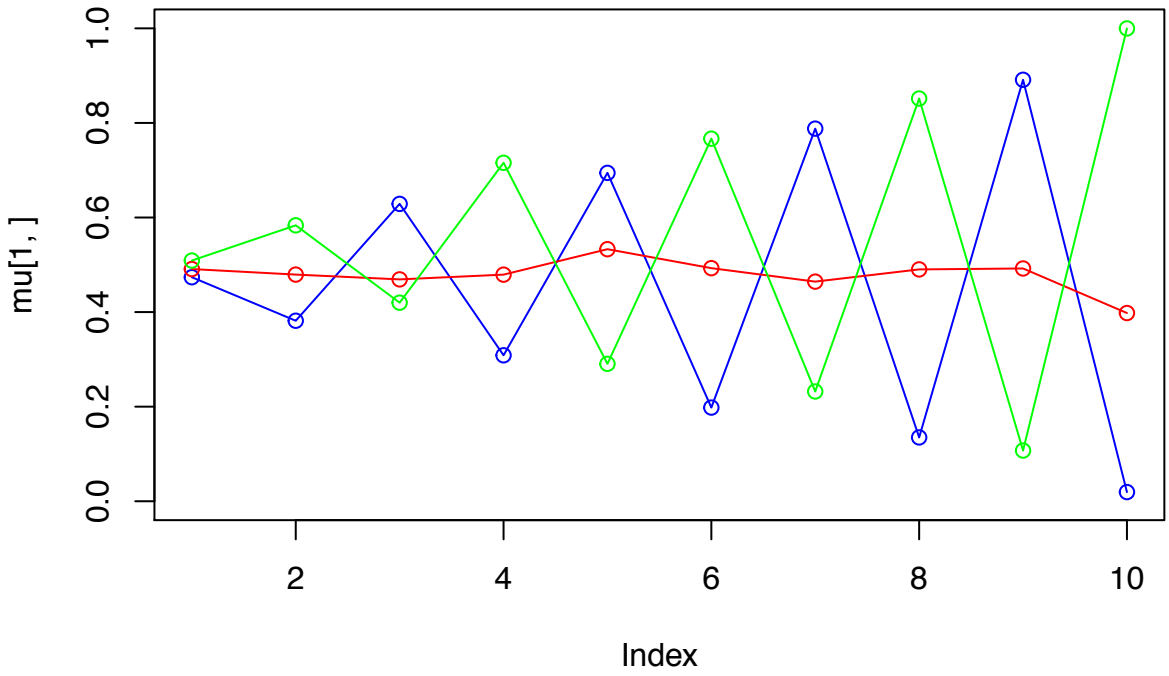
In this task we are asked to implement the EM algorirthm for mixtures of multivariate Benoulli distributions.

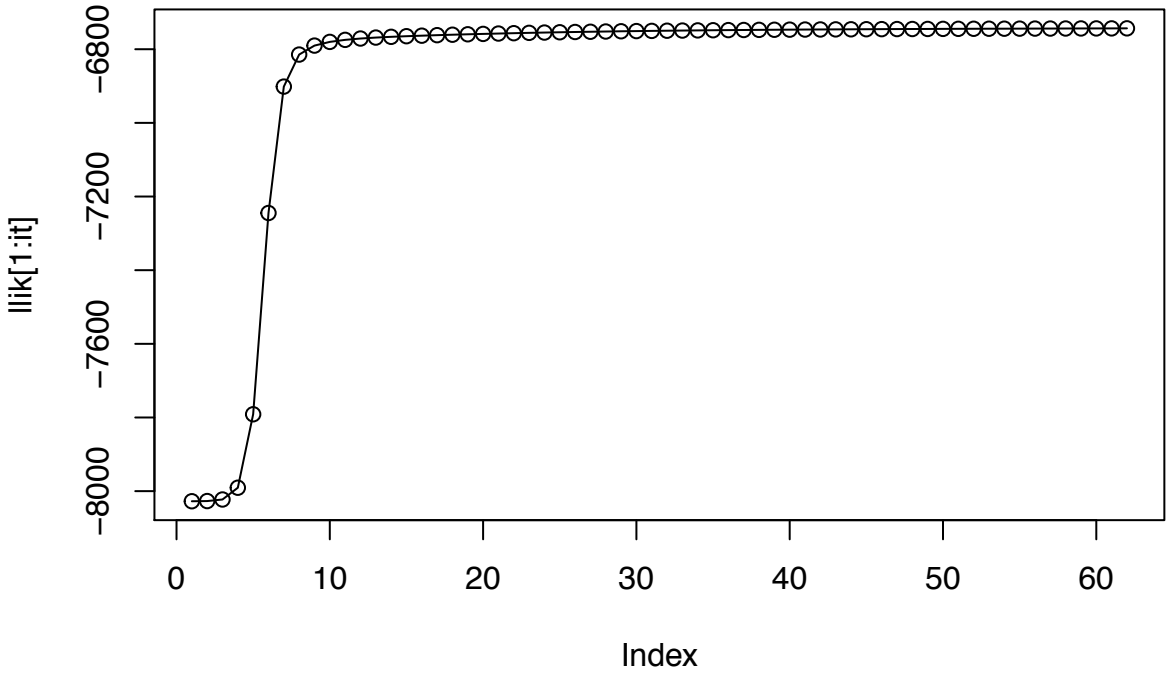
The plot above shows the real means of the clusters.



K = 3

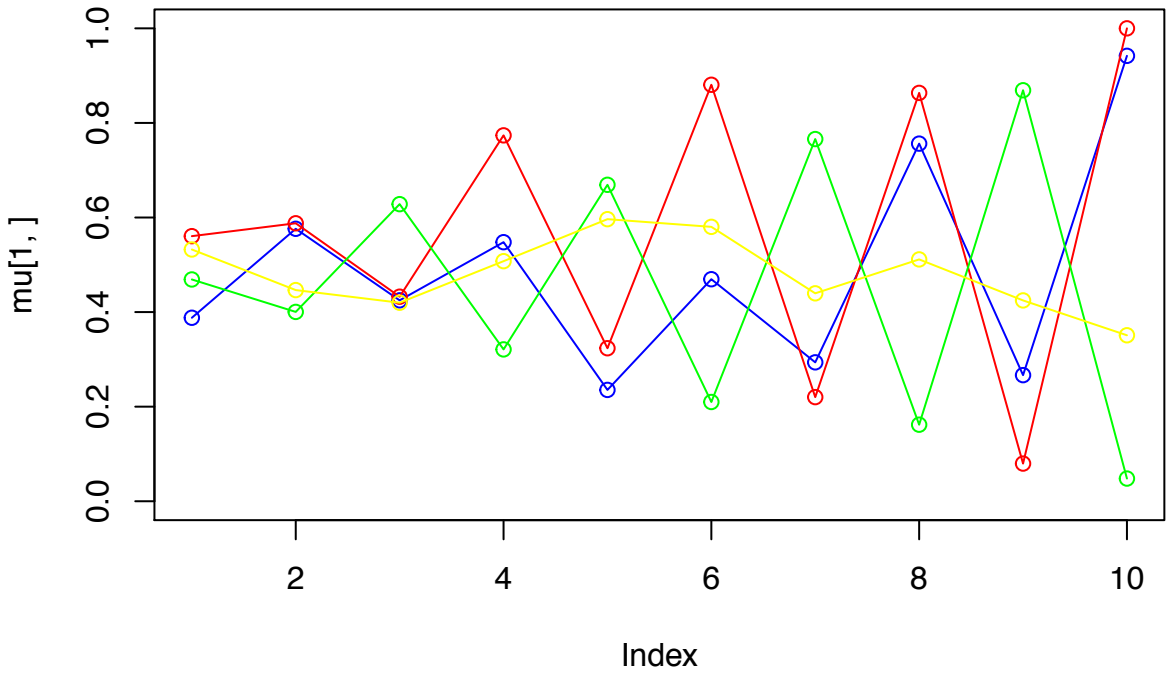
We can see that 3 are the real clusters and they fit pretty well with the data.

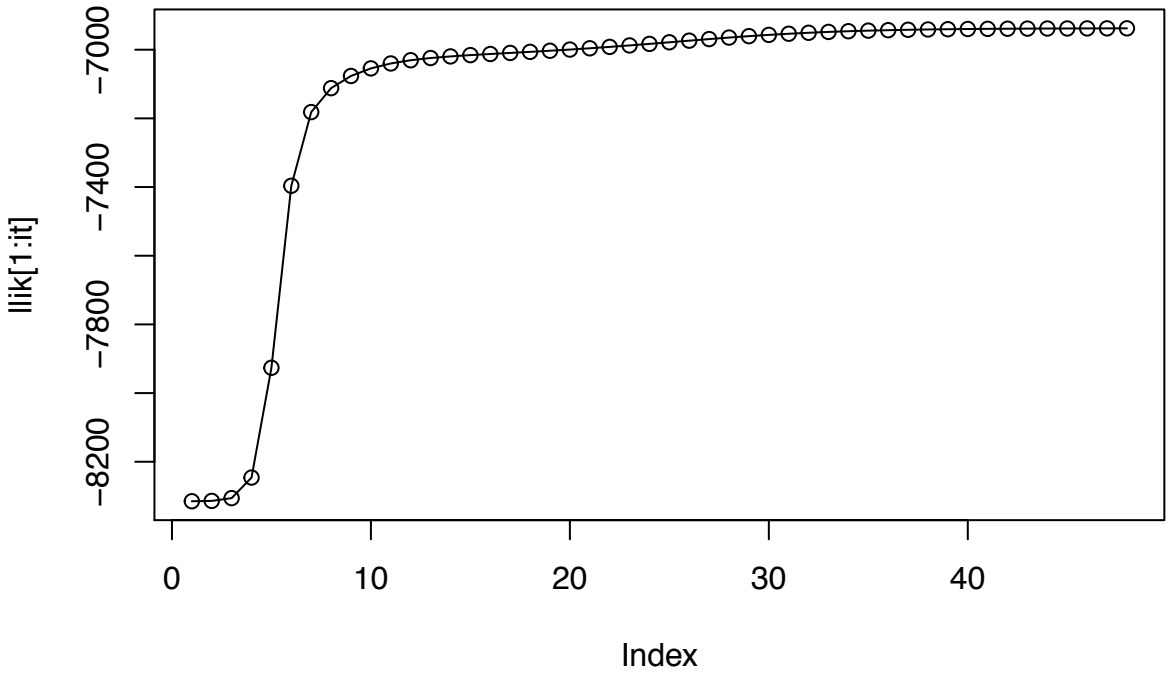

mu[1,]
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
Index



K = 4

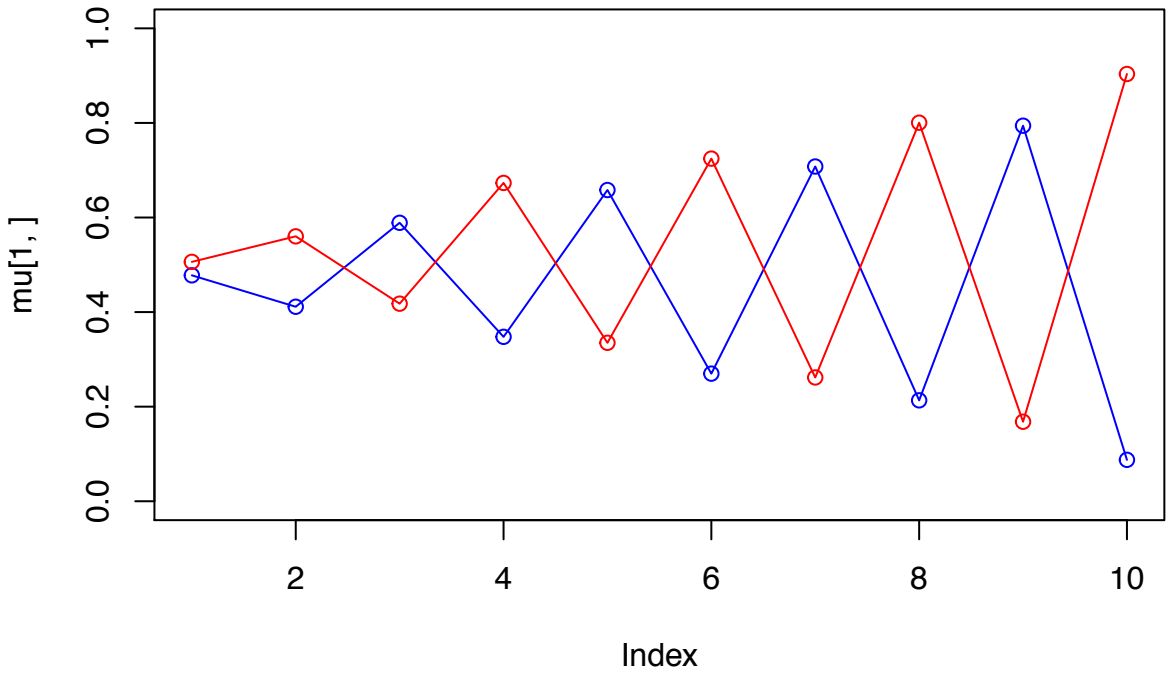
With 4 iterations the log likelihood is lower than with 3, and the clustes are quite different. It makes sense as we are trying to fit 4 clusters into 3 which is not the real number.

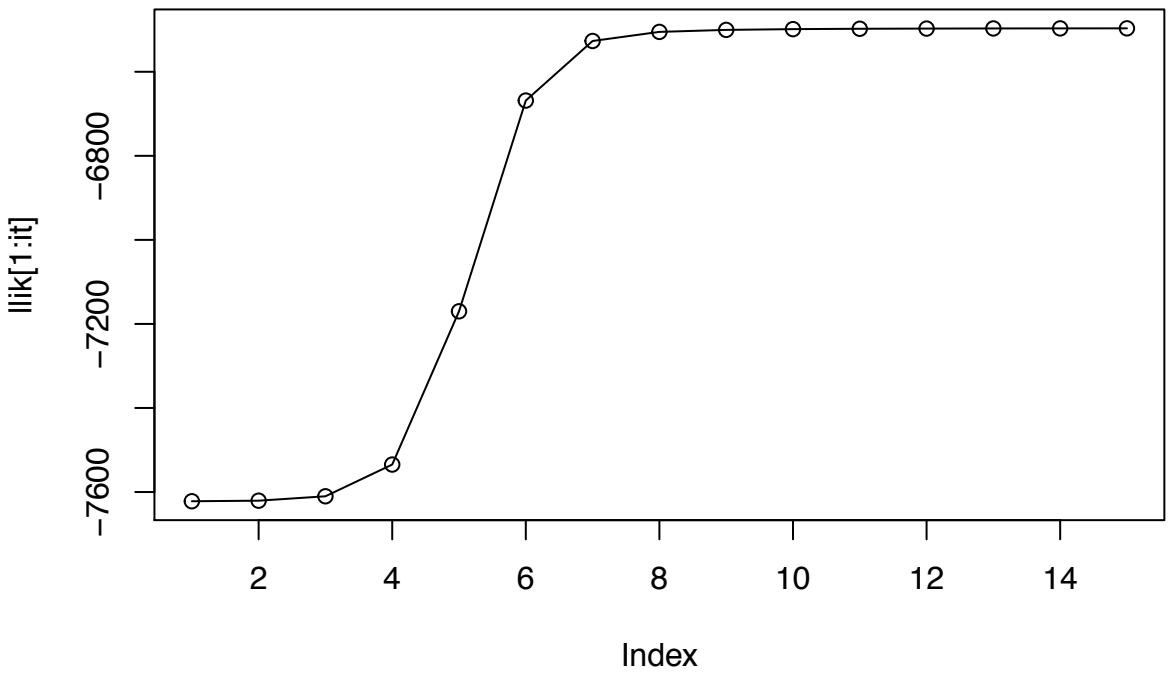

mu[1,]
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
Index



K = 2

When trying to fit 2 clusters we see an interesting thing, the log likelihood is higher than with 3 clusters, meaning that 2 clusters should be enough to fit the data. This can be because of the data we are using, as it only has two types of values (1 or 0), the best approach is to fit two clusters, however, as we have 3 real clusters we wouldn't be using the correct amount of clusters.


mu[1,]
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
Index



Code

```
library(mboost)
library(randomForest)
set.seed(12345)

spam<-read.csv("spambase.csv", sep = ";", dec=",")

n <- nrow(spam)
ind <- sample(n, floor(n*2/3))
train <- spam[ind,]
test <- spam[-ind,]

num <- seq(10,100,10)
ada <- blackboost(as.factor(Spam)~., data=train, family=AdaExp(), control=boost_control(mstop=1))

missada <- vector()
missrf <- vector()
for(i in num){
  ada <- blackboost(as.factor(Spam)~., data=train, family=AdaExp(), control=boost_control(mstop=i))
  Yada <- predict(ada, newdata=test, type="class")
  tab <- table(Yada, test$Spam)
  missada[i] <- (tab[1,2]+tab[2,1])/sum(tab)
```

```
  rf <- randomForest(as.factor(Spam)~., data=train, ntree=i)
  Yrf <- predict(rf, newdata=test, type="class")
  tab2 <- table(Yrf, test$Spam)
  missrf[i] <- (tab2[1,2]+tab2[2,1])/sum(tab2)
}

plot(missada, ylim = c(0.04,0.13), col="red")
points(missrf, col="blue")

## Assignment 2

set.seed(1234567890)
max_it <- 100 # max number of EM iterations
min_change <- 0.1 # min change in log likelihood between two consecutive EM iterations
N=1000 # number of training points
D=10 # number of dimensions
x <- matrix(nrow=N, ncol=D) # training data
true_pi <- vector(length = 3) # true mixing coefficients
true_mu <- matrix(nrow=3, ncol=D) # true conditional distributions
true_pi=c(1/3, 1/3, 1/3)
true_mu[1,]=c(0.5,0.6,0.4,0.7,0.3,0.8,0.2,0.9,0.1,1)
true_mu[2,]=c(0.5,0.4,0.6,0.3,0.7,0.2,0.8,0.1,0.9,0)
true_mu[3,]=c(0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5)
plot(true_mu[1,], type="o", col="blue", ylim=c(0,1))

points(true_mu[2,], type="o", col="red")
points(true_mu[3,], type="o", col="green")
# Producing the training data

for(n in 1:N) {
  k <- sample(1:3,1,prob=true_pi)
  for(d in 1:D) {
    x[n,d] <- rbinom(1,1,true_mu[k,d])
  }
}
K=3 # number of guessed components
z <- matrix(nrow=N, ncol=K) # fractional component assignments
pi <- vector(length = K) # mixing coefficients
mu <- matrix(nrow=K, ncol=D) # conditional distributions
llik <- vector(length = max_it) # log likelihood of the EM iterations
logp <- 0
# Random initialization of the paramters
pi <- runif(K,0.49,0.51)
pi <- pi / sum(pi)
for(k in 1:K) {
  mu[k,] <- runif(D,0.49,0.51)
}
pi
mu
for(it in 1:max_it) {

  #Sys.sleep(0.5)
  # E-step: Computation of the fractional component assignments
```

```
  # Your code here
  for(n in 1:N){
    temp2v <- numeric(3)
    for(k in 1:K){
      temp = 1
      for(i in 1:D){
        temp <- temp*dbinom(x[n,i], size =1, prob = mu[k,i])
      }
      temp2v[k] <- pi[k]*temp
      z[n,k] <- pi[k]*temp
    }
    z[n,] <- z[n,]/sum(temp2v)
  }
  piML <- colSums(z)/N
  muML <- matrix(nrow=K, ncol=D)
  for(i in 1:K){
    muML[i,] <- (t(x[,])%*%z[,i]/sum(z[,i]))
  }

  #Log likelihood computation.
  # Your code here
  logptemp <- 0
  for(n in 1:N){
    for(k in 1:K){
      temp1 <- 0
      for(i in 1:D){
        temp1 <- temp1+(x[n,i]*log(muML[k,i]))+((1-x[n,i])*log(1-muML[k,i]))
      }
      logptemp <- logptemp+(z[n,k]*(log(piML[k])+temp1))
    }
  }
  llik[it] <- logptemp

  #cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
  #flush.console()
  # Stop if the lok likelihood has not changed significantly
  # Your code here
  if(abs(logp-logptemp) < min_change) {
    plot(mu[1,], type="o", col="blue", ylim=c(0,1))
    points(mu[2,], type="o", col="red")
    points(mu[3,], type="o", col="green")
    #points(mu[4,], type="o", col="yellow")
    break
  }
  logp <- logptemp
  #M-step: ML parameter estimation from the data and fractional component assignments
  # Your code here
  mu <- muML
  pi <- piML
}
plot(llik[1:it], type="o")
```

Lab 2 Introduction to Machine Learning

Alejandro Garcia

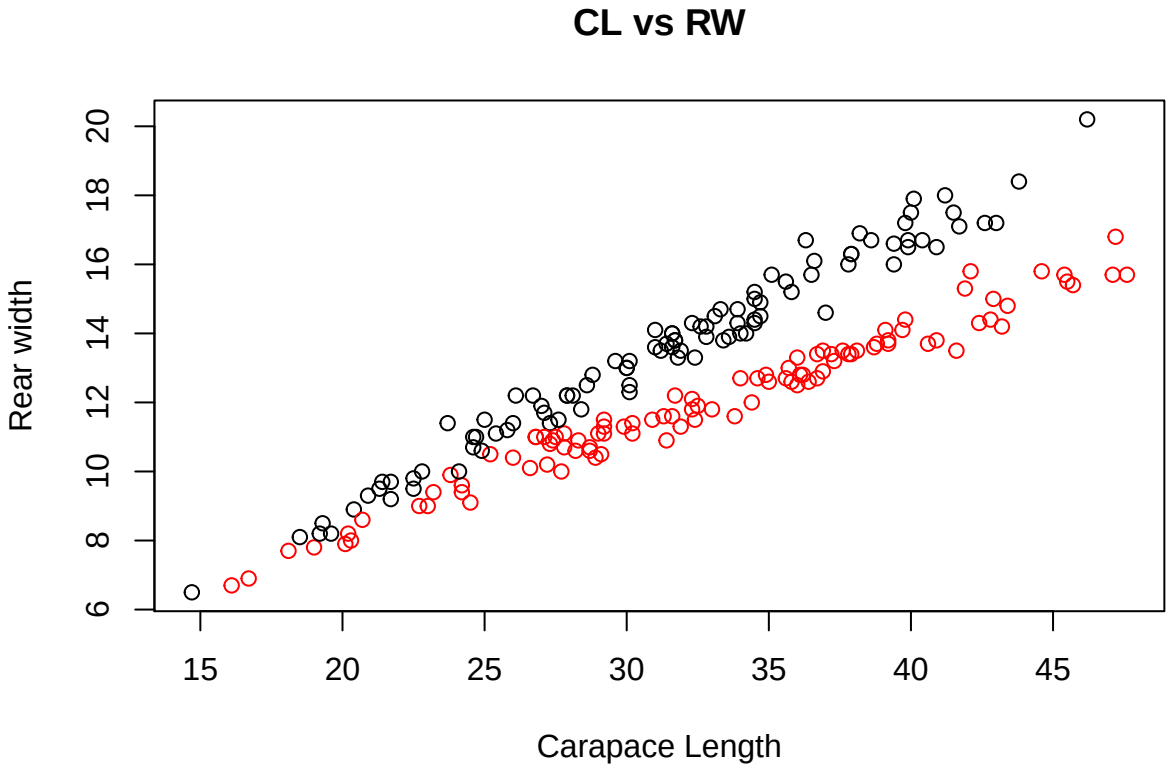
26 novembre 2017

Assignment 1 - LDA and logistic regression

The data file *australian-crabs.csv* contains measurements of various crabs.

1. In the following graph we show the relation between the carapace length (**CL**) and the rear width (**RW**) colored by Sex. The red dots represents the males crabs and the black dots the females.There are just few points of different classes overlapping, in general the dataset is pretty separable that's why we can use this kind of method to separate the data linearly with an high accuracy.

However LDA may be a bit biased because each of the classes doesn't seem to have same covariance matrix.



2. In this second part we implement *LDA*: we classify the observations and we extract the discriminant functions and equation of the decision boundary. The function `disc_fun()` has as output the vector w^T helpful to build the decision boundary.

```
#2)

#in this function we find w_0 and w_1 for the discriminant functions.

disc_fun=function(label, X ,S){
```

```
X1=X[X==label,]

w<-length(2)

mu<-c(mean(X1[,2]),mean(X1[,3])) #class label

w1<-solve(S) %*% mu

p<-nrow(X1)/dim(X)[1]

b1<-matrix
b1= -(t(mu)%*% solve(S) %*% mu + log(p))/2

return(c(w1[1], w1[2], b1[1,1]))
}

X=data.frame(Y=dat$sex,RW=dat$RW,CL=dat$CL)

X1=X[X=="Male",]
X2=X[X=="Female",]

#covariance matrix
S=cov(X1[,-1])*dim(X1[,-1])[1]+cov(X2[,-1])*dim(X2[,-1])[1]
S=S/dim(X)[1]

#discriminant function coefficients
res1=disc_fun("Male",X,S)
res2=disc_fun("Female",X,S)

res=res1-res2
```

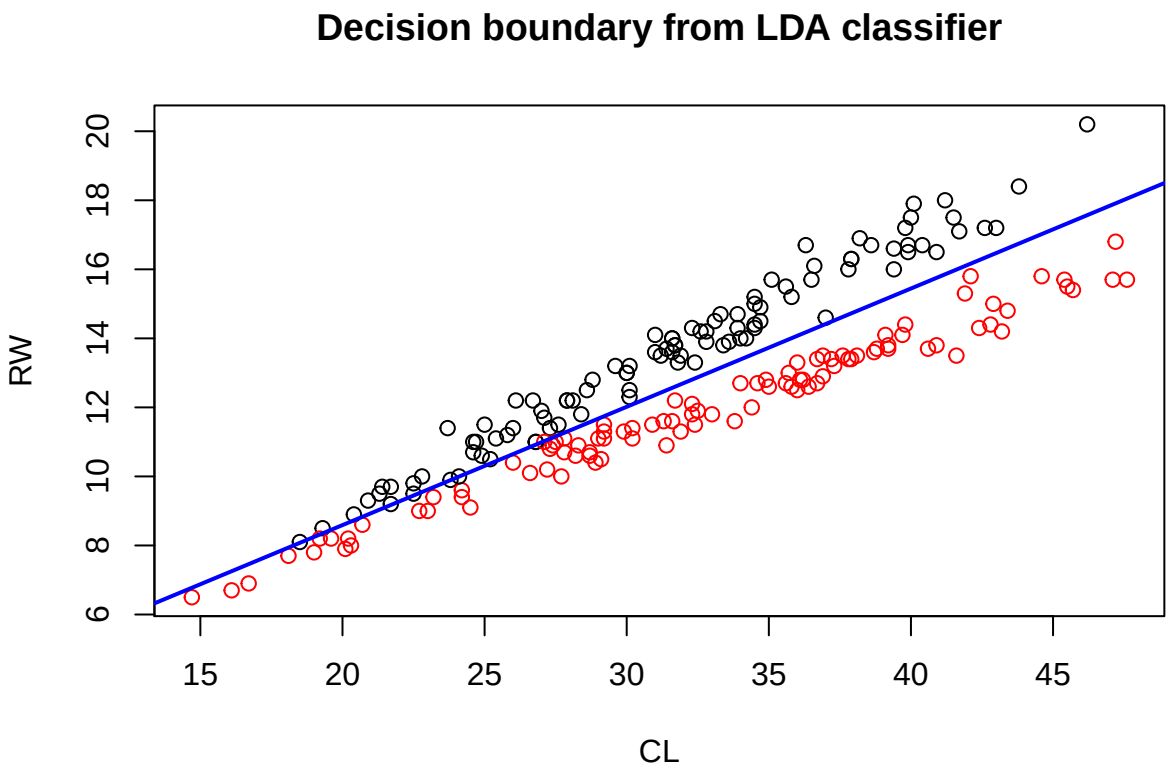
3. The following plot shows the data divided by the decision boundary found thanks to the function from step 2. The function of the boundary is linear and $w_0, W_1 and b_1$ are the outputs of the `disc_fun()` function:

$$f = (w_0^1 - w_0^2) * X_{RW} + (w_1^1 - w_1^2) * X_{CL} + (b_1^1 - b_1^2) \quad (1)$$

.

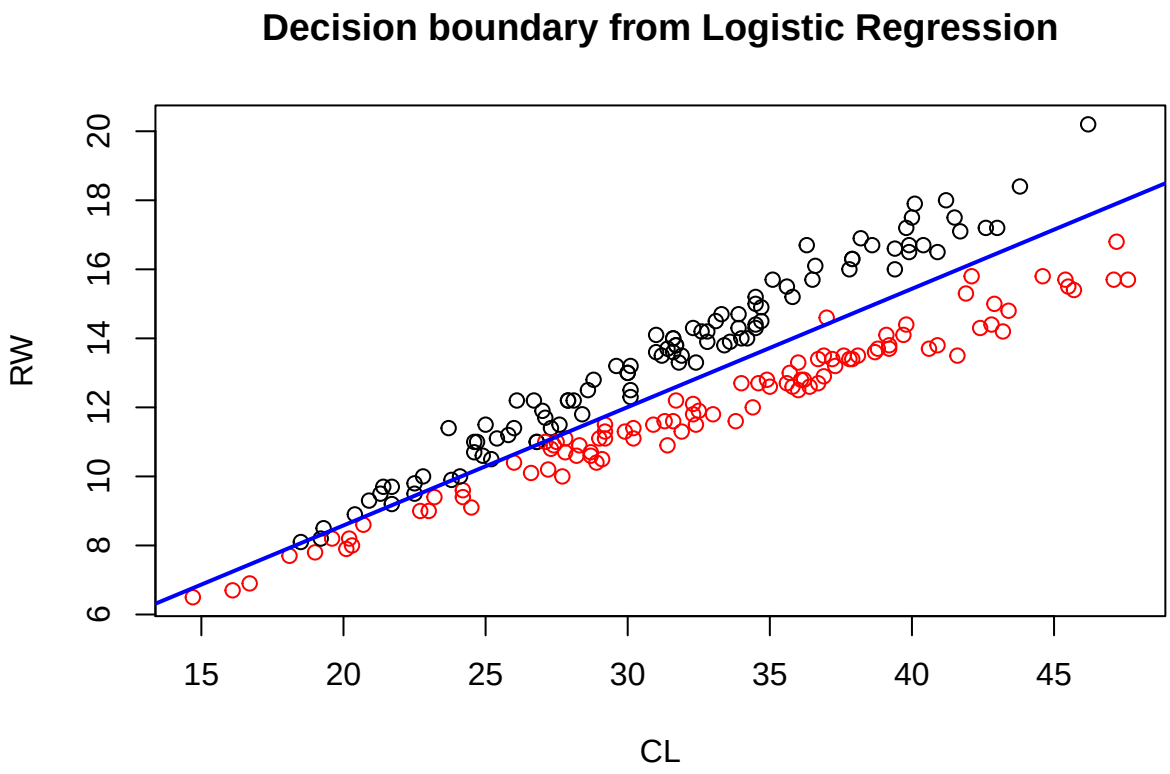
The decision boundary is: y= 1.74 x + 0.34

```
#3) classification
d=res[1]*X[,2]+res[2]*X[,3]+res[3]
Yfit=(d>0) #true female
plot(X[,3], X[,2], col=Yfit+1, xlab="CL", ylab="RW",main="Decision boundary from LDA classifier")


t_LDA=table(Yfit,dat$sex)
misclassification_LDA<-(t_LDA[1,2]+t_LDA[2,1])/sum(t_LDA)*100

abline(coef=c(-res[3]/res[1],-res[2]/res[1]),col="blue",lwd=2)
```



The fit is very good, this is because the data were already pretty separable. *LD* classifier performances are great when the data are linear.

4. In this last point we use *logistic regression* to make a similar classification using the function `glm()`. Logistic regression predictions are values between 0 and 1 and represent probabilities, so we classify as female if the prediction is lower than 0.5 and male if it's higher.



The decision boundary in this case is: y= 1.72 x + 0.34

The logistic model has this formula:

$$P(a) = \frac{1}{1 + e^{-a}} \tag{2}$$

where $a = w^T x$ and

$$w = (13.62, -12.56, 4.63) \tag{3}$$

Looking at both the decision boundaries the two methods show almost the same results, also the misclassification rates are the same. The logistic method, though, should perform better in this case because it doesn't assume any distribution in predictors while LDA does, it neither requires anything about the within-group covariance matrices of the predictors and it's more robust.

Assignment 2 - Analysis of credit scoring

In this assignment we use the *creditscoring.xls* file which contains data retrieved from a database in a private enterprise. The aim is to derive a prediction model to predict whether or not a new customer is likely to pay back the loan.

1. In this first part we import the data and divide them into training, validation and test sets:

```
#-------------------------------------#
#ASSIGNMENT 2
#1)

library(readxl)
```

```
## Warning: package 'readxl' was built under R version 3.4.3
```

```
dat2<-read_excel("creditscoring.xls",1)
dat2$good_bad<-as.factor(dat2$good_bad)

n=dim(dat2)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=dat2[id,]

id2<-(1:n)[!(1:n %in% id)]
n2=dim(dat2[-id,])[1]
id3=sample(id2, floor(n2*0.5))
test=dat2[id3,]
validation=dat2[-c(id3,id),]
```

2. Now we fit a decision tree to the training data using the measures of impurity *Deviance* and *Gini index* with the `tree()` function.

```
#2)

library(tree)
```

```
## Warning: package 'tree' was built under R version 3.4.2
```

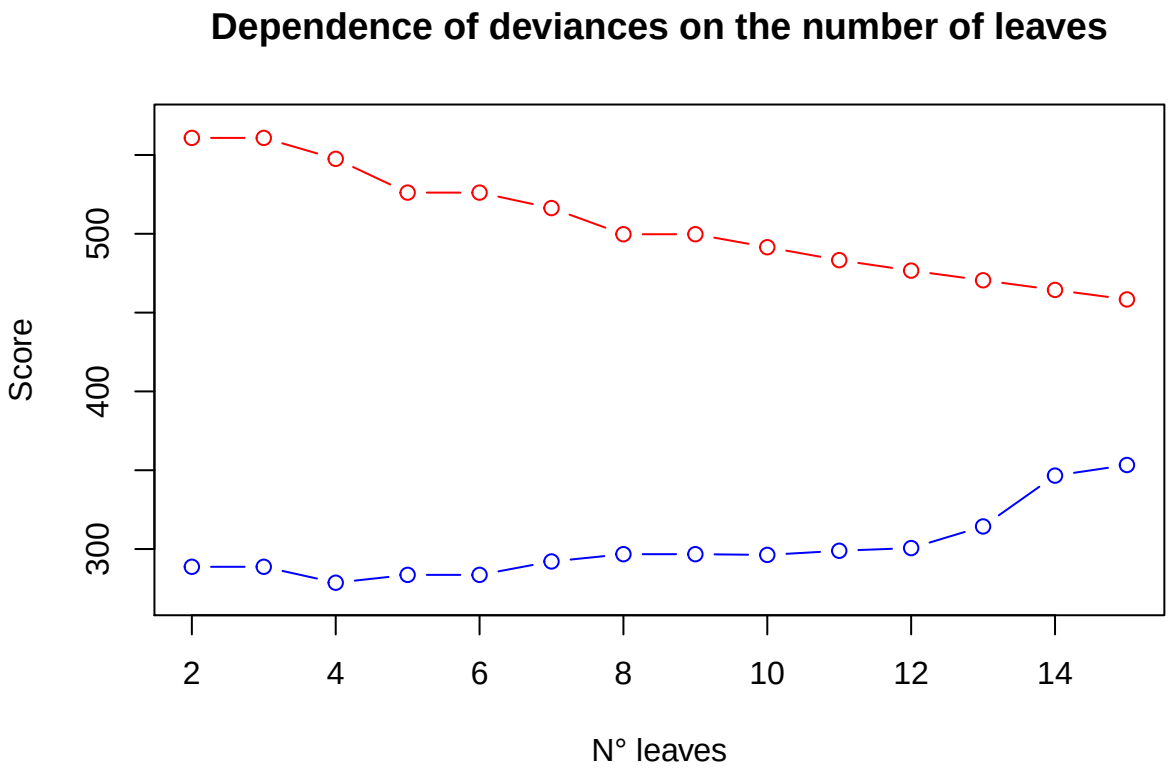
```
n3=dim(train)[1]

fit=tree(good_bad~., data=train,split=c("deviance","gini"))

Yfit=predict(fit, newdata=test,type="class")
t<-table(test$good_bad,Yfit)
misclassification<-(t[1,2]+t[2,1])/sum(t)*100

Yfit_train=predict(fit,newdata=train,type="class")
t_train<-table(train$good_bad,Yfit_train)
misclassification_train<-(t_train[1,2]+t_train[2,1])/sum(t_train)*100
```

The misclassification rate for the training data is 21.2% and the misclassification rate for the test data is 24.8 %.

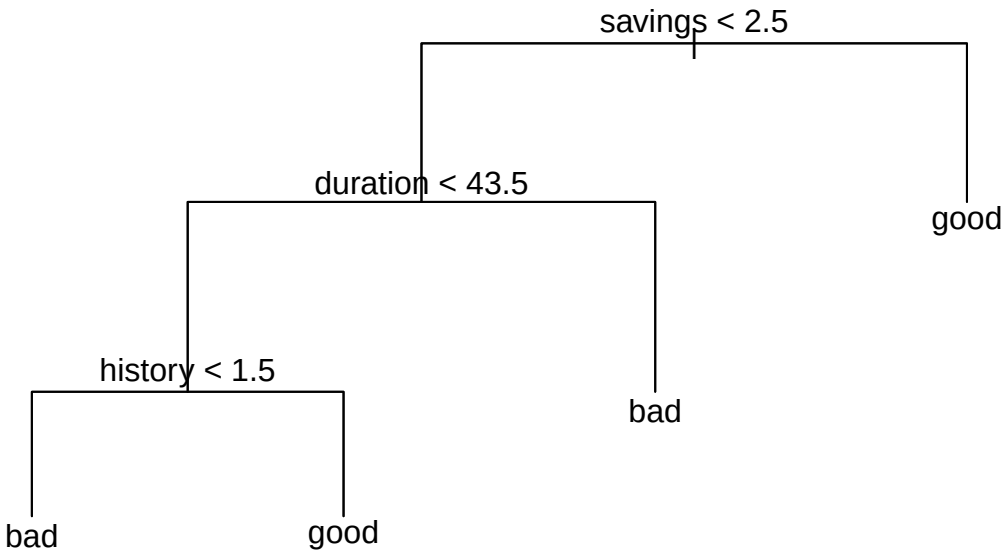
3. In this step we use the training and the validation sets to choose the optimal tree depth.

The following graph show the dependence of deviances from the training and the validation data on the number of leaves. The red line represents the training data and the black one is the validation. While the

training score decreases, the validation score increases and this is obvious because we are fitting the tree with the training data. What's curious is that the score error from the validation data is lower that the one from the test data because of how we splitted the data.



According to the graph the best tree has 4 leaves (the minimum of the blue function).




```
##
## Classification tree:
## snip.tree(tree = fit, nodes = c(5L, 3L, 9L))
## Variables actually used in tree construction:
## [1] "savings"  "duration" "history"
## Number of terminal nodes:  4
## Residual mean deviance:  1.117 = 547.5 / 490
## Misclassification error rate: 0.251 = 124 / 494
```

The optimal tree has four leaves.

When the variable `savings` is greater than 2.5 then the customer is classified as a good customer otherwise if the `duration` variable is greater than 43.5 the customer is classified as bad, otherwise if `history`is less than 1.5 then the customer is classified as bad if it's greater then he is classified as good.

The misclassification rate for the test data is 26%.

4. In this step we perform classification using *Naive Bayes*.

The following tab is the confusion matrix for the training data:

	bad	good
bad	95	98
good	52	255

the misclassification rate is 30%.

The following tab is the confusion matrix for the test data:

	bad	good
bad	50	61
good	25	114

the misclassification rate is 34.4%.

The misclassification rate from the test set is higher for the *Naive Bayes* method than for the *tree classification*. Usually decision trees work better with lots of data compared to Naive Bayes, this last method works quite well when the training data doesn't contain all possibilities so it's good when we have few data. In this case the best model is the classification tree because it's a less complex model (with only 4 leaves).

5. In this step we repeat the *Naive Bayes classification* using the following loss matrix

$$L = \begin{bmatrix} 0 & 1 \\ 10 & 0 \end{bmatrix} \tag{4}$$

.

This loss matrix tells us that loosing a client costs us less than giving a loan to new bad client.

The following tab is the confusion matrix for the training data:

	bad	good
bad	137	263
good	10	90

The following tab is the confusion matrix for the test data:

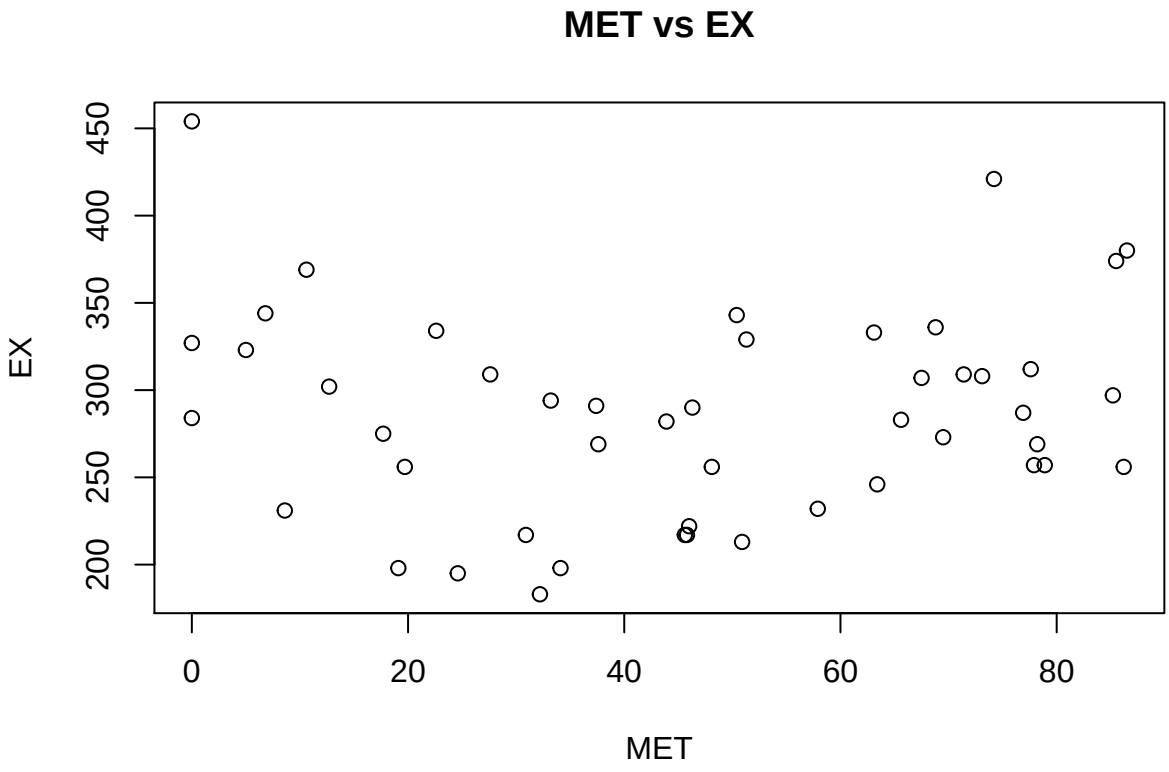
	bad	good
bad	66	130
good	9	45

The tables obtained by this step are way different from the ones obtained in step 4. The loss function gives more weight to the classification of bad clients so the misclassification rates are higher.

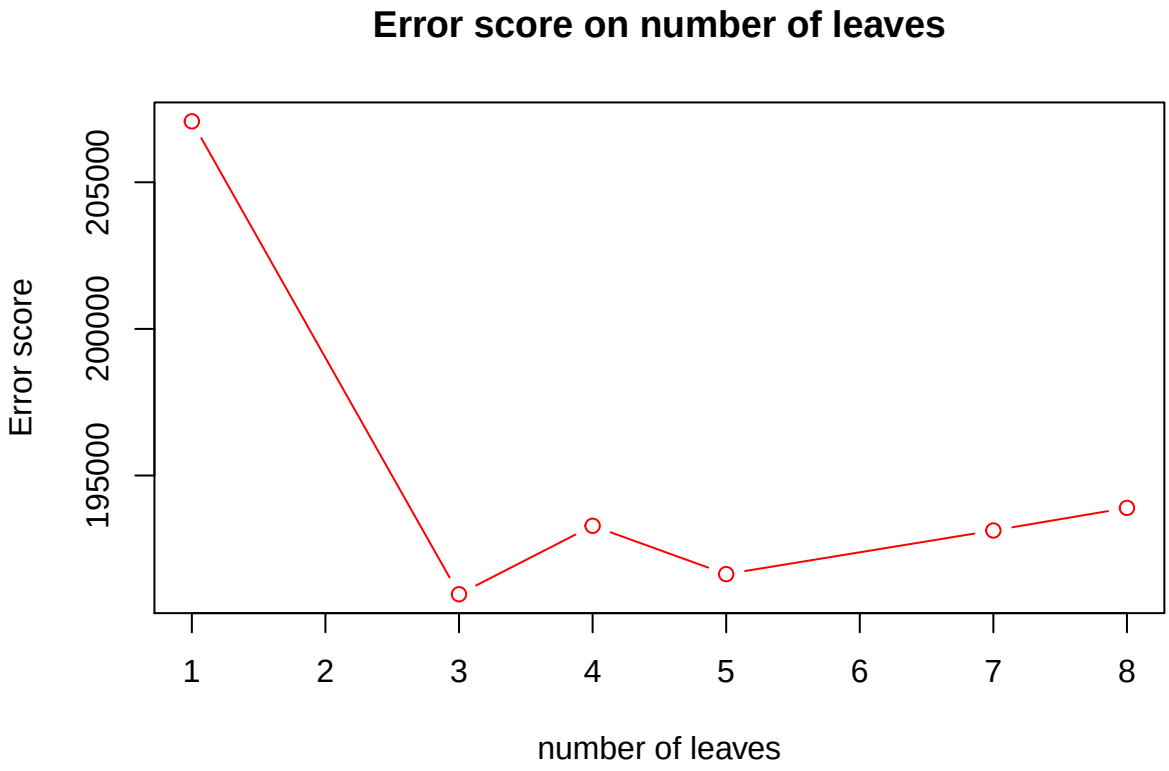
Assignment 3 - Uncertainty estimation

In this assignment we use tha data file *State.csv* containing per capita state and local public expenditures and associated state demographic and economic characteristics.

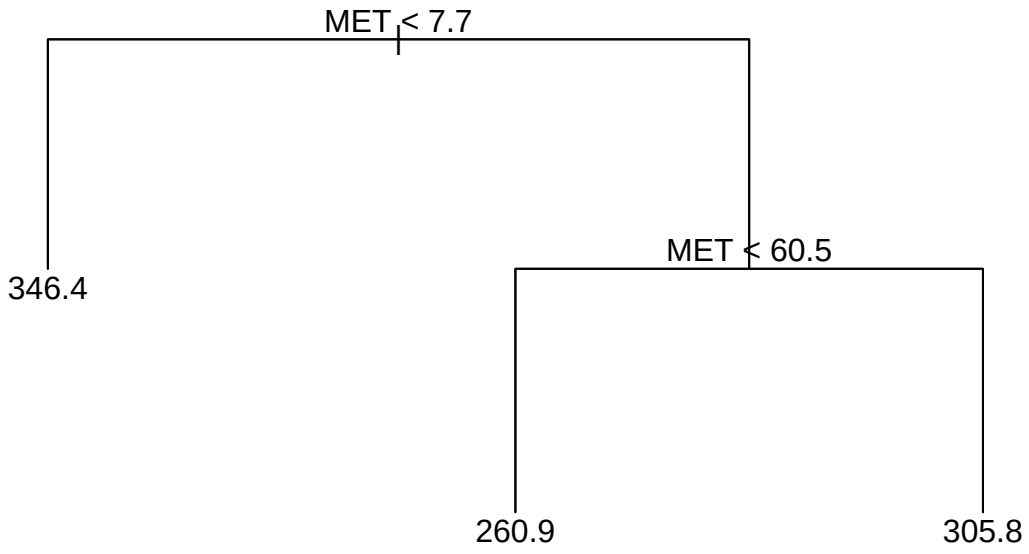
1. This first graph shows the relation between the variable **MET** and **EX**. The distribution of the data looks like a quadratic function.



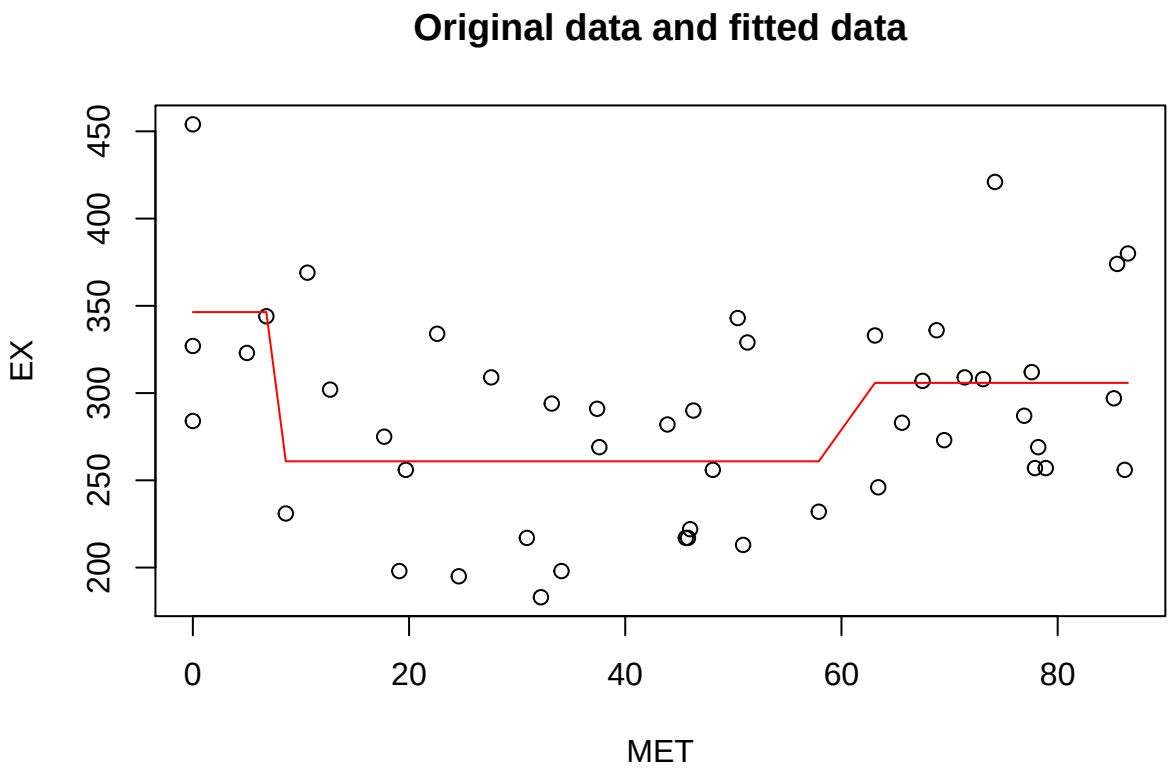
2. In this step we use the function `tree()` to fit a regression tree model with target **EX** and feature **MET** selecting the number of leaves by cross-validation. The following plot shows us that the best number of leaves is 3 (the lowest error score reached corrisponds to n°3).



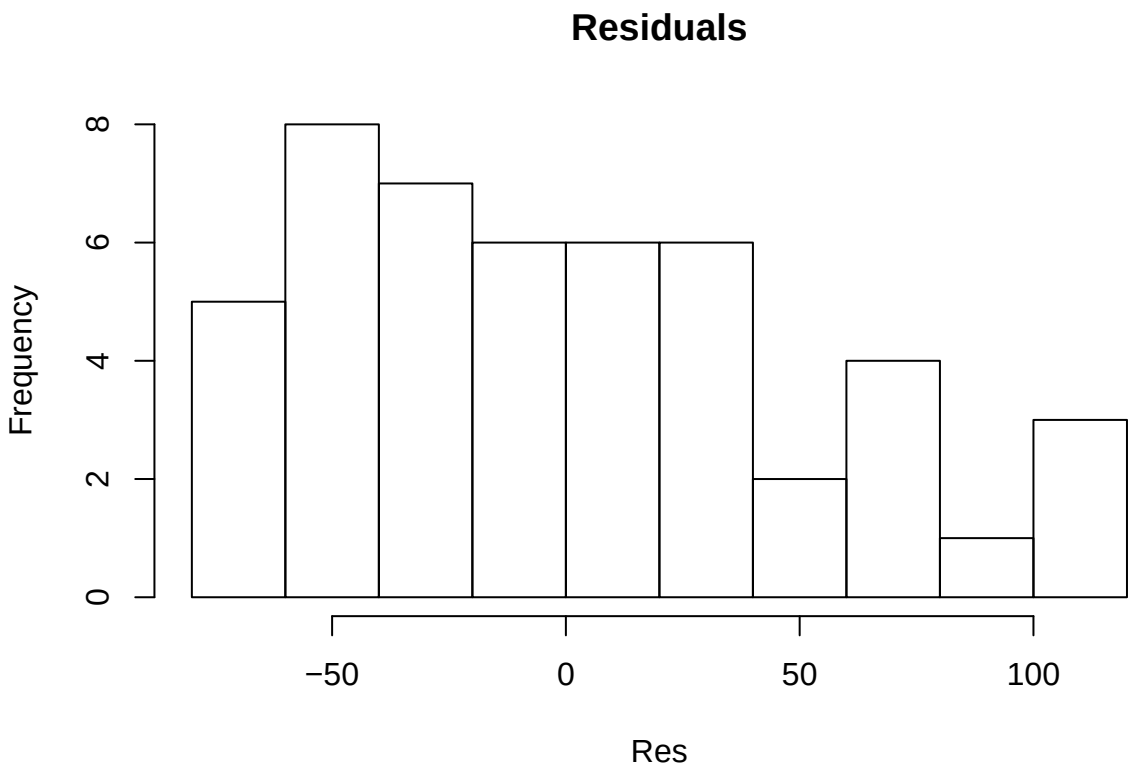
The selected tree is the following:



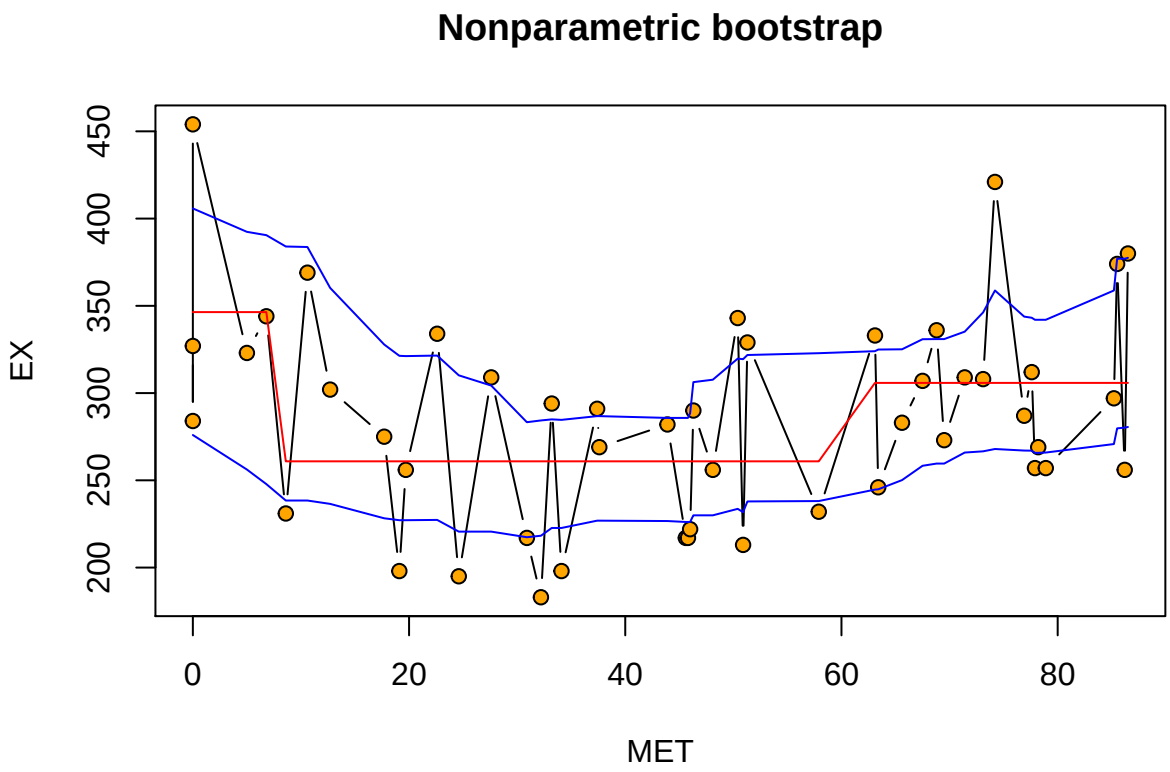
The following plot shows the original and the fitted data. The quality of the fit is not that good because the observations are very few.



The residuals are distributed as follow, as we can see they look like distribuited as a normal distribution.

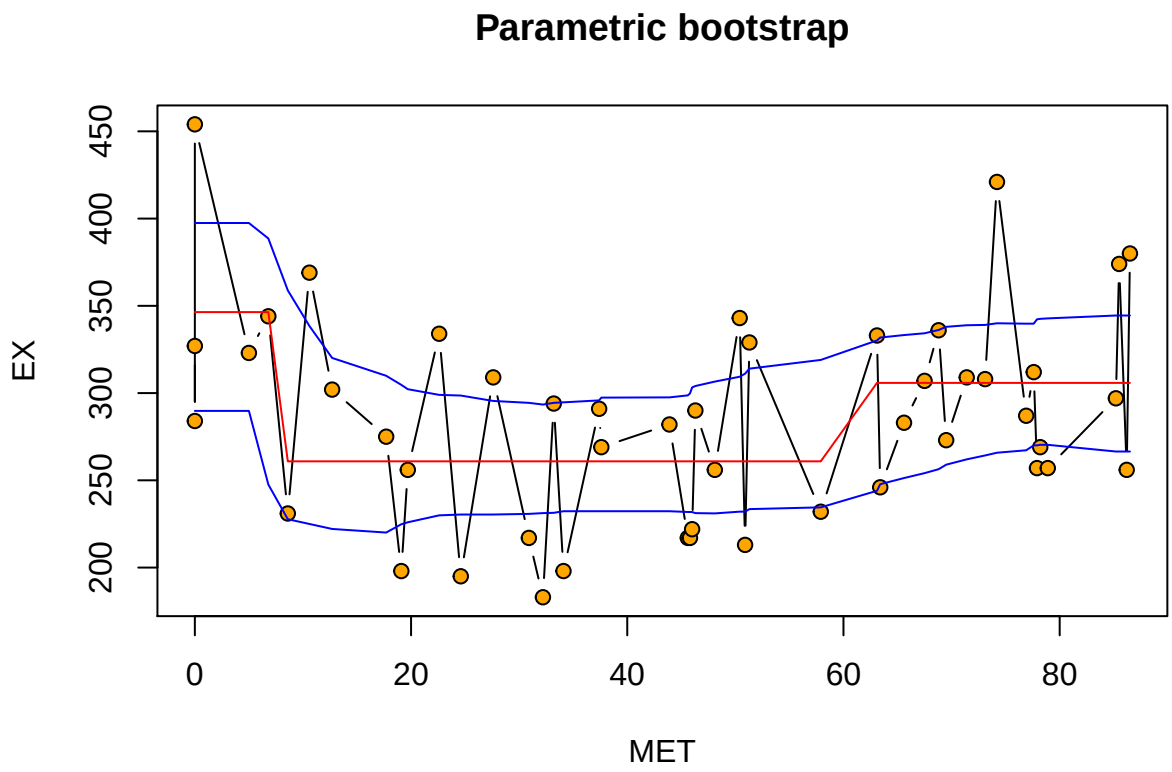


3. In this step we compute and we plot the 95% confidence bands for the regression tree model with 3 leaves (the best tree we found in step 2.) by using a non-parametric bootstrap. The confidence band looks bumpy, this is because in the non parametric bootstrap samples are drawn from a discrete set of observations so the samples understimate the amount of variation in the population sampled originally.



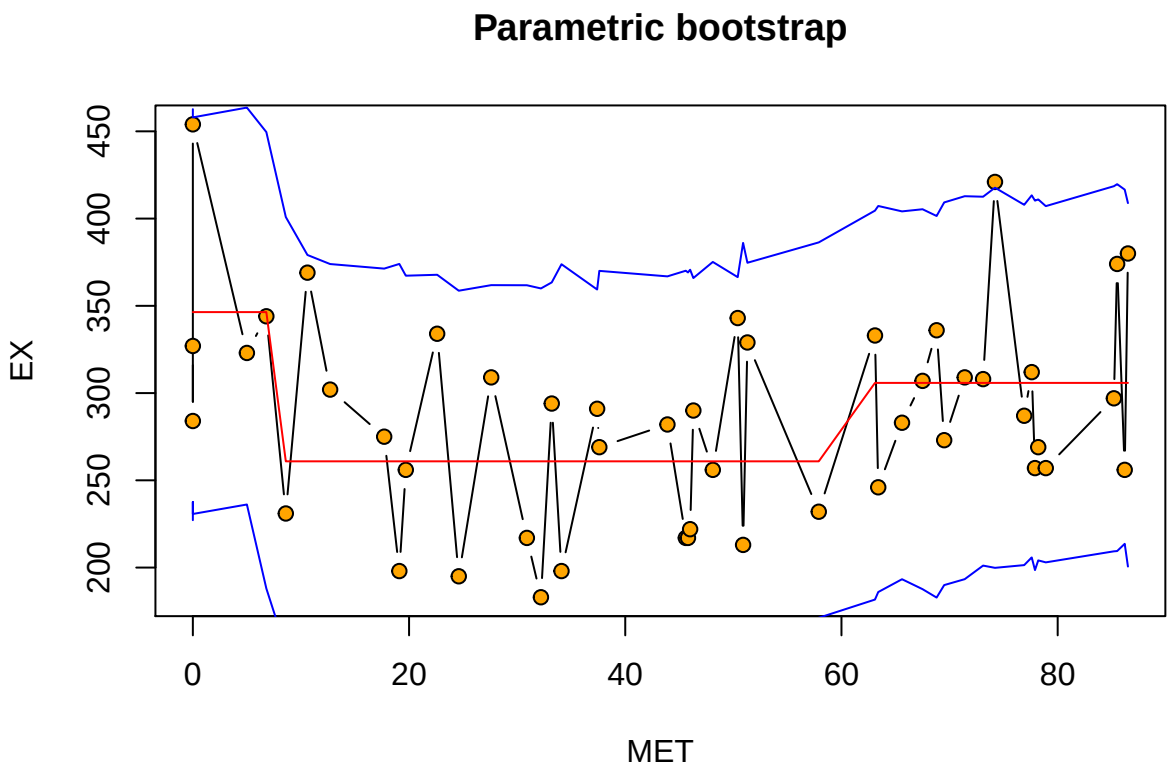
The confidence band follows the data: the band is wider in the first part when the variance is higher and it gets more narrow later. we are not assuming any distribution for out target variables so it makes sense.

4. Now we compute and plot the 95% confidence and the prediction bands by using a parametric bootstrap assuming that the response variable has a normal distribution Y has a $N(\mu_i, \sigma^2)$ where μ_i are labels in the tree leaves and σ^2 is the residual variance.



The confidence band is smoother and this is because we are assuming that our data have a normal distribution. The confidence band is more narrow.

The following plot shows the prediction band:



From the graph it doesn't look that only the 5% of the data are outside the prediction band but that less than 5% does: $1/48 = 2\%$. This may be because of the few data we have to fit the model or because we are using a normal distribution to predict that fits the data well but it's not the real distribution.

5. Considering the histogram of residuals from step 2, the few data we have and the last two steps we can conclude that the parametric bootstrap with normal distribution is not appropriate here, but the non-parametric has a better confidence band.

Appendix

```
knitr::opts_chunk$set(echo = TRUE)
#ASSIGNMENT 1.

dat<-read.csv("australian-crabs.csv",1)

#1)
plot(dat$CL,dat$RW,col=dat$sex,xlab="Carapace Length",ylab="Rear width", main="CL vs RW")

#2)

#in this function we find w_0 and w_1 for the discriminant functions.

disc_fun=function(label, X ,S){

X1=X[X==label,]

w<-length(2)

mu<-c(mean(X1[,2]),mean(X1[,3])) #class label

w1<-solve(S) %*% mu

p<-nrow(X1)/dim(X)[1]

b1<-matrix
b1= -(t(mu)%*% solve(S) %*% mu + log(p))/2

return(c(w1[1], w1[2], b1[1,1]))
}

X=data.frame(Y=dat$sex,RW=dat$RW,CL=dat$CL)

X1=X[X=="Male",]
X2=X[X=="Female",]

#covariance matrix
S=cov(X1[,-1])*dim(X1[,-1])[1]+cov(X2[,-1])*dim(X2[,-1])[1]
S=S/dim(X)[1]

#discriminant function coefficients
res1=disc_fun("Male",X,S)
res2=disc_fun("Female",X,S)

res=res1-res2

#3) classification
d=res[1]*X[,2]+res[2]*X[,3]+res[3]
Yfit=(d>0) #true female
plot(X[,3], X[,2], col=Yfit+1, xlab="CL", ylab="RW",main="Decision boundary from LDA classifier")
```

```
t_LDA=table(Yfit,dat$sex)
misclassification_LDA<-(t_LDA[1,2]+t_LDA[2,1])/sum(t_LDA)*100

abline(coef=c(-res[3]/res[1],-res[2]/res[1]),col="blue",lwd=2)
#4)
X$Y<-as.factor(X$Y)
f_LG<-glm(Y~.,data=X,family="binomial")
w<-coef(f_LG)
pred=predict(f_LG,newdata=X,type="response")
col<-vector()
col[which(pred<.5)]<-1
col[which(pred>.5)]<-2
matrix<-data.frame(pred,col)

pred[which(pred<.5)]<-0
pred[which(pred>=.5)]<-1

t_Logistic=table(pred,dat$sex)
misclassification_Logistic<-(t_Logistic[1,2]+t_Logistic[2,1])/sum(t_Logistic)*100

#new data

X=data.frame(Y=pred,RW=dat$RW,CL=dat$CL)

X1=X[which(X$Y==1),] #male
X2=X[which(X$Y==0),]

#covariance matrix
S=cov(X1[,-1])*dim(X1[,-1])[1]+cov(X2[,-1])*dim(X2[,-1])[1]
S=S/dim(X)[1]

#discriminant function coefficients
res1=disc_fun(1,X,S)
res2=disc_fun(0,X,S)

res=res1-res2

plot(X[,3], X[,2], col=col, xlab="CL", ylab="RW",main="Decision boundary from Logistic Regression")

abline(coef=c(-res[3]/res[1],-res[2]/res[1]),col="blue",lwd=2)
#-------------------------------------#
#ASSIGNMENT 2
#1)

library(readxl)
dat2<-read_excel("creditscoring.xls",1)
dat2$good_bad<-as.factor(dat2$good_bad)

n=dim(dat2)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=dat2[id,]
```

```r
id2<-(1:n)[!(1:n %in% id)]
n2=dim(dat2[-id,])[1]
id3=sample(id2, floor(n2*0.5))
test=dat2[id3,]
validation=dat2[-c(id3,id),]

#2)

library(tree)
n3=dim(train)[1]

fit=tree(good_bad~., data=train,split=c("deviance","gini"))

Yfit=predict(fit, newdata=test,type="class")
t<-table(test$good_bad,Yfit)
misclassification<-(t[1,2]+t[2,1])/sum(t)*100

Yfit_train=predict(fit,newdata=train,type="class")
t_train<-table(train$good_bad,Yfit_train)
misclassification_train<-(t_train[1,2]+t_train[2,1])/sum(t_train)*100

#3)
trainScore=rep(0,15)
testScore=rep(0,15)

for(i in 2:15) {

  prunedTree=prune.tree(fit,best=i)
  pred=predict(prunedTree, newdata=validation, type="tree")
  trainScore[i]=deviance(prunedTree)
  testScore[i]=deviance(pred)

}

plot(2:15, trainScore[2:15], type="b", col="red",ylim=c(270,570),main="Dependence of deviances on the nu
points(2:15, testScore[2:15], type="b", col="blue")

#the best model is the one with 4th leaves
finalTree=prune.tree(fit, best=4)

Yfit2=predict(finalTree, newdata=test, type="class")
t2=table(test$good_bad,Yfit2)
misclassification2<-(t2[1,2]+t2[2,1])/sum(t2)*100

plot(finalTree, main="Optimal tree")
text(finalTree,pretty=0)
summary(finalTree)

#4)
library(MASS)
library(e1071)
library(knitr)
library(kableExtra)
```

```r
fit2=naiveBayes(good_bad~., data=train)

Yfit3=predict(fit2,newdata=test)
Yfit4=predict(fit2,newdata=train)

tab_test<-table(Yfit3,test$good_bad)
tab_train<-table(Yfit4,train$good_bad)

mis_test<-(tab_test[1,2]+tab_test[2,1])/sum(tab_test)*100
mis_train<-(tab_train[1,2]+tab_train[2,1])/sum(tab_train)*100

kable(tab_train)
kable(tab_test)
#5)

set.seed(12345)

Prob_test<-predict(fit2,newdata=test,type="raw")
Prob_train<-predict(fit2,newdata=train,type="raw")

L<-matrix(nrow=2,ncol=2)
L[1,]<-c(0,1)
L[2,]<-c(10,0)

classification<-function(L,Prob){

  Y_hat<-vector()
  index<-which((Prob[,1]/Prob[,2])>(L[1,2]/L[2,1]))
  Y_hat[index]<-"bad"
  Y_hat[-index]<-"good"
  return(Y_hat)

}

Y_hat_test<-classification(L,Prob_test)
Y_hat_train<-classification(L,Prob_train)

tab_loss_test<-table(Y_hat_test,test$good_bad)
tab_loss_train<-table(Y_hat_train,train$good_bad)

kable(tab_loss_train)
kable(tab_loss_test)
#-------------------------------------#
#ASSIGNMENT 3

data_norder<-read.csv("State.csv",sep=";",dec = ",")

#1)

index<-order(data_norder$MET)
dat3<-data_norder[index,]

plot(dat3$MET,dat3$EX,type="p",xlab="MET",ylab="EX", main="MET vs EX")
```

```r
#2)

set.seed(12345)
library(tree)

dati<-data.frame(MET=dat3$MET,EX=dat3$EX)
n=nrow(dati)
fit_tree=tree(EX~MET, data=dat3,control=tree.control(nobs=n,minsize=8))

#selecting number of leaf
cv.res=cv.tree(fit_tree)
plot(cv.res$size, cv.res$dev, type="b", col="red", main="Error score on number of leaves",xlab="number

finalTree2<-prune.tree(fit_tree,best=3)
Yfit4<-predict(finalTree2)
#plotting final tree
plot(finalTree2,main="Optimal Tree")
text(finalTree2,pretty=0)
#plot original and fitted data
plot(dat3$MET,dat3$EX,type="p",xlab="MET",ylab="EX",main="Original data and fitted data")
points(dat3$MET,Yfit4,col="red",type="l")

#plot histogram of residuals
hist((dat3$EX-Yfit4),xlab="Res",ylab="Frequency",main="Residuals")
#3)

library(boot)

f=function(data, ind){

  data1=data[ind,]# extract bootstrap sample
  res=tree(EX~MET, data=data1,control=tree.control(nobs=nrow(dati),minsize=8))
  final<-prune.tree(res,best=3)
  priceP=predict(final,newdata=dat3)
  return(priceP)

  }

res_noparam<-boot(dat3, f, R=1000) #make bootstrap
e<-envelope(res_noparam)

plot(dat3$MET,dat3$EX,type="b",xlab="MET",ylab="EX",pch=21, bg="orange",main="Nonparametric bootstrap")
points(dat3$MET,res_noparam$t0,type="l",col="red")

#plot cofidencebands
points(dat3$MET,e$point[2,], type="l", col="blue")
points(dat3$MET,e$point[1,], type="l", col="blue")

#4)

rng=function(data, finalTree2) {
  data1<-data.frame(MET=data$MET,EX=data$EX) #database non ordered
  n=nrow(dati)
```

```
  data1$EX=rnorm(n,predict(finalTree2,newdata = data1),sd(dati$EX-Yfit4))
  return(data1)
  }

f1=function(data1){

  res=tree(EX~MET, data=data1,control=tree.control(nobs=nrow(dati),minsize=8))
  final<-prune.tree(res,best=3)
  priceP=predict(final,newdata=dati)
  return(priceP)

  }

res_param=boot(dat3, statistic=f1, R=1000, mle=finalTree2,ran.gen=rng, sim="parametric")

e2=envelope(res_param)

plot(dat3$MET,dat3$EX,type="b",xlab="MET",ylab="EX",pch=21, bg="orange",main="Parametric bootstrap")
points(dat3$MET,res_param$t0,type="l",col="red")

#plot cofidencebands
points(dat3$MET,e2$point[2,], type="l", col="blue")
points(dat3$MET,e2$point[1,], type="l", col="blue")



f2=function(data1){

  res=tree(EX~MET, data=data1,control=tree.control(nobs=nrow(dati),minsize=8))
  final<-prune.tree(res,best=3)
  priceP=predict(final,newdata=dati)
  n=nrow(dati)
  predictedP=rnorm(n,priceP,sd(resid(finalTree2)))
  return(predictedP)

  }

res_Pred=boot(dat3, statistic=f2, R=1000, mle=finalTree2,ran.gen=rng, sim="parametric")

e3=envelope(res_Pred)

plot(dat3$MET,dat3$EX,type="b",xlab="MET",ylab="EX",pch=21, bg="orange",main="Parametric bootstrap")
points(dat3$MET,res_param$t0,type="l",col="red")

#plot prediction band
points(dat3$MET,e3$point[2,], type="l", col="blue")
points(dat3$MET,e3$point[1,], type="l", col="blue")
```

Lab 3 Introduction to Machine Learning

Alessia De Biase, Alejandro Garcia, Balaji Ramkumar

15 dicembre 2017

1. Kernel Methods

In this assignment we implement a kenel method to predict the hourly temperatures for a date and place in Sweden. We use the files *stations.csv* and *temps50k.csv* wich contains informations about weather stations and temperature measurements in the stations at different days and times. The place we will consider in Sweden has **latitude=58.265** and **longitude=14.826**, the day we are considering is **2013-11-04** .

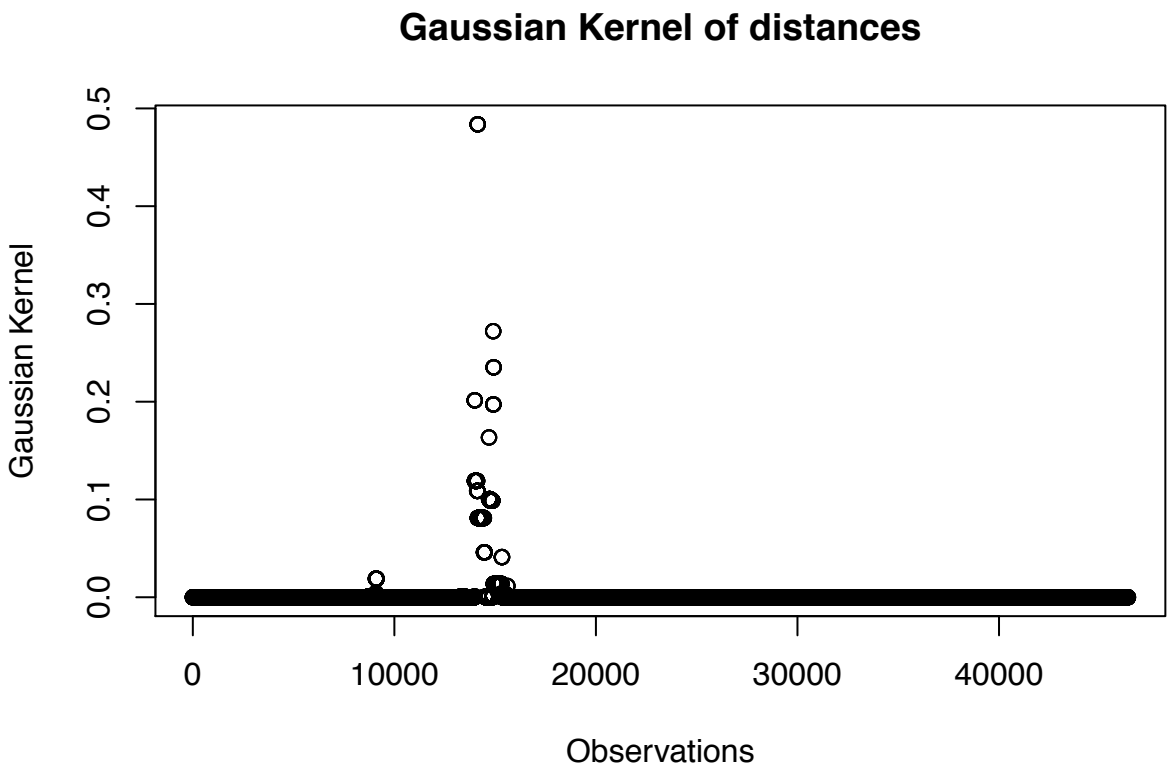
In the first estimation, our kernel will be the sum of three Gaussian kernels:

1. *Distance from a station to the point of interest:*

To calculate the distance between the point of interest and all the stations we use the function `distHaversine()` from the library `geosphere`.

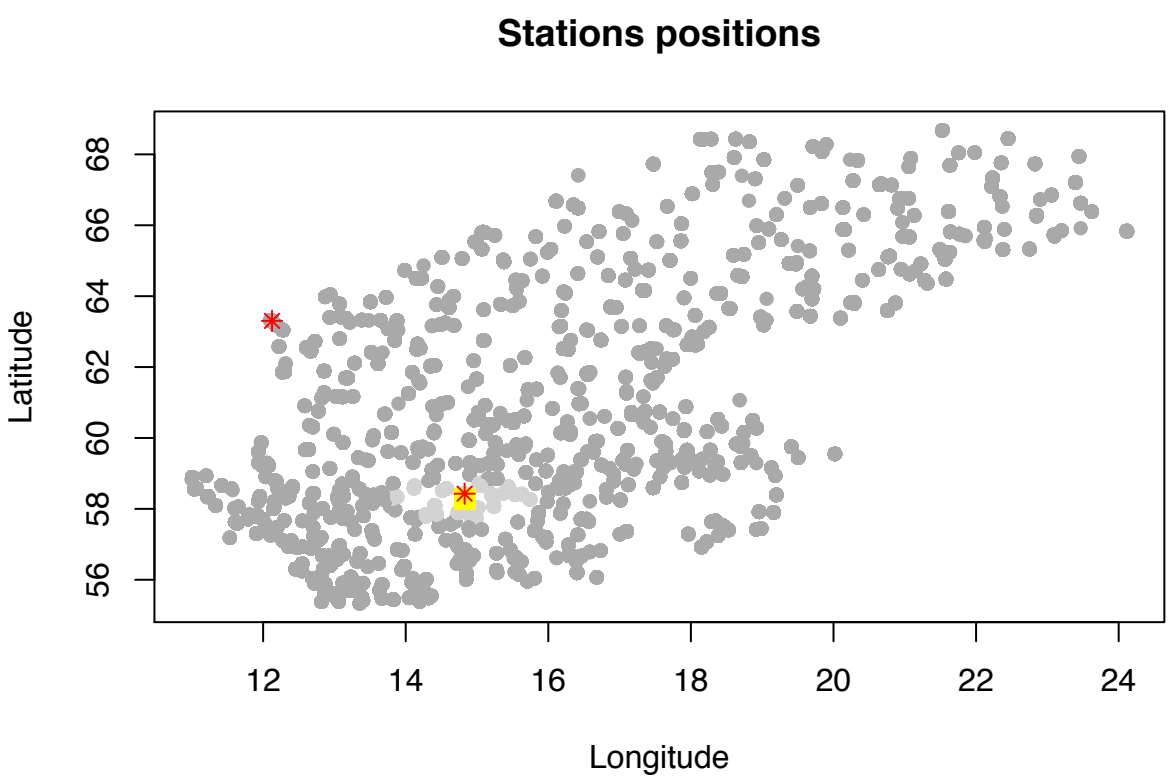
The following plot shows the first Gaussian Kernel:

```
distances<-vector()

for(i in 1:length(st$latitude)){
  distances[i] <- distHaversine(c(b,a),c(st$longitude[i],st$latitude[i]))
}

#smoothing coefficient
h_distance<-15000

d<-gaussian_kernel(distances,h_distance)

plot(1:length(d),d,xlab="Observations",ylab="Gaussian Kernel",main="Gaussian Kernel of distances")
```



To check if the kernel assigns higher values to the closest points and lower to the farest we plot all the observations, the point of interest (in yellow) and the closest and the further points (in red) found by the distance function. In darker grey we colored all stations considered far from our point of interest and with light grey all the close ones. By the graphic representation looks like the built kernel is reasonable.

```
col<-vector()
col[which(d<0.0001)]<-"dark grey"
col[which(d>=0.0001)]<-"light grey"

plot(st$longitude,st$latitude,col=col,pch=16,xlab="Longitude",ylab="Latitude",main="Stations positions"
points(b,a,col="yellow",pch=15,cex=1.5)
#closest point
points(st$longitude[which.max(d)],st$latitude[which.max(d)],col="red",pch=8)
#further point
points(st$longitude[which.min(d)],st$latitude[which.min(d)],col="red",pch=8)
```



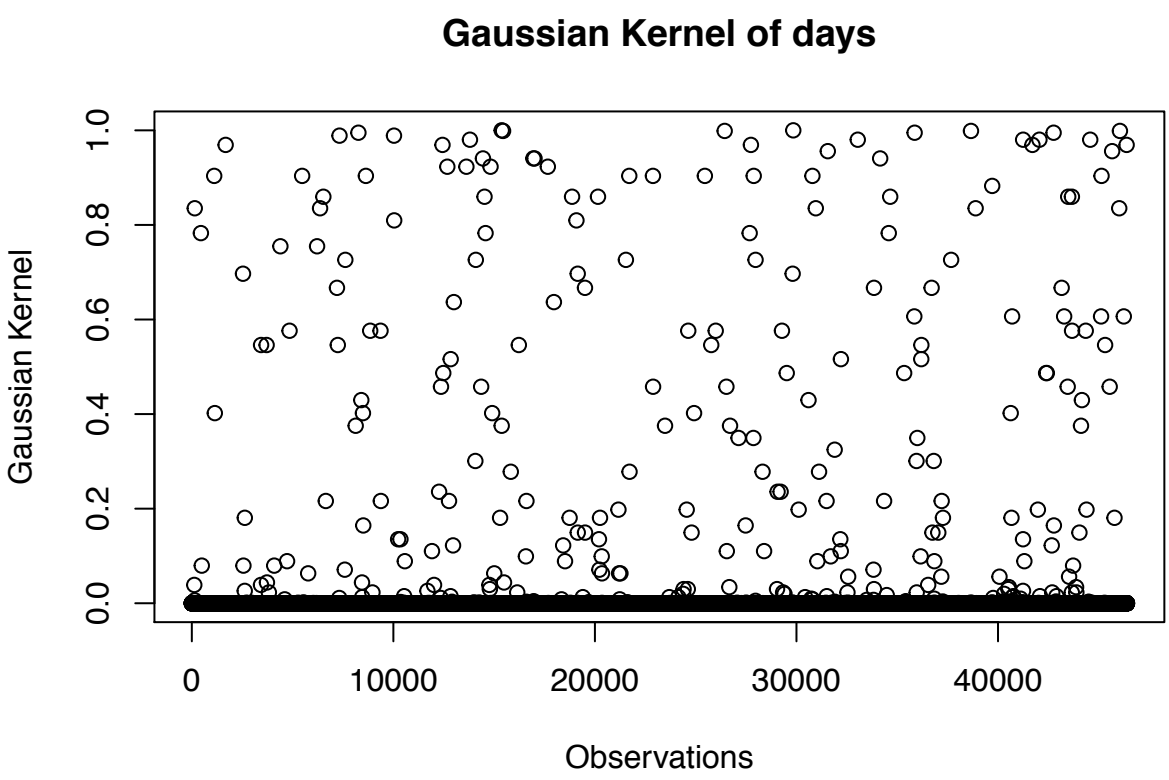
As we can see from the plot the closest points gain an higher value compared to the further, in terms of geographical distance this is obvious

2. *Distance between the day a temperature measurement was made and the day of interest:*

To calculate the distance between days we used the function `difftime()` with units equals to days.

The following plot shows the second Gaussian Kernel:

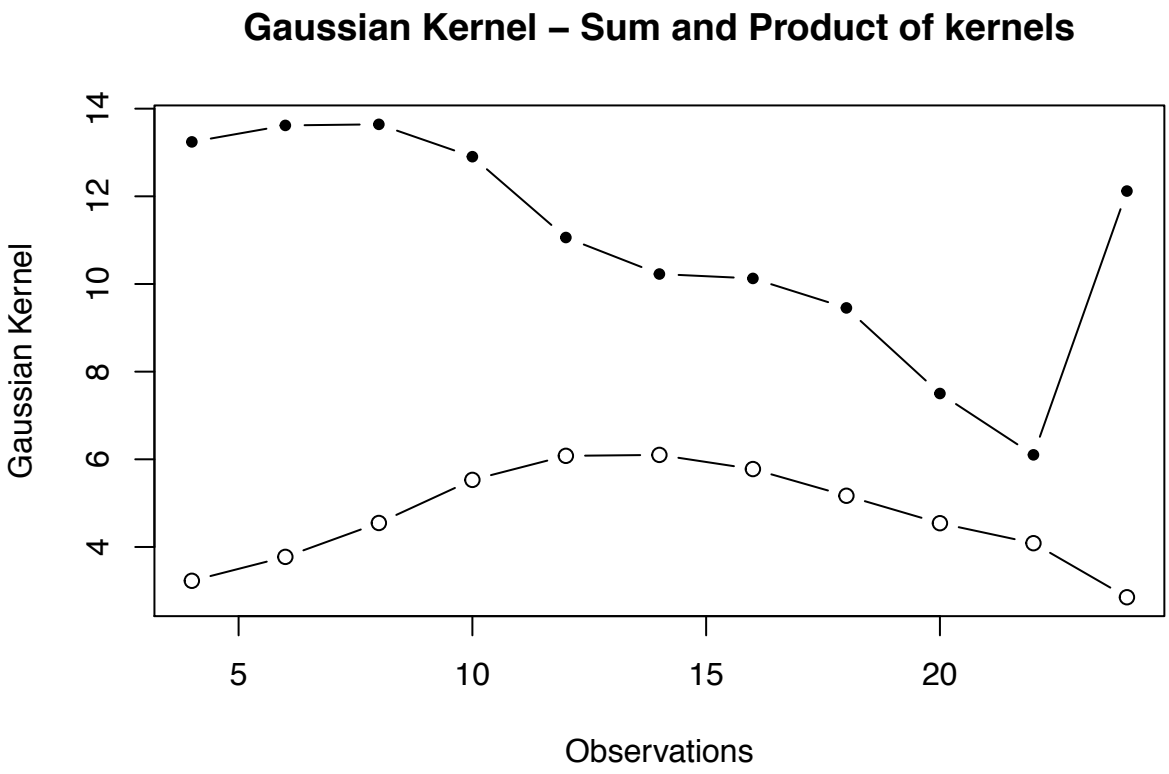
```
date<-as.Date(date,'%Y-%m-%d')

st$date<-as.Date(st$date,'%Y-%m-%d')

time_dist <- abs(as.numeric(difftime(rep(date, length(st$date)) ,st$date , units = c("days"))))

h_date <- 20

d1<-gaussian_kernel(time_dist,h_date)

plot(1:length(d1),d1,xlab="Observations",ylab="Gaussian Kernel",main="Gaussian Kernel of days")
```



3. *Distance between the hour of the day a temperature measurement was made and the hour of interest:*

The third distance is between the hour of the day a temperature measurement was made and the hours of interest. In this assignment we want to predict temperatures from 4 am to 24 pm in an interval of 2 hours.

The following plot shows the sum (empty dots) and the product (filled dots) of the three different kernels we found previously. Both the graphs look to be reasonable and show the common trend of temperature during the day, low during night and higher during daily time, with a maximum reached around 12:00. The difference between the two kernels is about the variation of temperature between hours. Summing the kernels the variation of temperature is lower compared to the product of the kernel, but this is obvious because the product of two positive number compared to the sum of them is a much stronger coefficient.



In calculating the kernels we tried different smoothing coefficients, until we got the one we thought it was the best fit for the kernel. It is important to say that the smoothing coefficient is very important, as it is in charge of selecting the number of important values for the kernel.

2. Neural Networks

In this second assignment we train a neural network to learn the trigonometric sine function. We first sample 50 points uniformly at random in the interval [0,10] and then we apply the function to each point. We separate the data into train and validation set and we use the second one to stop the gradient descent and to avoid overfitting. In the function `neuralnet()` from the library `nauralnet` we test different thresholds and then to find the most appropriate we calculate the MSE for the training and the validation sets.

```
#-------------------------------------#
#ASSIGNMENT 2

library(neuralnet)
set.seed(1234567890)
Var <- runif(50, 0, 10)
trva <- data.frame(Var, Sin=sin(Var))

tr <- trva[1:25,] # Training
va <- trva[26:50,] # Validation

winit <- runif(100, -1, 1)

MSE_va<-vector()
MSE_tr<-vector()

for(i in 1:10) {
    nn <- neuralnet(Sin~Var,data=tr,hidden=c(10),threshold = i/1000,startweights=winit)
    results<-compute(nn,va$Var)
    prediction = results$net.result

    MSE_tr[i]<-sum((compute(nn,tr$Var)$net.result-tr$Sin)^2)/nrow(tr)
    MSE_va[i]<-sum((prediction-va$Sin)^2)/nrow(va)

}

plot(MSE_va,ylim=c(0,max(MSE_va)),type="b",col="red",main="MSE for training and test sets")
lines(MSE_tr,type="b")
```

The red line of the following plot represents the trend of the validation, the black one the trend of the training MSE (always lower than the validation, of course).

From the graph we consider the threshold corrisponding to 4/1000 because is the minimum value of MSE for the validation set.

The final neural network learned with the chosen threshold is the following:

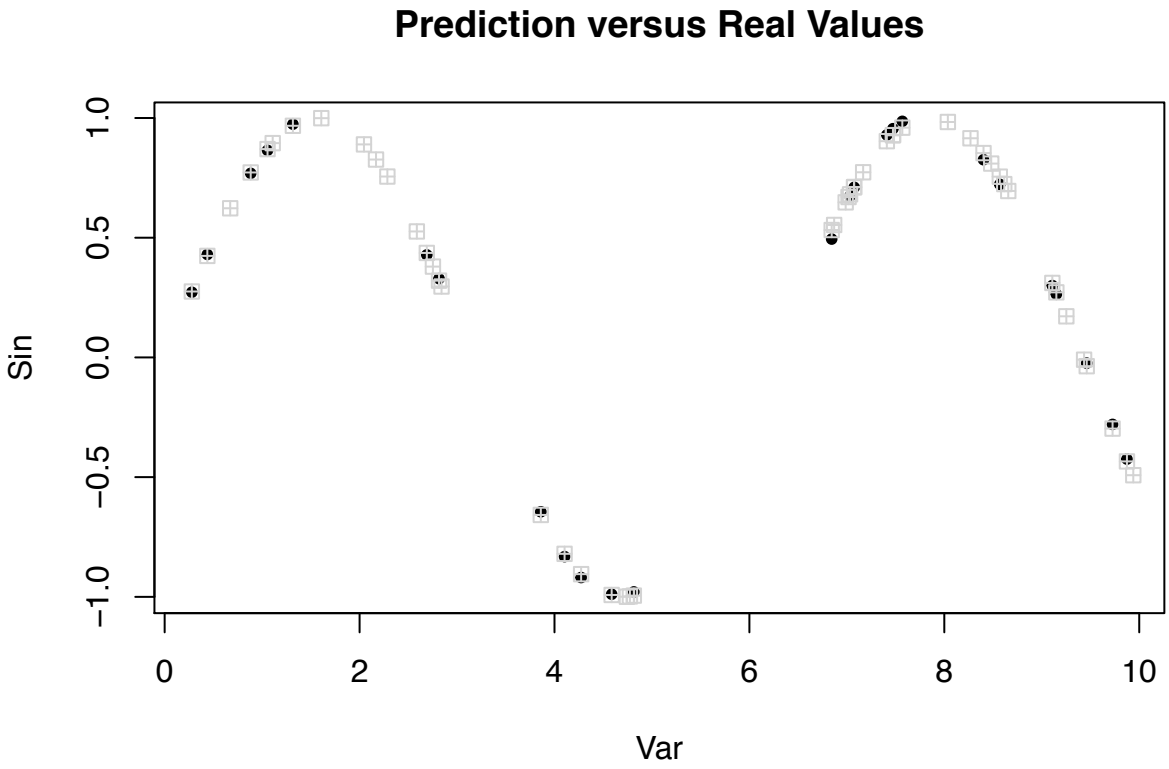
```
plot(nn <- neuralnet(Sin~Var,data=tr,hidden=c(10),threshold = which.min(MSE_va)/1000,startweights=winit)
```

In this last plot the black dots are from the data while the grey ones are the ones predicted with the neural network with the choosen threshold. As we can see the plot represents in a very accurate way the sin function we wanted to learn.

```
plot(prediction(nn)$rep1,main="Prediction versus Real Values",pch=20)
```

```
## Data Error:  0;
```

```
points(trva, col = "light grey",pch=12)
```


Prediction versus Real Values
Sin
1.0
0.5
0.0
−0.5
−1.0
0
2
4
6
8
10
Var

Appendix

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

#ASSIGNMENT 1.

library(geosphere)

set.seed(1234567890)
stations <- read.csv("stations.csv",fileEncoding = "Latin1")
temps <- read.csv("temps50k.csv",fileEncoding = "Latin1")
st <- merge(stations,temps,by="station_number")

#Latitude and longitude of the place we want estimate the temperature
a <- 58.265
b <- 14.826

# The date to predict
date <- "2013-11-04"

#all measurements posterior to the date chosen have been filtered out
pre_date<-as.numeric(difftime(rep(date, length(st$date)) ,st$date , units = c("days")))
st<-st[which(pre_date>=0),]

#gaussian kernel function
gaussian_kernel<-function(dist,h){
  k<-exp(-dist^2/(2*h^2))
  return(k)
}



distances<-vector()

for(i in 1:length(st$latitude)){
  distances[i] <- distHaversine(c(b,a),c(st$longitude[i],st$latitude[i]))
}

#smoothing coefficient
h_distance<-15000

d<-gaussian_kernel(distances,h_distance)

plot(1:length(d),d,xlab="Observations",ylab="Gaussian Kernel",main="Gaussian Kernel of distances")


col<-vector()
col[which(d<0.0001)]<-"dark grey"
col[which(d>=0.0001)]<-"light grey"

plot(st$longitude,st$latitude,col=col,pch=16,xlab="Longitude",ylab="Latitude",main="Stations positions"
points(b,a,col="yellow",pch=15,cex=1.5)
#closest point
```

```r
points(st$longitude[which.max(d)],st$latitude[which.max(d)],col="red",pch=8)
#further point
points(st$longitude[which.min(d)],st$latitude[which.min(d)],col="red",pch=8)


date<-as.Date(date,'%Y-%m-%d')

st$date<-as.Date(st$date,'%Y-%m-%d')

time_dist <- abs(as.numeric(difftime(rep(date, length(st$date)) ,st$date , units = c("days"))))

h_date <- 20

d1<-gaussian_kernel(time_dist,h_date)

plot(1:length(d1),d1,xlab="Observations",ylab="Gaussian Kernel",main="Gaussian Kernel of days")



ora<-as.numeric(factor(c("04:00:00","06:00:00","08:00:00","10:00:00","12:00:00","14:00:00","16:00:00","

time<-as.numeric(st$time)

temp <- vector(length=length(ora))
temp2 <- vector(length=length(ora))

for(i in 1:length(ora)){

 hour_dist <- abs(ora[i]-time)

 h_time <-20

 d2<-gaussian_kernel(hour_dist,h_time)

  #sum the kernel
  tot<-d+d1+d2 #d2 sarà un vettore diverso ogni volta che cambio l'orario di riferimento

  #multipl. the kernel
  tot2<-d*d1*d2


  temp[i]<-t(tot)%*%st$air_temperature/sum(tot)

  temp2[i]<-t(tot2)%*%st$air_temperature/sum(tot2)

}

orario<-c(4,6,8,10,12,14,16,18,20,22,24)
plot(orario,temp,type="b",xlab="Observations",ylab="Gaussian Kernel",main="Gaussian Kernel - Sum and Pr
lines(orario,temp2,type="b",pch=20)


#-----------------------------------#
```

```r
#ASSIGNMENT 2

library(neuralnet)
set.seed(1234567890)
Var <- runif(50, 0, 10)
trva <- data.frame(Var, Sin=sin(Var))

tr <- trva[1:25,] # Training
va <- trva[26:50,] # Validation

winit <- runif(100, -1, 1)

MSE_va<-vector()
MSE_tr<-vector()

for(i in 1:10) {
    nn <- neuralnet(Sin~Var,data=tr,hidden=c(10),threshold = i/1000,startweights=winit)
    results<-compute(nn,va$Var)
    prediction = results$net.result

    MSE_tr[i]<-sum((compute(nn,tr$Var)$net.result-tr$Sin)^2)/nrow(tr)
    MSE_va[i]<-sum((prediction-va$Sin)^2)/nrow(va)

}

plot(MSE_va,ylim=c(0,max(MSE_va)),type="b",col="red",main="MSE for training and test sets")
lines(MSE_tr,type="b")

plot(nn <- neuralnet(Sin~Var,data=tr,hidden=c(10),threshold = which.min(MSE_va)/1000,startweights=winit)

plot(prediction(nn)$rep1,main="Prediction versus Real Values",pch=20)
points(trva, col = "light grey",pch=12)
```

Lab 2 Block 2

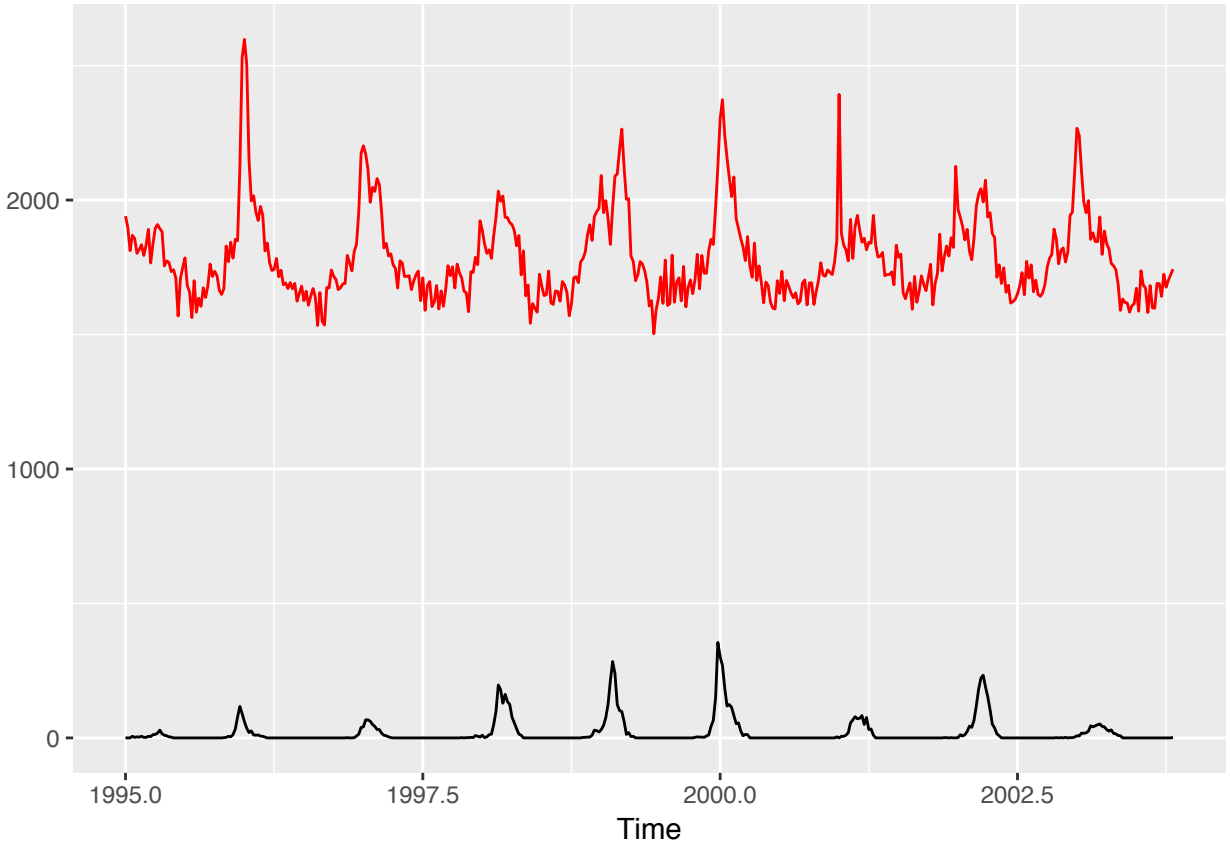
Alejandro Garcia, Balaji Ramkumar,Alessia de Biase

15 de diciembre de 2017

Assignment 1. Using GAM and GLM to examine the mortality rates

This part of the assignment is to understand the implementation of Generalized Additive Model [GAM] and Generalised Linear Model [GLM]. To perform this question we use a data called as *Influenza*.

1) We create a time series plot of infulenza and mortality with time as x asis.



From the plot, we can understand that that everytime there is an increase in the rate of influenza, the mortality rate increases.

2) In this part of the question, we use GAM function from mgcv package and create a model in which Mortality is normally distributed and modelled as a linear function of Year and spline function of Week.And the parameters are selected by cross - validation.

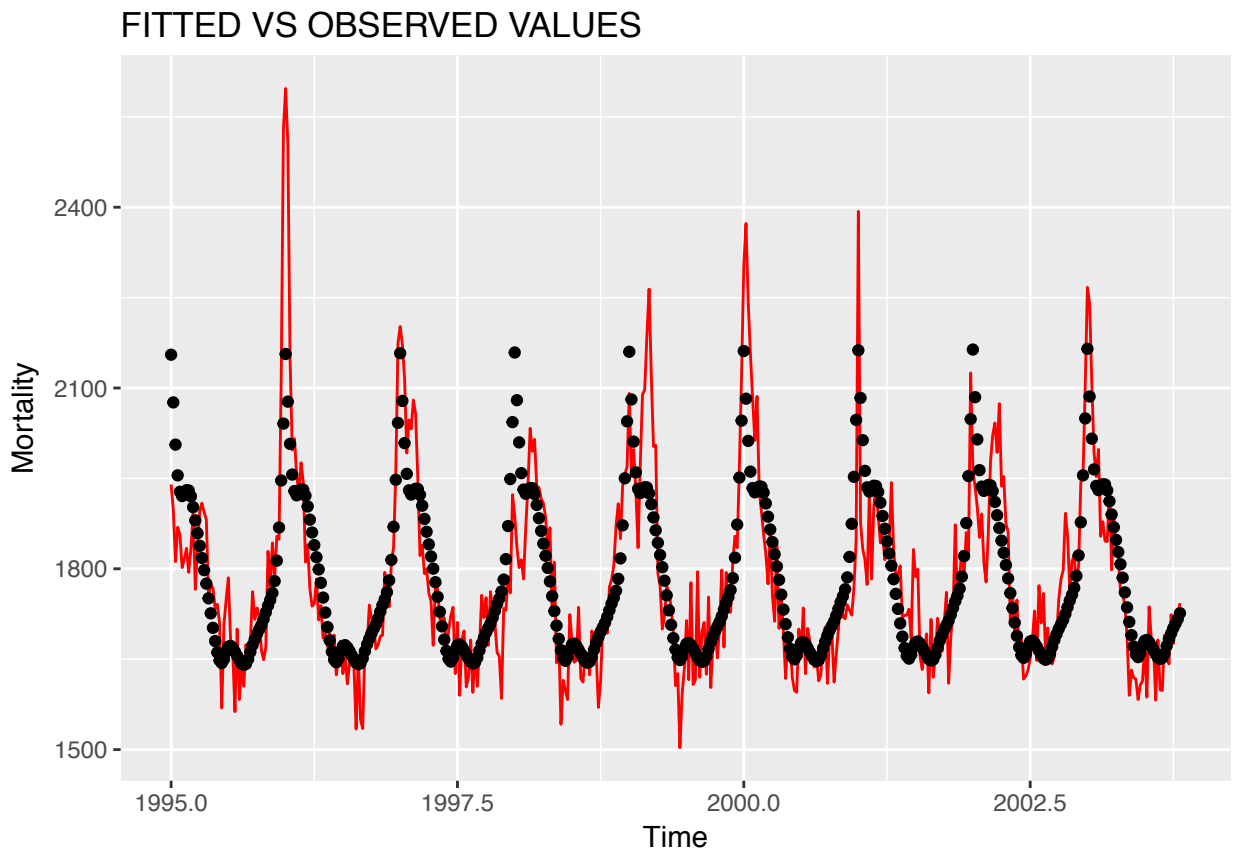
```
classifier <- gam(Mortality ~ Year + s(Week,k=50),data = data )
classifier$coefficients
```

```
## (Intercept)        Year   s(Week).1   s(Week).2   s(Week).3   s(Week).4
##  -680.62643     1.23286    16.99435 -1299.13810  -111.78012   765.87563
##   s(Week).5   s(Week).6   s(Week).7   s(Week).8   s(Week).9  s(Week).10
##  -233.49208   745.72432   167.01091  -638.39383   249.57351  -631.81413
```

```
##  s(Week).11  s(Week).12  s(Week).13  s(Week).14  s(Week).15  s(Week).16
##  -181.98409   560.95684  -181.75047  -540.19723  -178.15056   502.41844
##  s(Week).17  s(Week).18  s(Week).19  s(Week).20  s(Week).21  s(Week).22
##   191.47920  -521.38746  -188.08784  -504.94972   192.45194   513.38960
##  s(Week).23  s(Week).24  s(Week).25  s(Week).26  s(Week).27  s(Week).28
##   191.66244  -507.80552   189.06695   509.86054   193.53311  -504.45162
##  s(Week).29  s(Week).30  s(Week).31  s(Week).32  s(Week).33  s(Week).34
##  -195.00270  -508.28893  -194.75458   507.52209   193.20985  -506.54504
##  s(Week).35  s(Week).36  s(Week).37  s(Week).38  s(Week).39  s(Week).40
##  -195.36146  -506.23128  -194.66513  -508.23614  -194.90030   507.17629
##  s(Week).41  s(Week).42  s(Week).43  s(Week).44  s(Week).45  s(Week).46
##  -194.28299  -506.53906   194.30386   506.02477   194.05958  -504.71420
##  s(Week).47  s(Week).48  s(Week).49
##   193.04201 -2608.07862   126.11865
```

Probilistic model: $y = -680.62643 + 1.233 * x1 + s(Week) + \epsilon \; N(0, \sigma^2)$

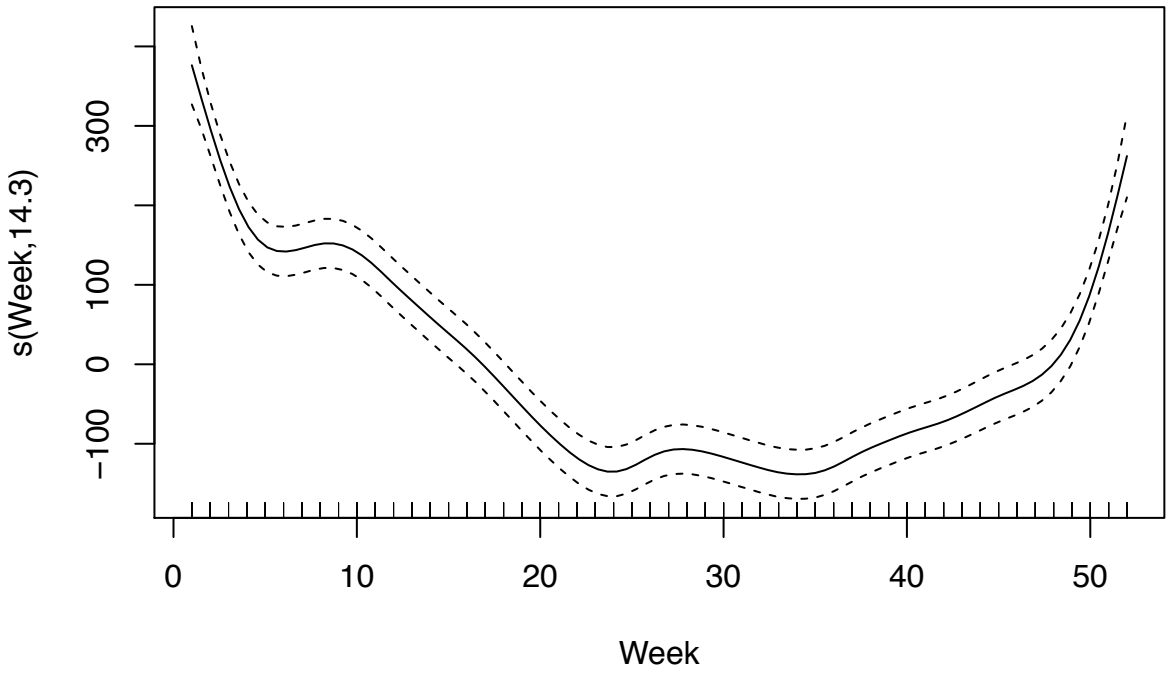
3) In this part of the question, we plot the predicted and observed mortality against time.



The quality of the fit is good becayse the fitted values are almost the same as the observed values.

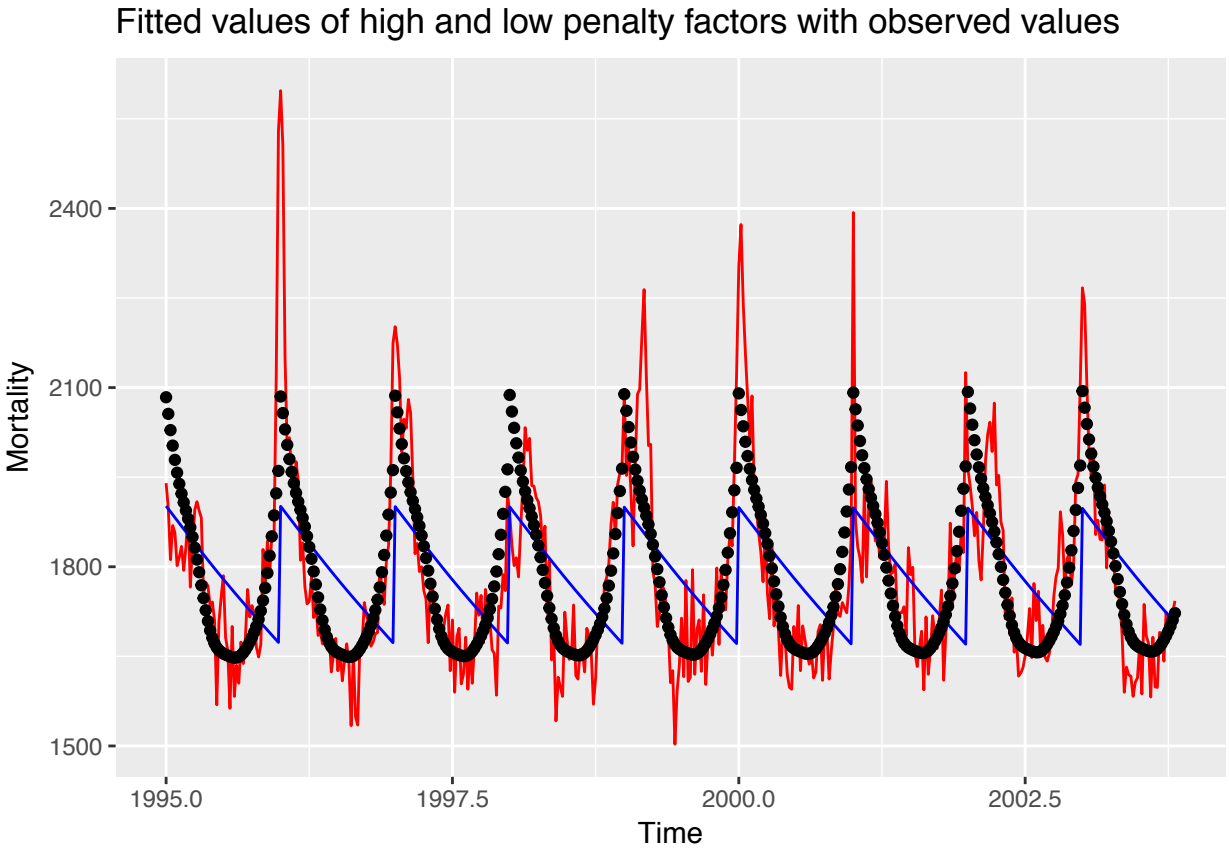
```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## Mortality ~ Year + s(Week, k = 50)
##
```

```
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -680.626   3367.919  -0.202    0.840
## Year           1.233      1.685   0.732    0.465
##
## Approximate significance of smooth terms:
##          edf Ref.df     F p-value
## s(Week) 14.3  17.84 53.93  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.677   Deviance explained = 68.8%
## GCV = 8709.1  Scale est. = 8399.7    n = 459
```



we can say that the graph exhibits a seasonal distribution. Initially the graph comes down and once in reaches close to week 35, It raises again.

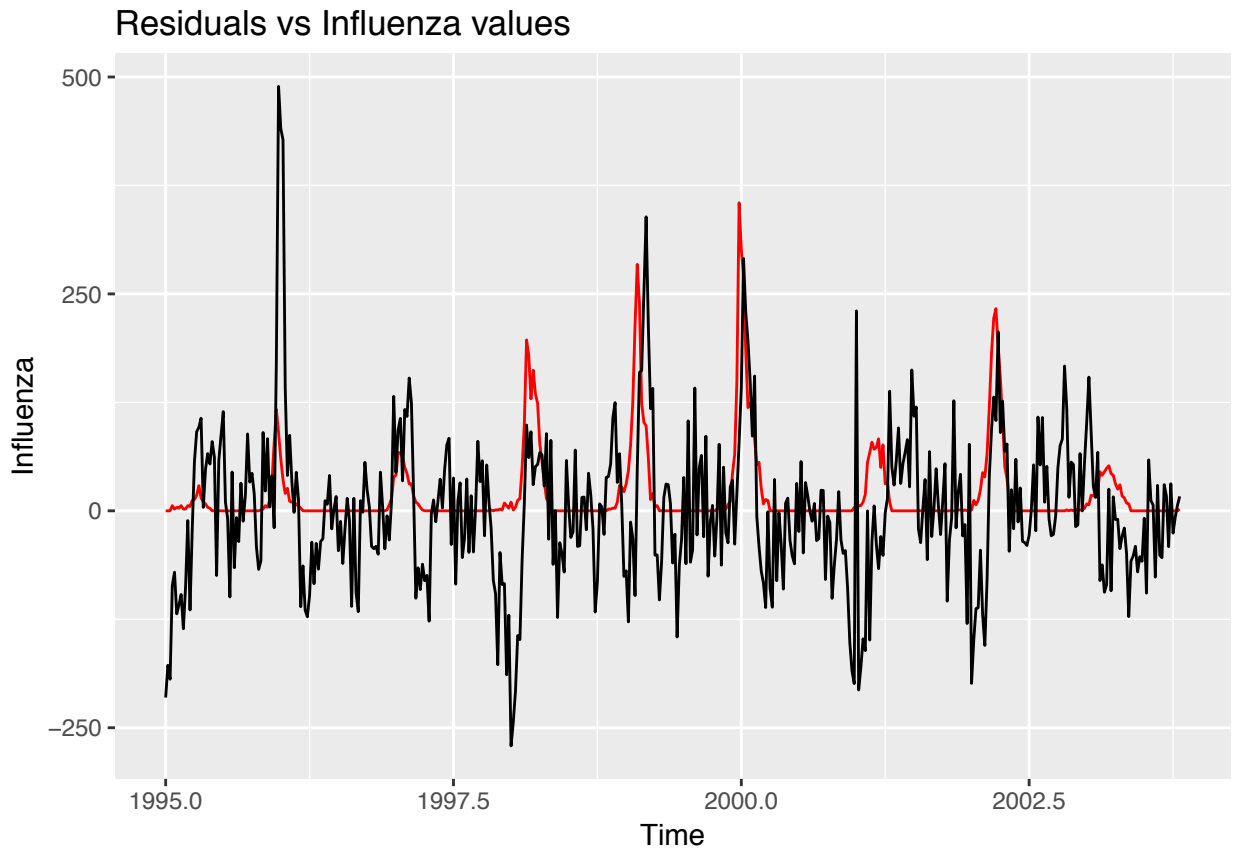
4) In this part of the question, we make a plot of predicted and observed mortality rates for high and low penalty factors.



From the graph we can understand that the graph of lower penalty factors fits well with the observed values when compared to the graph of higher penalty factors.

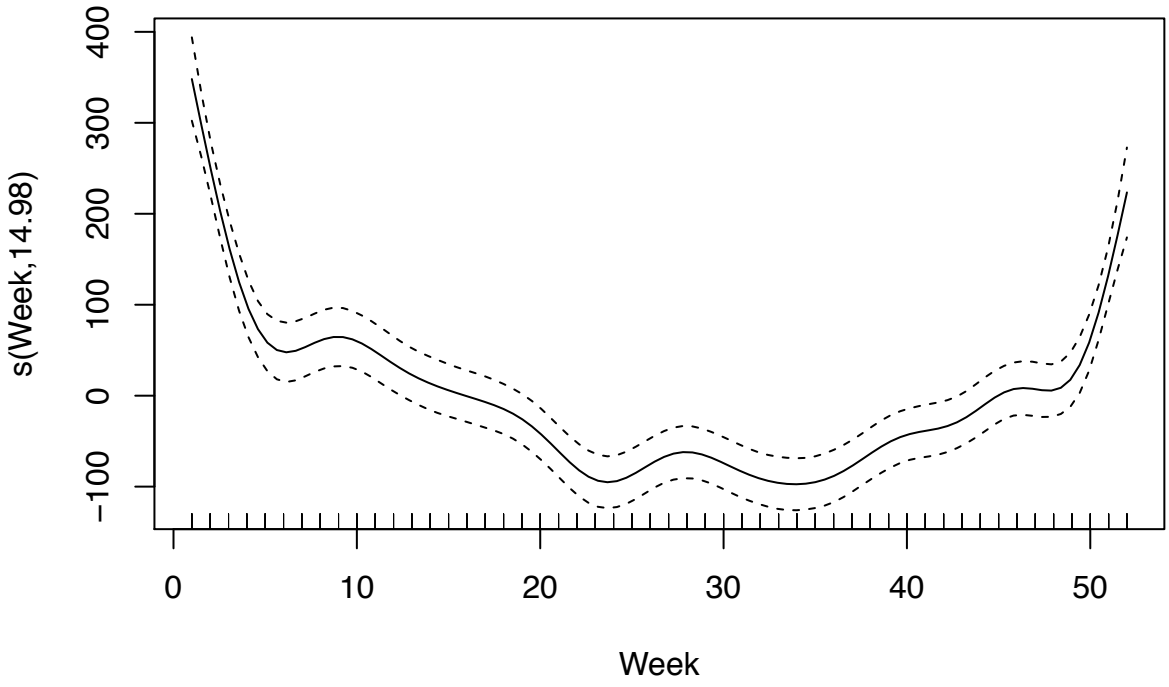
We find the degree of freedom decreases with the increase in penalty factors.

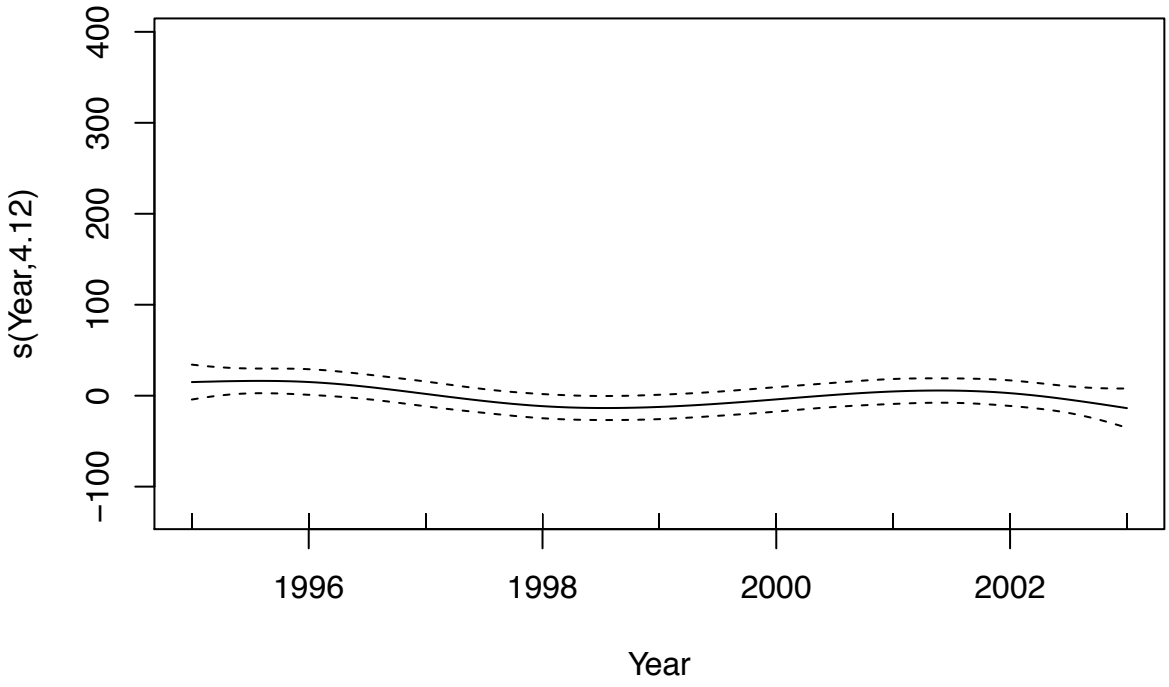
5) We plot the residuals and the influenza values against time.

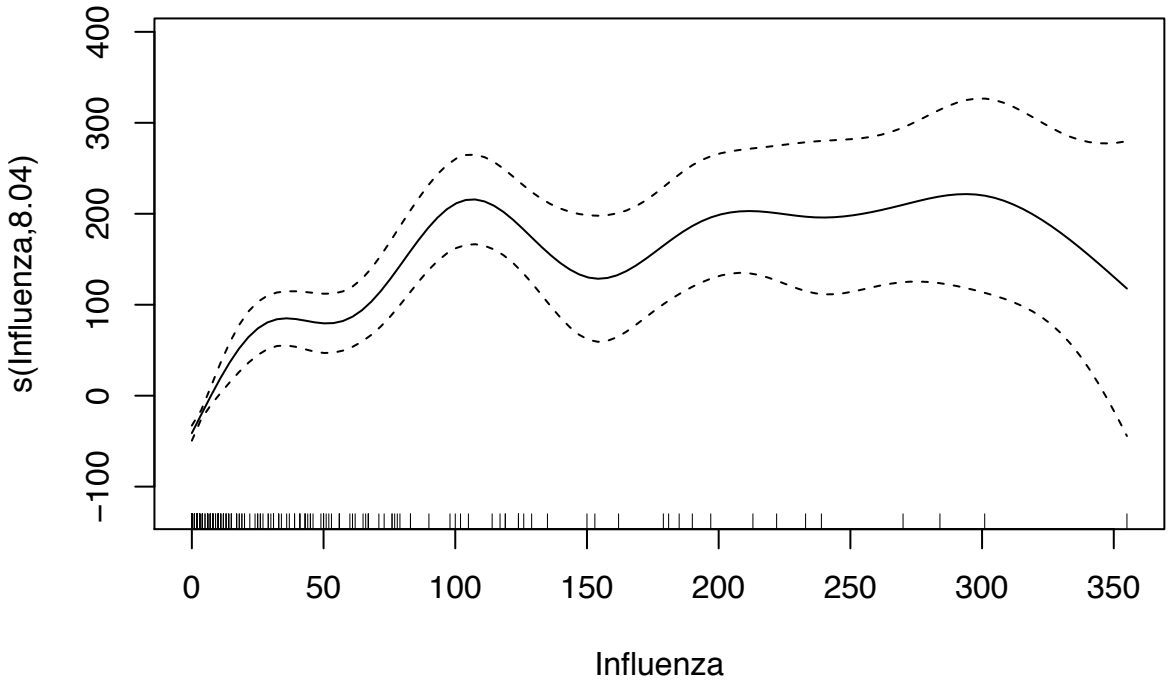


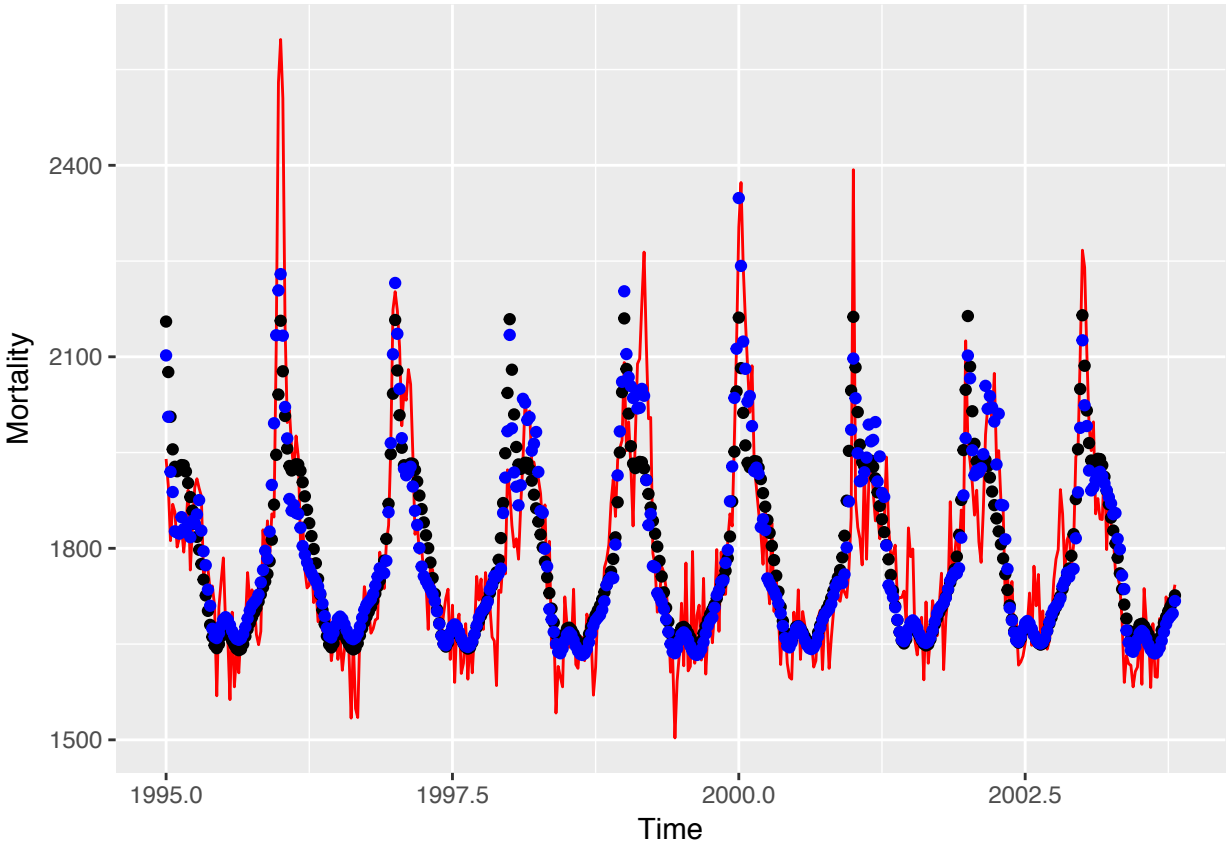
Yes, The residuals are correlated with the outbreaks of influenza.

6) We fit a gam model in which mortality is fitted as aditivee function of splines of year,week,number of confirmed cases of influenza.


s(Week,14.98)
−100
0
100
200
300
400
0
10
20
30
40
50
Week


s(Year,4.12)
−100
0
100
200
300
400
1996
1998
2000
2002
Year


s(Influenza,8.04)
−100
0
100
200
300
400
0
50
100
150
200
250
300
350
Influenza



Yes the values are better than the previous model because,the predicted values reach higher values which are almost same as the observed values.

Assignment 2. High-dimensional methods

In this assignment we are asked to use the data.csv containing information about 64 e-mails which were manually collected from DBWorld mailing list. Which we will use for high dimensional procedures.

1)

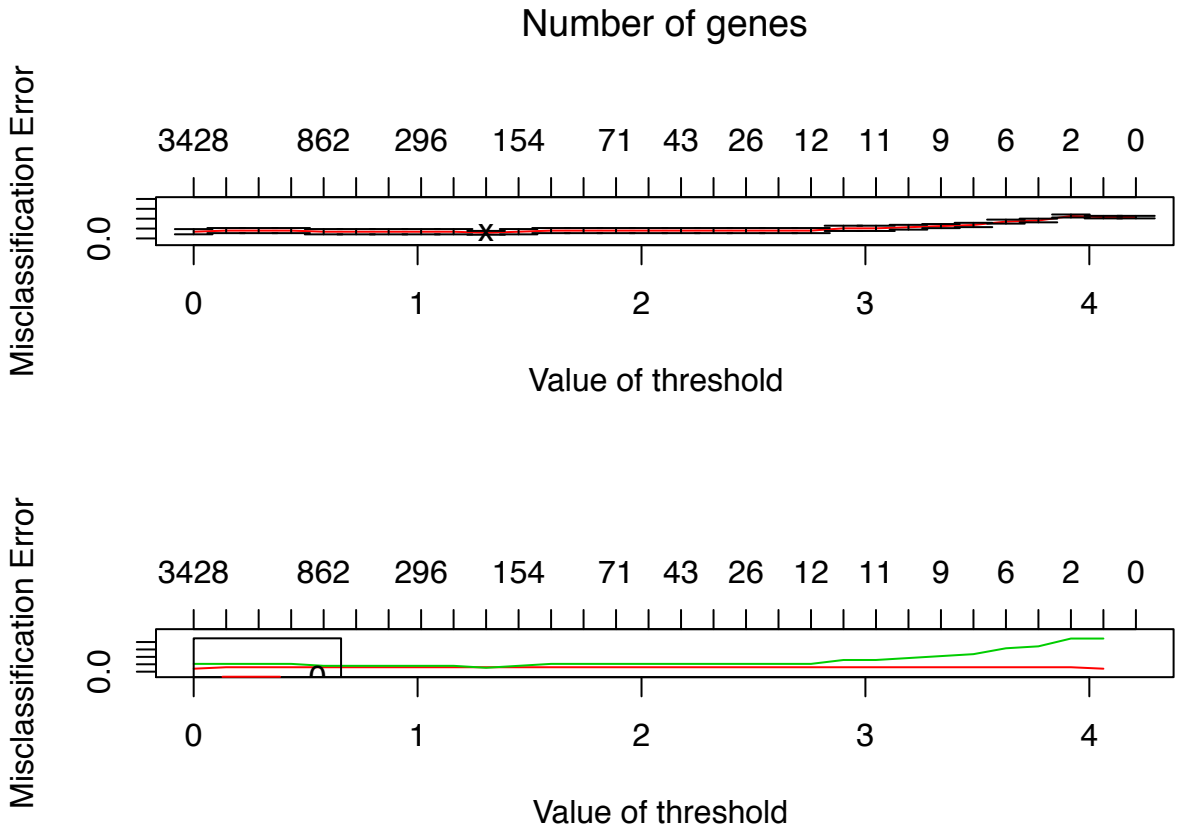
The first step is to divide the data into train and test (70/30) without scaling. Afterwards we perform nearest shrunken centroids classification of training in which the threshold is chosen by cross-validation.

We can see that the cross-validation shows that the best threshold for the shrunken centroids is the one with the lowest missclassification which is 1.3.

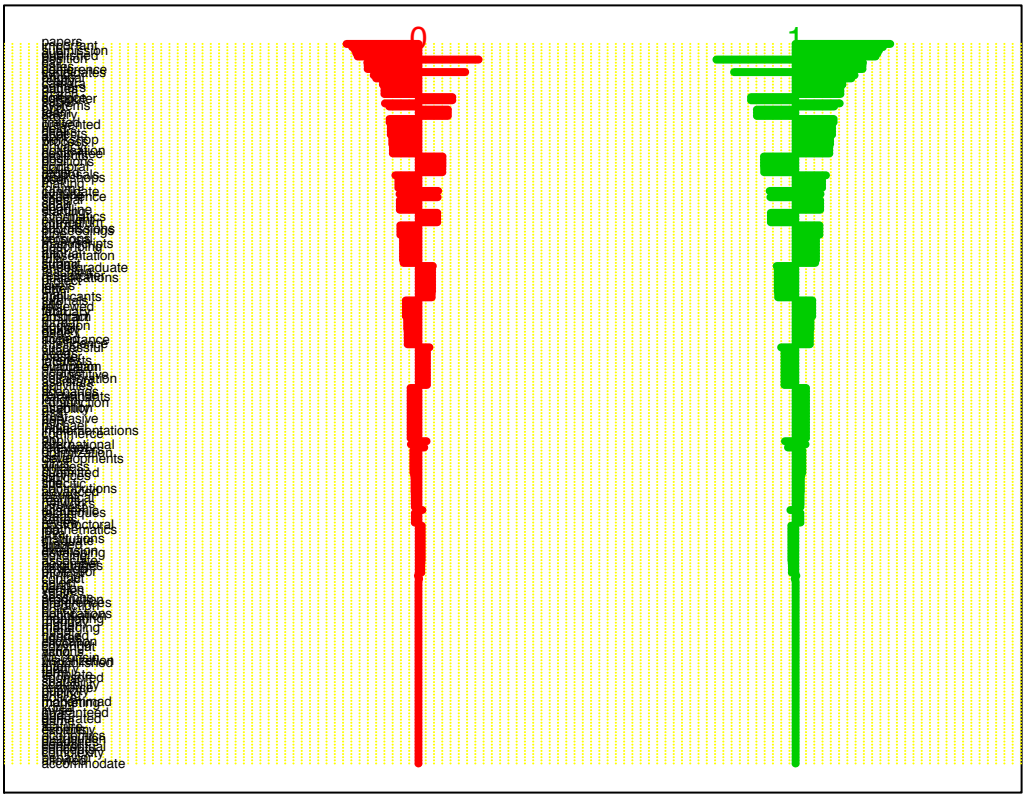
```
## 1234567891011121314151617181920212223242526272829 30
```

```
## 12Fold 1 :1234567891011121314151617181920212223242526272829 30
## Fold 2 :1234567891011121314151617181920212223242526272829 30
## Fold 3 :1234567891011121314151617181920212223242526272829 30
## Fold 4 :1234567891011121314151617181920212223242526272829 30
## Fold 5 :1234567891011121314151617181920212223242526272829 30
## Fold 6 :1234567891011121314151617181920212223242526272829 30
## Fold 7 :1234567891011121314151617181920212223242526272829 30
## Fold 8 :1234567891011121314151617181920212223242526272829 30
## Fold 9 :1234567891011121314151617181920212223242526272829 30
```

```
## Fold 10 :12345678910111213141516171819202122324252627282930
```



When having the threshold decided, we plot the centroid, the plot shows the features selected by the algorithm with the centroid shrunken by the algorithm, we can appreciate how the centroids are getting smaller while going down, which means that they are not as relevant. In addition, the number of genes with threshold equals 1.3 are 231.




```
##         id   0-score 1-score
##    [1,] 3036 -0.3822 0.5029
##    [2,] 2049 -0.3527 0.4641
##    [3,] 4060 -0.3376 0.4441
##    [4,] 1262 -0.3309 0.4354
##    [5,] 3364 -0.3231 0.4252
##    [6,] 3187 0.3188  -0.4195
##    [7,] 596  -0.2725 0.3585
##    [8,] 869  -0.2706 0.356
##    [9,] 1045 -0.2706 0.356
##   [10,] 607  0.2476  -0.3258
##   [11,] 4282 -0.2383 0.3136
##   [12,] 2990 -0.2254 0.2966
##   [13,] 599  -0.1896 0.2495
##   [14,] 3433 -0.1896 0.2495
##   [15,] 389  -0.1815 0.2389
##   [16,] 2588 -0.1815 0.2389
##   [17,] 3022 -0.1815 0.2389
##   [18,] 850  0.1793  -0.2359
##   [19,] 3725 0.1793  -0.2359
##   [20,] 3035 -0.1785 0.2349
##   [21,] 4129 -0.1559 0.2051
##   [22,] 3125 0.1559  -0.2051
##   [23,] 4177 0.1556  -0.2047
##   [24,] 3671 0.1556  -0.2047
##   [25,] 2974 -0.1542 0.2029
```

```
##  [26,] 2463 -0.1542 0.2029
##  [27,] 329  -0.148  0.1947
##  [28,] 681  -0.148  0.1947
##  [29,] 1891 -0.148  0.1947
##  [30,] 3243 -0.148  0.1947
##  [31,] 283  -0.14   0.1842
##  [32,] 4628 -0.14   0.1842
##  [33,] 3286 -0.14   0.1842
##  [34,] 3274 -0.1368 0.18
##  [35,] 810  -0.1368 0.18
##  [36,] 2889 -0.1368 0.18
##  [37,] 1233 0.1272  -0.1674
##  [38,] 3188 0.1272  -0.1674
##  [39,] 3191 0.1272  -0.1674
##  [40,] 3312 0.1272  -0.1674
##  [41,] 3891 0.1265  -0.1664
##  [42,] 3458 0.1265  -0.1664
##  [43,] 3324 -0.1231 0.162
##  [44,] 1643 -0.1077 0.1417
##  [45,] 2561 -0.1077 0.1417
##  [46,] 3090 -0.1077 0.1417
##  [47,] 4629 -0.1077 0.1417
##  [48,] 606  0.1041  -0.137
##  [49,] 2058 -0.1012 0.1332
##  [50,] 1501 0.1012  -0.1332
##  [51,] 3952 -0.1    0.1316
##  [52,] 680  -0.1    0.1316
##  [53,] 3836 -0.1    0.1316
##  [54,] 1061 -0.0998 0.1314
##  [55,] 1007 0.0995  -0.1309
##  [56,] 1477 0.0995  -0.1309
##  [57,] 2103 0.0995  -0.1309
##  [58,] 3992 0.0995  -0.1309
##  [59,] 2295 -0.0971 0.1278
##  [60,] 4061 -0.0971 0.1278
##  [61,] 2305 -0.097  0.1276
##  [62,] 3285 -0.097  0.1276
##  [63,] 92   -0.0832 0.1094
##  [64,] 1127 -0.0832 0.1094
##  [65,] 2583 -0.0832 0.1094
##  [66,] 3323 -0.0832 0.1094
##  [67,] 4500 -0.0832 0.1094
##  [68,] 1698 -0.0832 0.1094
##  [69,] 3241 -0.0832 0.1094
##  [70,] 4364 -0.0832 0.1094
##  [71,] 4062 -0.0796 0.1048
##  [72,] 4039 0.0757  -0.0996
##  [73,] 740  0.0721  -0.0949
##  [74,] 2438 0.0721  -0.0949
##  [75,] 2442 0.0721  -0.0949
##  [76,] 3311 0.0721  -0.0949
##  [77,] 3383 0.0721  -0.0949
##  [78,] 3559 0.0721  -0.0949
##  [79,] 4176 0.0721  -0.0949
```

```
##  [80,] 4402 0.0721  -0.0949
##  [81,] 267  0.0701  -0.0923
##  [82,] 2553 0.0701  -0.0923
##  [83,] 63   -0.0681 0.0896
##  [84,] 1563 -0.0681 0.0896
##  [85,] 1594 -0.0681 0.0896
##  [86,] 3589 -0.0681 0.0896
##  [87,] 3882 -0.0681 0.0896
##  [88,] 4365 -0.0681 0.0896
##  [89,] 3301 -0.0612 0.0805
##  [90,] 1636 -0.061  0.0802
##  [91,] 1072 -0.061  0.0802
##  [92,] 386  -0.061  0.0802
##  [93,] 2198 -0.0586 0.0771
##  [94,] 3021 -0.0586 0.0771
##  [95,] 3386 -0.0586 0.0771
##  [96,] 76   -0.0583 0.0767
##  [97,] 2150 -0.0583 0.0767
##  [98,] 4075 0.0579  -0.0762
##  [99,] 107  0.0447  -0.0589
## [100,] 336  0.0447  -0.0589
## [101,] 776  0.0447  -0.0589
## [102,] 831  0.0447  -0.0589
## [103,] 1088 0.0447  -0.0589
## [104,] 1450 0.0447  -0.0589
## [105,] 1456 0.0447  -0.0589
## [106,] 1542 0.0447  -0.0589
## [107,] 2170 0.0447  -0.0589
## [108,] 2613 0.0447  -0.0589
## [109,] 2837 0.0447  -0.0589
## [110,] 4529 0.0447  -0.0589
## [111,] 363  -0.0429 0.0564
## [112,] 879  -0.0429 0.0564
## [113,] 2433 -0.0429 0.0564
## [114,] 3051 -0.0429 0.0564
## [115,] 3514 -0.0429 0.0564
## [116,] 3711 -0.0429 0.0564
## [117,] 4449 -0.0429 0.0564
## [118,] 501  -0.0429 0.0564
## [119,] 803  -0.0429 0.0564
## [120,] 2046 -0.0429 0.0564
## [121,] 2082 -0.0429 0.0564
## [122,] 2690 -0.0429 0.0564
## [123,] 2877 -0.0429 0.0564
## [124,] 3118 -0.0429 0.0564
## [125,] 4342 -0.0429 0.0564
## [126,] 4451 -0.0429 0.0564
## [127,] 4452 -0.0429 0.0564
## [128,] 272  0.0425  -0.0559
## [129,] 2175 -0.0408 0.0537
## [130,] 3515 0.0302  -0.0397
## [131,] 172  -0.0284 0.0373
## [132,] 1149 -0.0284 0.0373
## [133,] 2219 -0.0284 0.0373
```

```
## [134,] 2964 -0.0284 0.0373
## [135,] 2984 -0.0284 0.0373
## [136,] 2887 -0.0284 0.0373
## [137,] 4605 -0.0284 0.0373
## [138,] 4064 -0.028  0.0369
## [139,] 3800 -0.0238 0.0313
## [140,] 134  -0.0222 0.0292
## [141,] 919  -0.0222 0.0292
## [142,] 3957 -0.0222 0.0292
## [143,] 4268 -0.0222 0.0292
## [144,] 4281 -0.0222 0.0292
## [145,] 2220 -0.0211 0.0277
## [146,] 2847 -0.0211 0.0277
## [147,] 3582 -0.0211 0.0277
## [148,] 4181 -0.0211 0.0277
## [149,] 2167 -0.0204 0.0268
## [150,] 67   0.0204  -0.0268
## [151,] 2005 -0.0203 0.0267
## [152,] 4185 -0.0203 0.0267
## [153,] 3588 -0.0203 0.0267
## [154,] 3794 -0.0203 0.0267
## [155,] 579  0.017   -0.0223
## [156,] 1147 0.017   -0.0223
## [157,] 1524 0.017   -0.0223
## [158,] 1591 0.017   -0.0223
## [159,] 1702 0.017   -0.0223
## [160,] 1797 0.017   -0.0223
## [161,] 2141 0.017   -0.0223
## [162,] 2251 0.017   -0.0223
## [163,] 2278 0.017   -0.0223
## [164,] 2619 0.017   -0.0223
## [165,] 3194 0.017   -0.0223
## [166,] 340  0.017   -0.0223
## [167,] 2894 0.0156  -0.0205
## [168,] 1144 0.0148  -0.0195
## [169,] 2392 0.0148  -0.0195
## [170,] 3295 0.0148  -0.0195
## [171,] 2713 -0.0036 0.0048
## [172,] 899  0.0032  -0.0042
## [173,] 1859 -0.0011 0.0015
## [174,] 3764 -0.0011 0.0015
## [175,] 104  -0.0011 0.0015
## [176,] 940  -0.0011 0.0015
## [177,] 967  -0.0011 0.0015
## [178,] 1343 -0.0011 0.0015
## [179,] 1587 -0.0011 0.0015
## [180,] 1861 -0.0011 0.0015
## [181,] 1965 -0.0011 0.0015
## [182,] 2574 -0.0011 0.0015
## [183,] 2623 -0.0011 0.0015
## [184,] 2754 -0.0011 0.0015
## [185,] 2757 -0.0011 0.0015
## [186,] 2839 -0.0011 0.0015
## [187,] 2890 -0.0011 0.0015
```

```
## [188,] 3169 -0.0011 0.0015
## [189,] 3224 -0.0011 0.0015
## [190,] 3231 -0.0011 0.0015
## [191,] 3289 -0.0011 0.0015
## [192,] 3802 -0.0011 0.0015
## [193,] 3943 -0.0011 0.0015
## [194,] 4490 -0.0011 0.0015
## [195,] 4499 -0.0011 0.0015
## [196,] 84   -0.0011 0.0015
## [197,] 196  -0.0011 0.0015
## [198,] 455  -0.0011 0.0015
## [199,] 837  -0.0011 0.0015
## [200,] 856  -0.0011 0.0015
## [201,] 857  -0.0011 0.0015
## [202,] 920  -0.0011 0.0015
## [203,] 1062 -0.0011 0.0015
## [204,] 1214 -0.0011 0.0015
## [205,] 1291 -0.0011 0.0015
## [206,] 1292 -0.0011 0.0015
## [207,] 1490 -0.0011 0.0015
## [208,] 1560 -0.0011 0.0015
## [209,] 1721 -0.0011 0.0015
## [210,] 1745 -0.0011 0.0015
## [211,] 1818 -0.0011 0.0015
## [212,] 1833 -0.0011 0.0015
## [213,] 2197 -0.0011 0.0015
## [214,] 2359 -0.0011 0.0015
## [215,] 2598 -0.0011 0.0015
## [216,] 2746 -0.0011 0.0015
## [217,] 2888 -0.0011 0.0015
## [218,] 3259 -0.0011 0.0015
## [219,] 3361 -0.0011 0.0015
## [220,] 3570 -0.0011 0.0015
## [221,] 3703 -0.0011 0.0015
## [222,] 3948 -0.0011 0.0015
## [223,] 3966 -0.0011 0.0015
## [224,] 4202 -0.0011 0.0015
## [225,] 4212 -0.0011 0.0015
## [226,] 4236 -0.0011 0.0015
## [227,] 4363 -0.0011 0.0015
## [228,] 4435 -0.0011 0.0015
## [229,] 4526 -0.0011 0.0015
## [230,] 4606 -0.0011 0.0015
## [231,] 4664 -0.0011 0.0015
```

Using the function pamr.listgenes we can get the features selected by the method, as seen above, and get the 10 most contributing, which are:

```
## papers
## important
## submission
## due
## published
## position
## call
```

```
## conference
## dates
## candidates
```

The words above are clearly used during Conferences emails, so it is normal that they have a strong effect on the discrimination between the conference mails and the other.

Finally, the test error is:

```
##       pred
## y.test  0  1
##      0 10  0
##      1  2  8
```

missclassification= 0.1

2)

In this part we compute the test error and number of contributing features for the elastic net and Support vector machine.

Elastic net

First we will use the elastic net with the binomial response and $\alpha = 0.5$ in which penalty is selected by cross-validation.

Once we do the cross-validation, with a penalty of $\lambda = 0.1311628$, we can see that the number of features different than 0, which mean that they are selected are 39.

```
##     y.test
## pred  0  1
##    0 10  2
##    1  0  8
```

The missclassification is of 0.1, which is the same error as performing shrunken centroids.

Support vector machine

```
##  Setting default kernel parameters
```

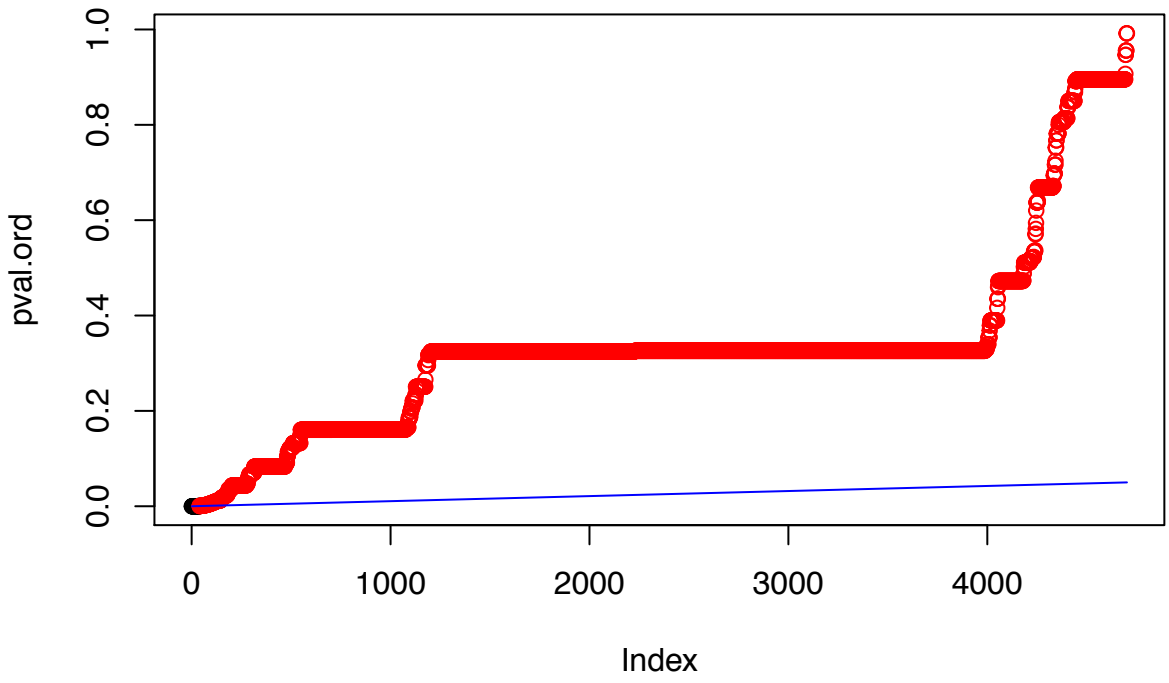
```
##        y.test
## predsvm  0  1
##       0 10  1
##       1  0  9
```

When using SVM with vanilladot kernel, the number of features is 43 and a missclassiffication error of 0.05 which makes it better than shrunken centroids and elastic net.

3)

Benjamini-Hochberg

In this part we are asked to implement the Benjamini-Hochberg method for the original data, and use t.test() for computing p-values. We selected a slope of 0.05 to reject the hypothesis.



As we can see in the plot, the amount of features corresponding to the rejected hypothesis are:

```
##         apply       authors          call        camera     candidate    candidates        chairs
##           272           389           596           599           606           607           681
##    submission          team        topics      workshop
##          4060          4177          4282          4628
```

As we can see the features above, having the p-value lower than the slope means that, the hypothesis of them being relevant is rejected.

Appendix

```
knitr::opts_chunk$set(echo = TRUE)
library(readxl)
library(ggplot2)
library(mgcv)
library("gamreg")

data <- read_excel("influenza.xlsx")

ggplot() + geom_line(aes(x=data$Time,y= data$Mortality),color="RED") + geom_line(aes(x=data$Time,y=data$
 xlab("Time") + ylab(NULL)


classifier <- gam(Mortality ~ Year + s(Week,k=50),data = data )
classifier$coefficients

ggplot() + geom_line(aes(x=data$Time,y= data$Mortality),color="RED") + geom_point(aes(x=data$Time,y=clas
  xlab("Time") + ylab("Mortality")+
  ggtitle(label = "FITTED VS OBSERVED VALUES")



summary(classifier)
plot(classifier)

classifier1 <- gam(Mortality ~ Year + s(Week, sp = 1000),data = data ) #High
classifier2 <- gam(Mortality ~ Year + s(Week, sp = 0.1),data = data ) #Low

ggplot() + geom_line(aes(x=data$Time,y= data$Mortality),color="RED") + geom_line(aes(x=data$Time,y=class
  geom_point(aes(x=data$Time,y=classifier2$fitted.values),color="BLACK")+
  xlab("Time") + ylab("Mortality")+ggtitle("Fitted values of high and low penalty factors with observed


ggplot() + geom_line(aes(x=data$Time,y= data$Influenza),color="RED") + geom_line(aes(x=data$Time,y=class
  xlab("Time") + ylab("Influenza") +
ggtitle("Residuals vs Influenza values")

classifier3 <- gam(Mortality ~ Year + s(Week,k=20)+s(Year,k=8)+s(Influenza,k=10),data = data)

plot(classifier3)

ggplot() + geom_line(aes(x=data$Time,y= data$Mortality),color="RED") + geom_point(aes(x=data$Time,y=clas
  geom_point(aes(x=data$Time,y=classifier3$fitted.values),color = "BLUE")+
  xlab("Time") + ylab("Mortality")

library(pamr)
email<-read.csv("data.csv", sep=";", fileEncoding = "latin1")

email$Conference<-as.factor(email$Conference)
n=dim(email)[1]
set.seed(12345)
id=sample(1:n, floor(n*.7))
```

```
train=email[id,]
test=email[-id,]

rownames(email)=1:nrow(email)

x=t(train[,-4703])
y=train[[4703]]

x.test=t(test[,-4703])
y.test=test[[4703]]
mydata=list(x=x,y=as.factor(y),geneid=as.character(1:nrow(x)), genenames=rownames(x))
mydatatest = list(x=x.test, y=as.factor(y.test), geneid=as.character(1:nrow(x.test)), genenames=rowname

model=pamr.train(mydata)

cvmodel=pamr.cv(model,mydata)
#print(cvmodel)
pamr.plotcv(cvmodel)


pamr.plotcen(model, mydata, threshold=1.3)
a=pamr.listgenes(model,mydata,threshold=1.3)
#cat( paste( colnames(data)[as.numeric(a[,1])], collapse='\n'))
ten<-names(email)[as.numeric(a[1:10])]

for(i in 1:10){
  cat(ten[i], "\n")
}

pred <- pamr.predict(model, mydatatest$x, threshold=1.3)

t<-table(y.test,pred)
t
misscn <- t[1,2]+t[2,1]/sum(t)


library(glmnet)
library(kernlab)
set.seed(12345)

cv.gl <- cv.glmnet(t(x), y , family="binomial", alpha=.5)
lambmin <- cv.gl$lambda.min
fit.elnet <- glmnet(t(x), y, family="binomial",lambda=lambmin, alpha=.5)
feat <- length(which(coef(fit.elnet)!=0))
pred <- predict(cv.gl, s=cv.gl$lambda.min, newx=t(x.test), type="class")
t<-table(pred, y.test)
t
missc <- (t[1,2]+t[2,1])/sum(t)

vars <- length(which(coef(fit.elnet)!=0))
svm <- ksvm(t(x),y, kernel="vanilladot", type="C-svc", scale=F)
predsvm <- predict(svm, t(x.test))
ts <- table(predsvm, y.test)
```

```
ts
missvm <- (ts[1,2]+ts[2,1])/sum(ts)
varsvm <- length(coef(svm)[[1]])
pval=c()
pval <-apply(email[-4703],2, FUN=function(x){
  as.numeric(t.test(x~email$Conference)$p.value)})

j <- 1:length(pval)
M <- length(pval)
slope <- 0.05*j/M

pval.ord <- sort(as.numeric(pval))
for(i in 1:length(pval)){
  if(pval.ord[i] >= slope[i]){break}

}
col=c()
slopeval <- as.numeric(pval[order(pval)[i]])
col[pval.ord<slopeval] <- 1
col[pval.ord>=slopeval] <- 2
plot(pval.ord, col=col)
lines(slope, col="blue")

which(pval<slopeval)
```

TDDE01: LAB 2

Group lab report

Hampus Carlsson (hamca886@student.liu.se)
Christoffer Fors Johansson (chrjo522@student.liu.se)
Daniel Gellerman (dange008@student.liu.se)

Contribution

The contribution to this document looks as follows. After discussing different approaches Daniel, Hampus and Christoffer contributed with assignment 2, 3 and 4, respectively.

Assignment 2: Analysis of credit scoring

2.

	Deviance	**Gini index**
Misclassification rate train	0.212	0.24
Misclassification rate test	0.248	0.344

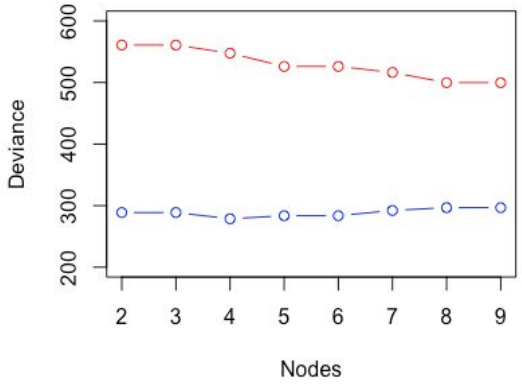
Table 2.1

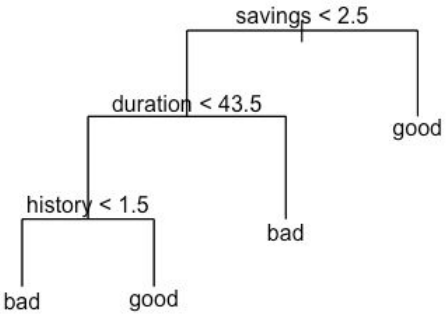
Deviance gives lower misclassification rates, which makes it the choice for the following steps.

3.

To the left is the plot for the deviance depending on the number of leaves. The red line is for the training data and the blue is for the validation data. Notice here that the model appears to perform better on the validation data than on the training data. The tree with lowest deviance has four nodes and is presented beneath. The depth of the tree is 3 and the variables it uses are savings, duration and history. The misclassification rate for the test data is 0.251.

From the tree it is easy to interpret that the most essential variable is savings, since if the client has a savings value above 2.5 it is immediately considered as a good client. The second most important variable is duration and on third place is history.





4.

	Confusion Matrix	Misclassification Rate

Naive Bayes on training data.	Prediction
	`> missclassNBtrain`
	`bad good`
	ref `bad 95 52`
	`good 98 255`
		0.3

Naive Bayes on test data.	Prediction
	`> missclassNBtest`
	`bad good`
	ref `bad 50 25`
	`good 61 114`
		0.344

Table 2.2

The misclassification rate for the test data is higher with the Naïve Bayes method, which makes the tree()-function a better choice for this data.

5.

	Confusion Matrix	Misclassification Rate

Naive Bayes on test data. WITH LOSS MATRIX	Prediction
	`bad good`
	ref `bad 66 9`
	`good 130 45`
		0.546

Naive Bayes on train data. WITH LOSS MATRIX	Prediction
	`bad good`
	ref `bad 137 10`
	`good 263 90`
		0.556

Table 2.3

Using the loss matrix $L = \begin{bmatrix} 0 & 1 \\ 10 & 0 \end{bmatrix}$, which assigns a loss of 10 to predicted values of "good" which in fact are "bad" we get a confusing matrix that shows a lot fewer false positive predictions. For the training data we see a reduction from 25 (Table 2.2) to 10 (Table 2.3).

Assignment 3: Uncertainty estimation

1) Looking at the input data (see left column of Table 3.1) we see no apparent linear relationship in the data. There might be some visible regions where we have a region of higher values for low values of MET and then another region of slightly lower values in the middle part of the plot; this is then followed by another region of somewhat higher values of EX as MET increases. If these could in fact be considered as regions maybe a tree model could be appropriate.
2)

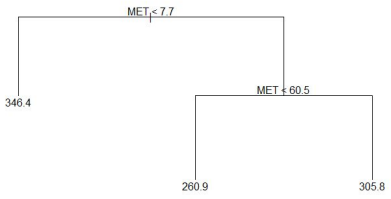


Figure 3.1 - Tree model for estimating MET based on EX

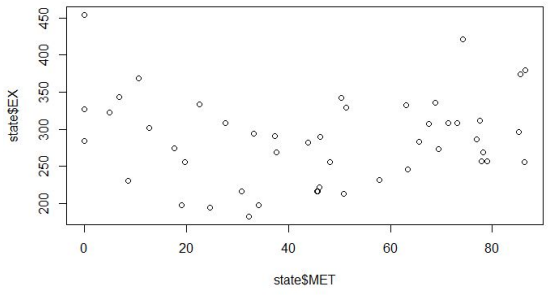
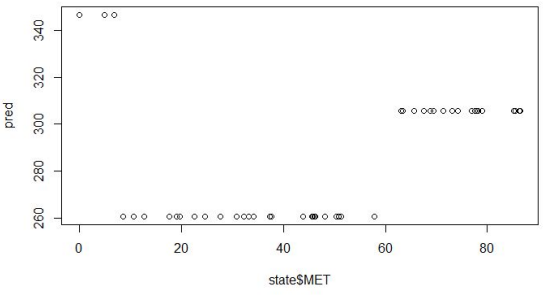
 The original data, EX is plotted against MET i ascending order.	 The data classified using the tree model presented above.

Table 3.1

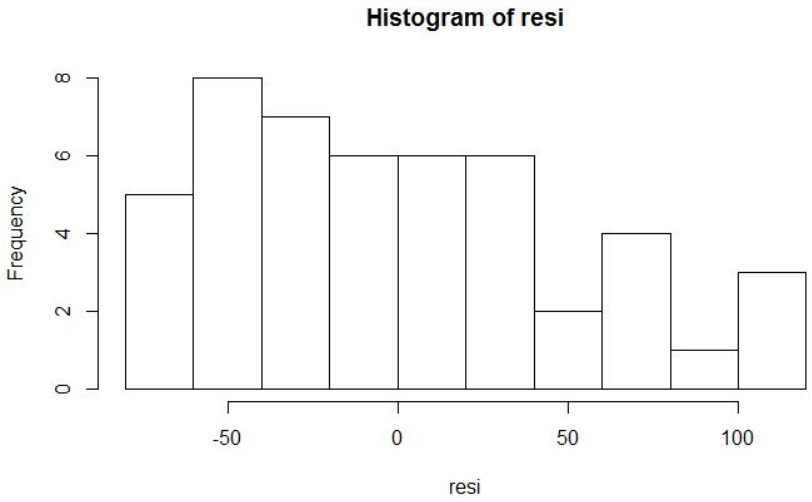


Figure 3.2 - Residuals for the tree model

Looking at the histogram of residuals is Figure 3.2 we see nothing that reminds us of any distribution we know. From Table 3.1 right column it is seen that the values predicted by the model follow a much simpler pattern than the original relationship between EX and MET. The model behaves like a piecewise function and has identified 3 regions.

3.3)

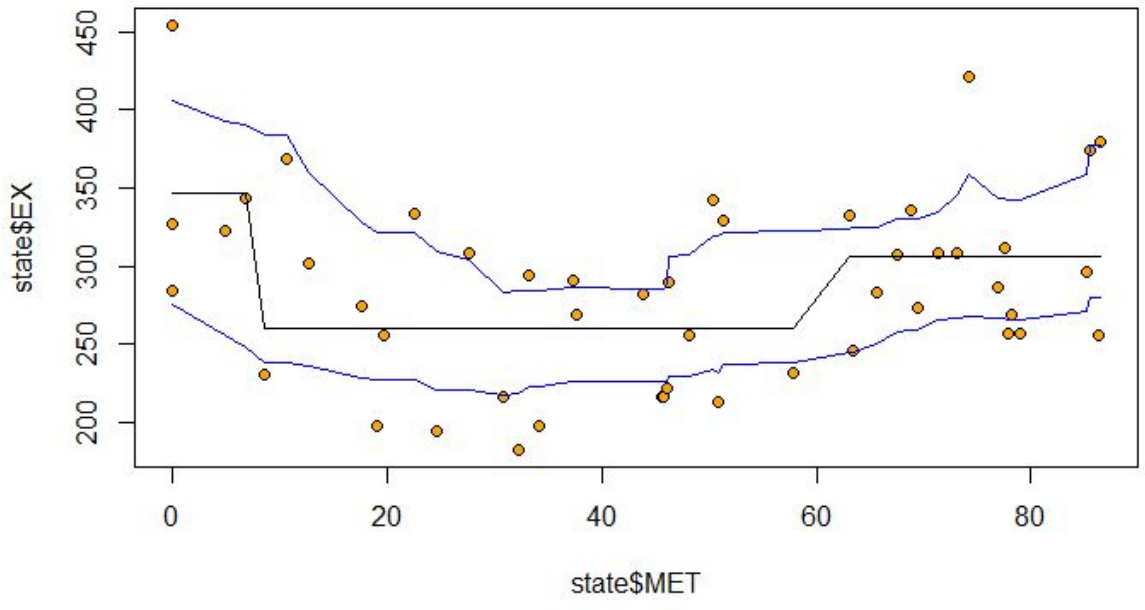


Figure 3.3 - Fitted tree model with confidence bands using non-parametric bootstrap

Addressing the bumps: The model we are using is not a smooth function and therefore the confidence interval is not smooth either.

It appears that the confidence band is wider in the regions of less data i.e the confidence band is widest in the beginning where the data is spread and few and then the band gets tighter where the data is dense and less spread towards the higher values of MET.

The model does not seem to reliable as there is no region where the confidence bands are really close to the predicted model. However the model seems to capture the "gist" of what is going on in the data.

3.4)

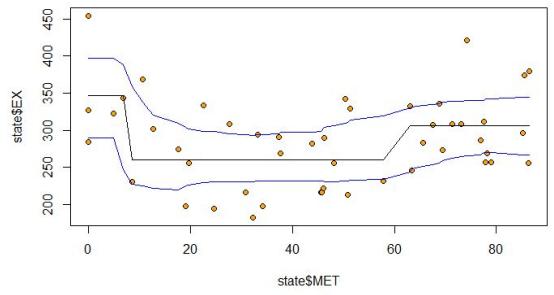
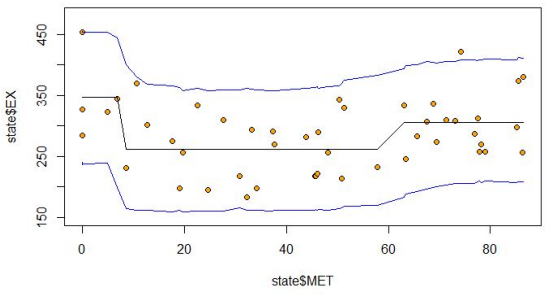
	
Confidence bands for parametric bootstrap	Prediction bands for parametric bootstrap

Table 3.2

The confidence band for parametric bootstrap appears almost identical to the one of the non-parametric bootstrap. Thus, our belief of whether the model is good or not has not changed since we conducted the non-parametric bootstrap.

Less than 5 % of the data is outside the prediction bands (1/48 ~ 2%). From this we conclude that the prediction bands are reliable.

3.5) Since the histogram of residuals above indicated no particular distribution the nonparametric bootstrap should be the more appropriate one as the parametric one is generally only better when the distribution of residuals is known.

Assignment 4. Principal components

1) Figure 4.1 illustrate how much variation is explained by each feature. Also, by looking at the data we can see that our first 2 components explains more than 99% of the total variance:

[1] "93.332" "6.263"

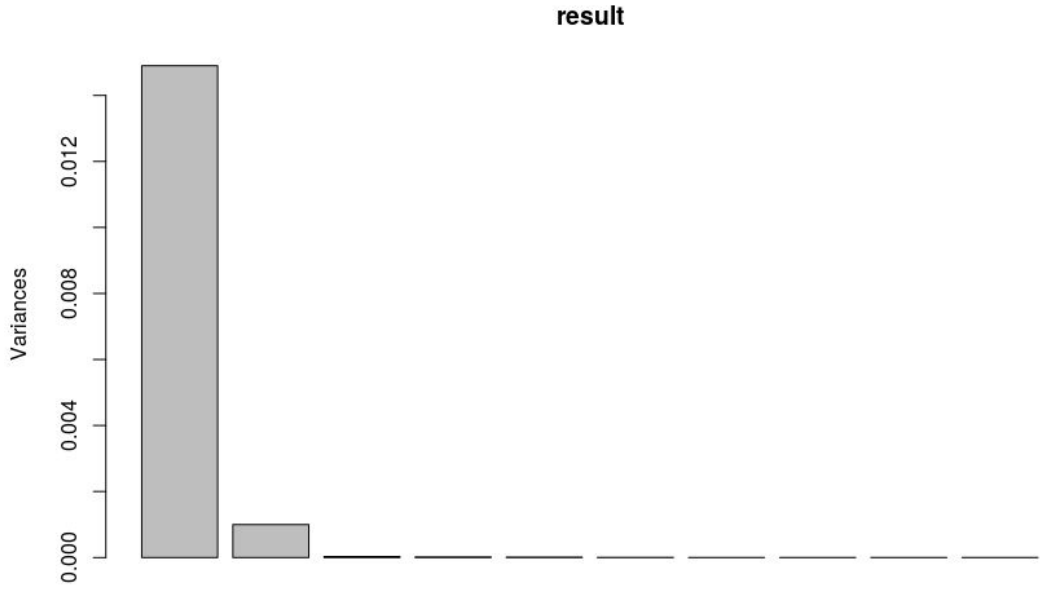


Figure 4.1

Figure 4.2 below shows the scores in the coordinates (PC1, PC2). We can see that there are a few diesel fuels with different score than the others at the right hand side of the plot, which could be considered as quite unusual.

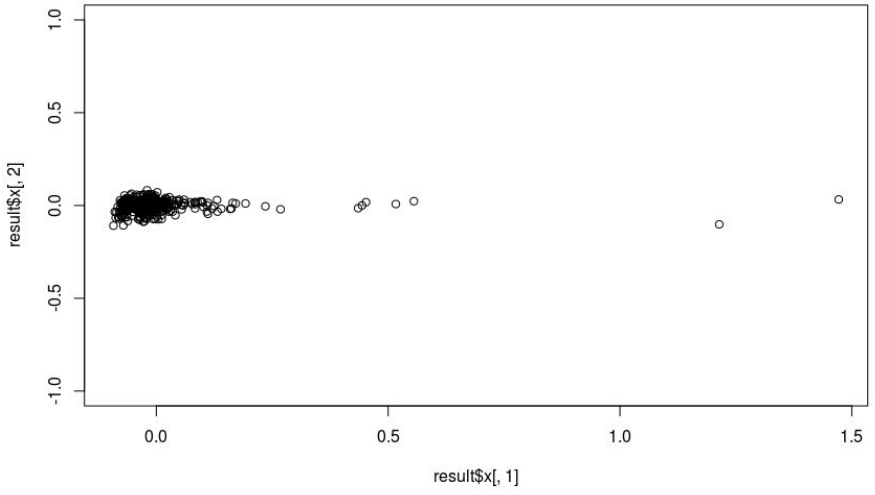


Figure 4.2

2) Figure 4.3 and Figure 4.3 shows the trace plots of the loading of the components for PC1 and PC2, respectively. We can see that the trace plot for PC1 is somehow moving while the trace plot for PC2 remains close to 0 for the first ~100 positions in the vector, indicating that the last 20 features is "the few original ones".

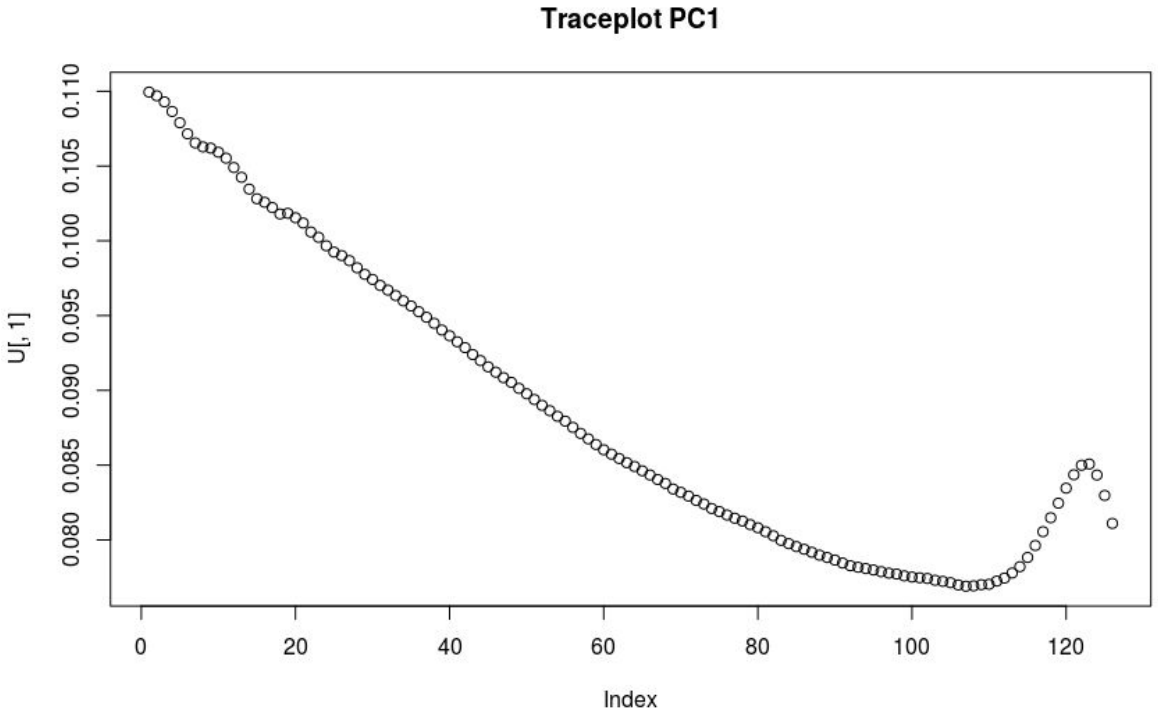


Figure 4.3

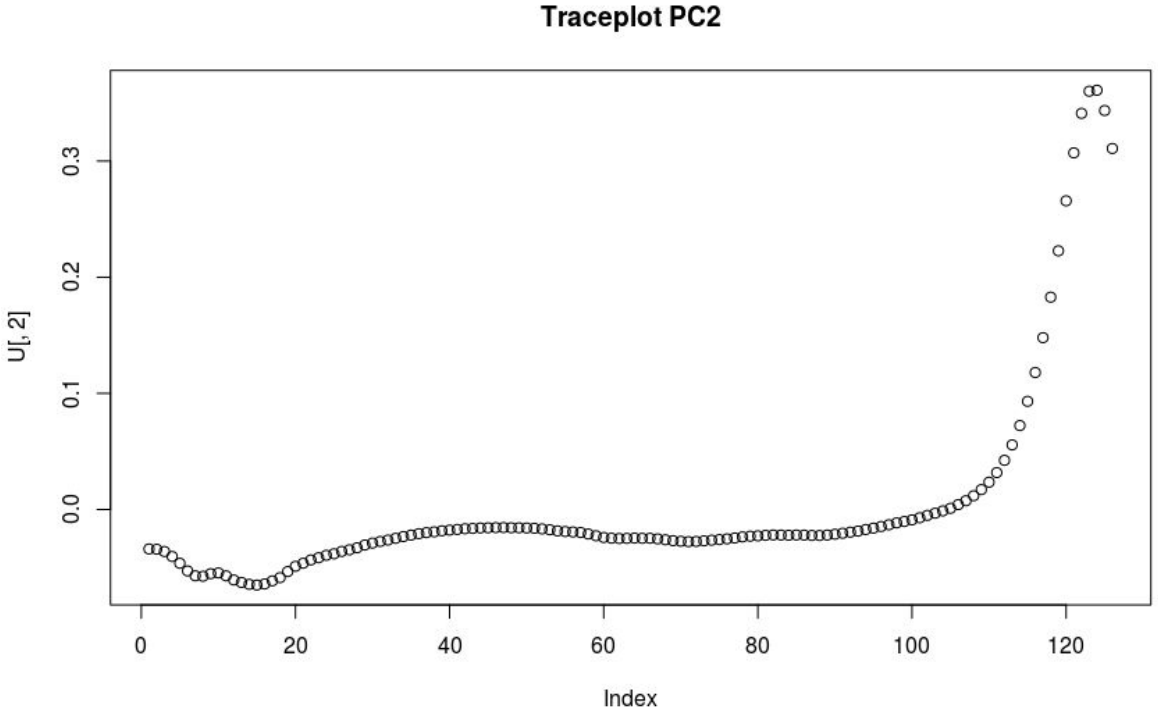


Figure 4.4

3) **a)** Figure 4.5 and Figure 4.6 shows the trace plots PC1 and PC2 using W' matrix. The columns of W' appear to have the same non-zero components as PC1 and PC2 from step 2. W' contain the loadings.

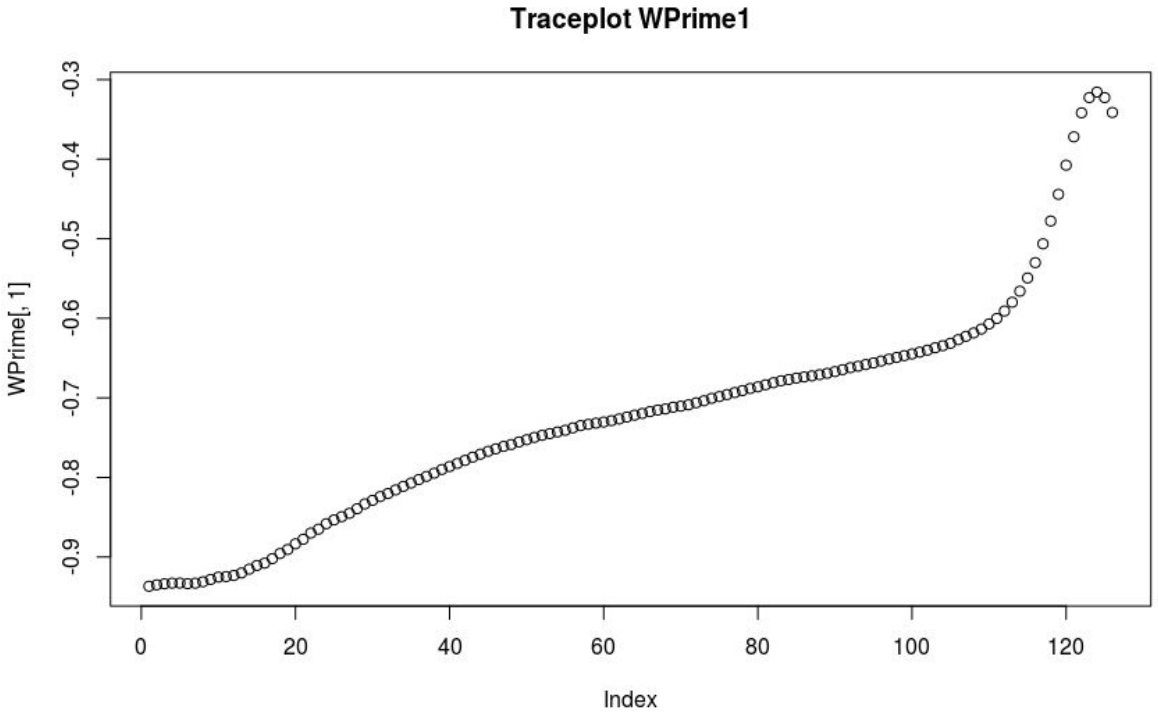


Figure 4.5

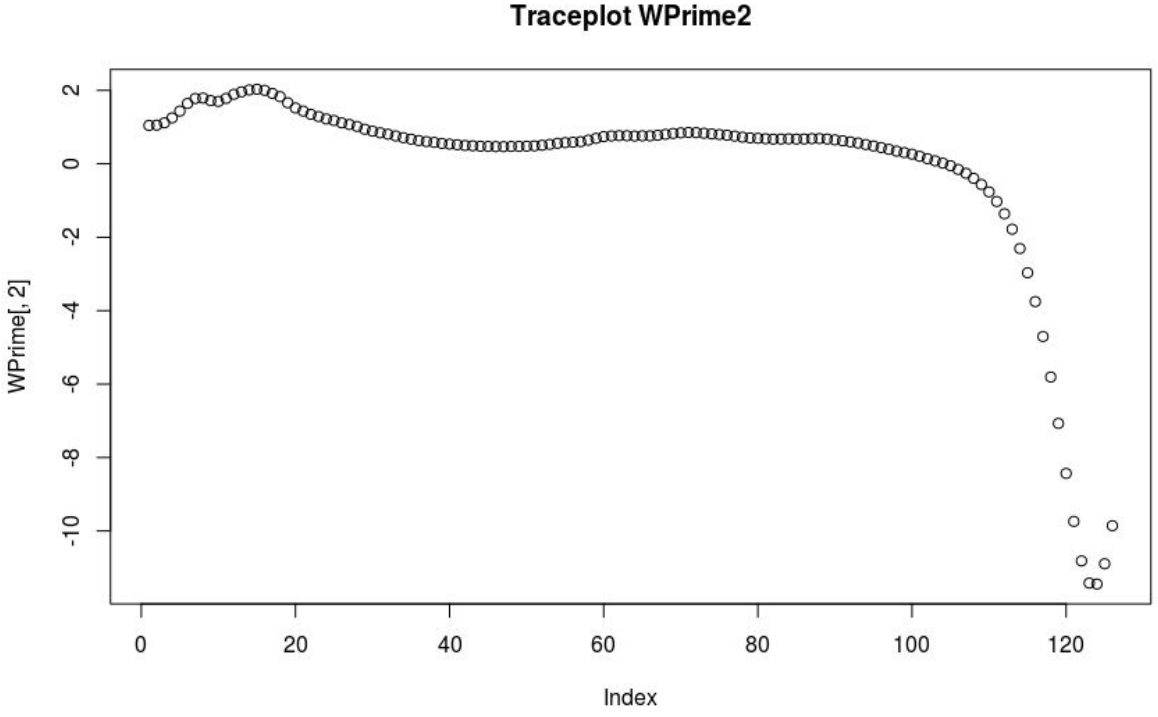


Figure 4.6

b) Figure 4.7 below illustrates the the scores of the first two latent features. We can see the similarities with the scores from step 1 if we look at the number of outliers in the figure, and also the general clustering of the data.

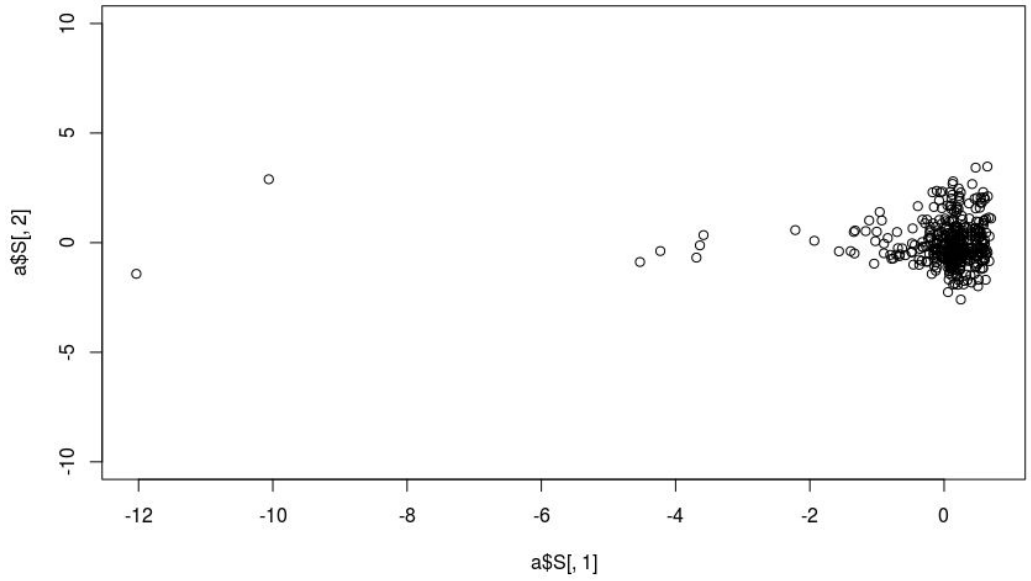


Figure 4.7

4) Figure 4.8 below illustrates the plot showing the dependence of the mean-squared predicted error on the number of components in the model. We can see that it is reasonable to select at least around 10 number of components since almost all the gain is around there.

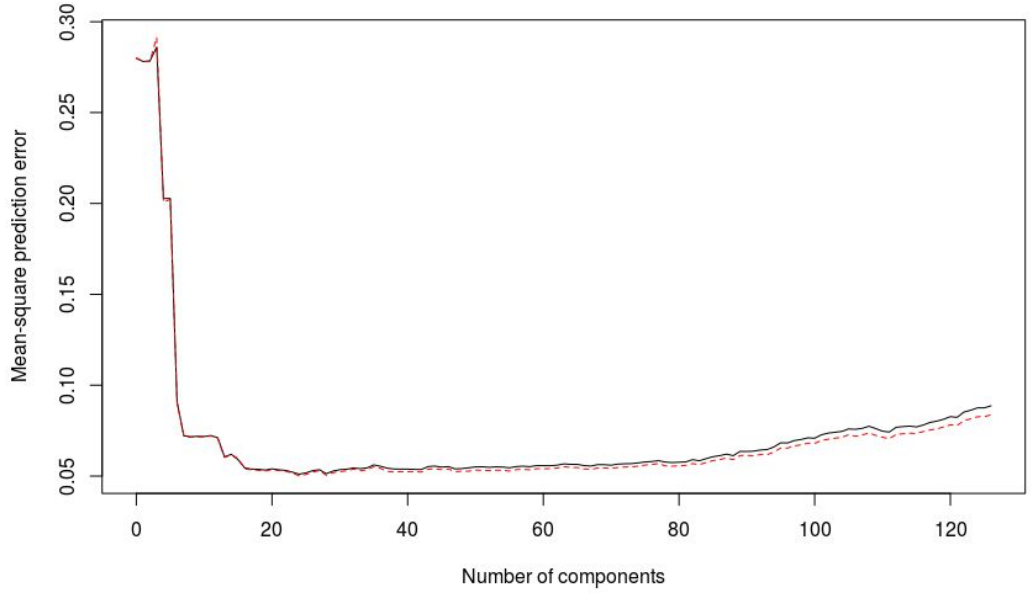


Figure 4.8

Appendix A: Code

Assignment 2:

```
#1
import_data=function(){
  data = read.csv('creditscoring.csv', sep = ";", dec = ",")
  data$good_bad = as.factor(data$good_bad) #, levels = c('bad', 'good'), labels = c(0,1))
  n=dim(data)[1]
  set.seed(12345)
  id=sample(1:1000, floor(n*0.5))
  train<<-data[id,]
  test_val = data[-id,]
  id2 = sample(1:(n/2), floor(n/2*0.5))
  test <<- test_val[id2,]
  validation <<- test_val[-id2,]
}

import_data()

#2
func1 = function(train, test){
  n = dim(train)[1]
  m = dim(test)[1]
fit.dev <<- tree(good_bad~., data = train, split = 'deviance')
fit.gini = tree(good_bad~., data = train, split = 'gini')

predict.dev = predict(fit.dev, newdata = test, type = 'class')
predict.gini = predict(fit.gini, newdata = test, type = 'class')
predict.dev.train = predict(fit.dev, newdata = train, type = 'class')
predict.gini.train = predict(fit.gini, newdata = train, type = 'class')

missclass.gini.test <<- 1-sum(diag(table(test$good_bad, predict.gini)))/m
missclass.dev.test <<- 1-sum(diag(table(test$good_bad, predict.dev)))/m
missclass.gini.train <<- 1-sum(diag(table(train$good_bad, predict.gini.train)))/n
missclass.dev.train <<- 1-sum(diag(table(train$good_bad, predict.dev.train)))/n
}
func1(train, test)



#3
trainScore = rep(0,9)
testScore = rep(0,9)

for (i in 2:9) {

  prunedTree = prune.tree(fit.dev, best = i)
  pred = predict(prunedTree, newdata = validation, type = 'tree')
  trainScore[i] = deviance(prunedTree)
  testScore[i] = deviance(pred)

}
```

```
plot(2:9, trainScore[2:9], type = 'b', col = 'red', ylab = 'Deviance',xlab = 'Nodes' ,ylim
= c(200,600))
points(2:9, testScore[2:9], type = 'b', col = 'blue')

#Kollar missclass för bästa trädet
finalTree=prune.tree(fit.dev, best = 4)
Yfit = predict(finalTree, newdata=test, type = 'class')
missclassTest = 1-sum(diag(table(test$good_bad, Yfit)))/dim(test)[1]
summary(finalTree)
plot(finalTree)
text(finalTree)

#4
testLength = dim(test)[1]
trainLength = dim(train)[1]

fit = naiveBayes(train$good_bad~.,data = train)
missclassNBtest = table(test$good_bad,predict(fit, test))
missclassNBtrain = table(train$good_bad,predict(fit, train))
missclassrateNBtest = 1-sum(diag(missclassNBtest))/testLength
missclassrateNBtrain = 1-sum(diag(missclassNBtrain))/trainLength

#5
predict2 = predict(fit, train, type = 'raw')
predictBad = predict2[,'bad']
predictGood = predict2[,'good']
loss = matrix(c(0,1,10,0), byrow = TRUE, nrow = 2)

predictedTrain = predictGood/predictBad

predictedTrain = ifelse(predictedTrain>loss[2,1],'good','bad')
table(train$good_bad, predictedTrain)

predict3 = predict(fit, test, type = 'raw')
predictBad2 = predict3[,'bad']
predictGood2 = predict3[,'good']
loss = matrix(c(0,1,10,0), byrow = TRUE, nrow = 2)

predictedTest = predictGood2/predictBad2

predictedTest=ifelse(predictedTest>loss[2,1],'good','bad')
table(test$good_bad, predictedTest)
```

Assignment 3

```
state=read.csv2("State.csv")

# Assignment 3.1
state = state[order(state["MET"]),]
plot(state$MET, state$EX)

# Assignment 3.2
library(tree)

model = tree(EX~MET, data = state, control = tree.control(dim(state)[1], minsize = 8))
set.seed(12345)
cv.res=cv.tree(model)
plot(cv.res$size, cv.res$dev, type="b",col="red")
selectedTree = prune.tree(model, best = 3)

pred = predict(selectedTree, newdata = state)

plot(state$MET, pred)

resi = residuals(selectedTree)

hist(resi)

# Assignmment 3.3

library(boot)

f=function(data, ind){

  data1=data[ind,]# extract bootstrap sample
  res=tree(EX~MET, data = data1, control = tree.control(dim(data1)[1], minsize = 8))
  res = prune.tree(res, best = 3)

  ex=predict(res,newdata=state)
  return(ex)
}

res=boot(state, f, R=1000)

# Defaults to 95 %
e=envelope(res) #compute confidence bands


plot(state$MET, state$EX, pch=21, bg="orange")
points(state$MET,pred,type="l") #plot fitted line
#plot cofidence bands
points(state$MET,e$point[2,], type="l", col="blue")
points(state$MET,e$point[1,], type="l", col="blue")


# The model is not a smooth function, and therefore our confidance intervals are not smooth
either.
```

```r
# Assignment 3.4

mle = selectedTree
rng=function(data, mle) {

  data1=data.frame(EX=data$EX,MET=data$MET)

  n=length(data$EX)

  #generate new Price
  data1$EX=rnorm(n,predict(mle,newdata=data1),sd(residuals(mle)))

  return(data1)
}
f1=function(data1){

  res=tree(EX~MET, data = data1, control = tree.control(dim(data1)[1], minsize = 8))
  res = prune.tree(res, best = 3)

  ex=predict(res,newdata=state)
  return(ex)
}
res_param=boot(state, statistic=f1, R=1000, mle=mle,ran.gen=rng, sim="parametric")

e_param=envelope(res_param) #compute confidence bands


plot(state$MET, state$EX, pch=21, bg="orange")
points(state$MET,pred,type="l") #plot fitted line
#plot cofidence bands
points(state$MET,e_param$point[2,], type="l", col="blue")
points(state$MET,e_param$point[1,], type="l", col="blue")

f2=function(data1){

  res=tree(EX~MET, data = data1, control = tree.control(dim(data1)[1], minsize = 8))
  res = prune.tree(res, best = 3)


  pred=predict(res,newdata=state)
  n=length(state$EX)
  predicted_ex=rnorm(n,pred, sd(residuals(res)) )
  return(predicted_ex)
}

res_pred=boot(state, statistic=f2, R=10000, mle=mle,ran.gen=rng, sim="parametric")

e_param=envelope(res_pred) #compute confidence bands


plot(state$MET, state$EX, pch=21, bg="orange", ylim = c(150, 480))
points(state$MET,pred,type="l") #plot fitted line
#plot cofidence bands
```

```
points(state$MET,e_param$point[2,], type="l", col="blue")
points(state$MET,e_param$point[1,], type="l", col="blue")
```

Assignment 4:

```
# Read the data
NIRSpectra = read.csv2("Documents/Machine Learning/lab2/NIRSpectra.csv")

# -- Step 1 ---
# Make a copy of the original data
data = NIRSpectra
# Remove the Viscosity
data$Viscosity = c()
result = prcomp(data)

lambda = result$sdev^2
# Eigenvalues
lambda
# Proportion of variation
sprintf("%2.3f", lambda / sum(lambda) * 100)
screeplot(result)

# Scores in the coordinates (PC1, PC2)
plot(result$x[,1], result$x[,2], ylim=c(-1, 1))

# -- Step 2 ---
# Principal component loadings
U = result$rotation
plot(U[,1], main="Traceplot PC1")
plot(U[,2], main="Traceplot PC2")

# --- Step 3 ---
library(fastICA)
set.seed(12345)
# 3.a
a <- fastICA(data, 2, alg.typ = "parallel", fun = "logcosh", alpha = 1,
             method = "R", row.norm = FALSE, maxit = 200, tol = 0.0001, verbose = TRUE)

WPrime = a$K %*% a$W
# Trace plots
plot(WPrime[,1], main="Traceplot WPrime1")
plot(WPrime[,2], main="Traceplot WPrime2")

# Ask about this!
plot(a$S[,1], a$S[,2], ylim=c(-10, 10))

# --- Step 4 ---
library(pls)
set.seed(12345)

pcr_model = pcr(NIRSpectra$Viscosity~., data = data, validation = "CV")
validationplot(pcr_model, val.type = "MSEP", xlab = "Number of components", ylab
="Mean-square prediction error", main = "")
```