Lab 2 TDDE01

Filip Cornell

2018-12-04

1 Assignment 1

First, we initialize functions necessary for the task.

```r
classificationRate = function(confMatrix) {
  return(sum(diag(confMatrix))/sum(confMatrix))
}

log_reg = function(data, fitModel, pLimit = 0.5) {
  # Specify response in predict,
  fits = predict(fitModel, data)

  probabilities = fits

  classifications = apply(as.matrix(probabilities), 1, function(row) {
    if (row > pLimit) {
      return(1)
    } else {
      return(0)
    }
  })
  return(classifications)
}
```

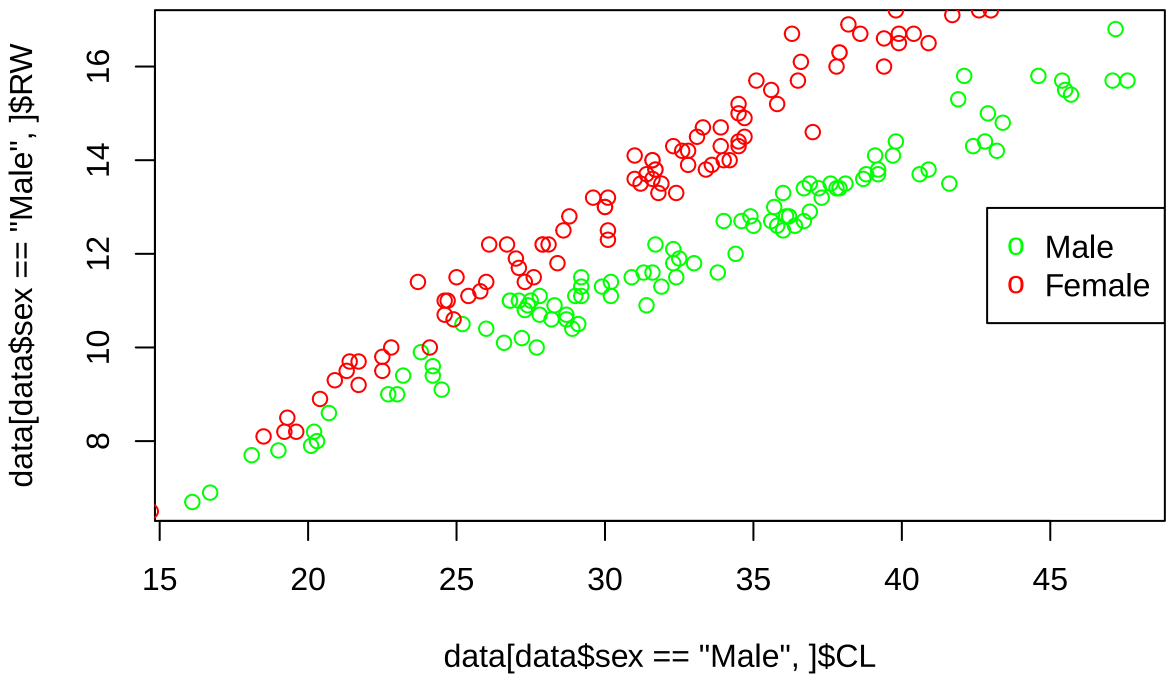
1.1 Task 1

Plotting the sex of the crabs on the axis of `RW` and `CL` shows a clear possibility of separating the two sexes through linear classification.

```r
# Task 1

data = read.csv("australian-crabs.csv")

# Should I use training and test? Probably yes
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data[id,]
test=data[-id,]
plot(data[data$sex == "Male",]$CL,
     data[data$sex == "Male",]$RW, col = "green")
points(data[data$sex == "Female",]$CL,
       data[data$sex == "Female",]$RW, col = "red")
legend("right",
       pch = c("o","o"),
       col = c("green", "red"),
       legend = c("Male","Female"))
```



1.2 Task 2

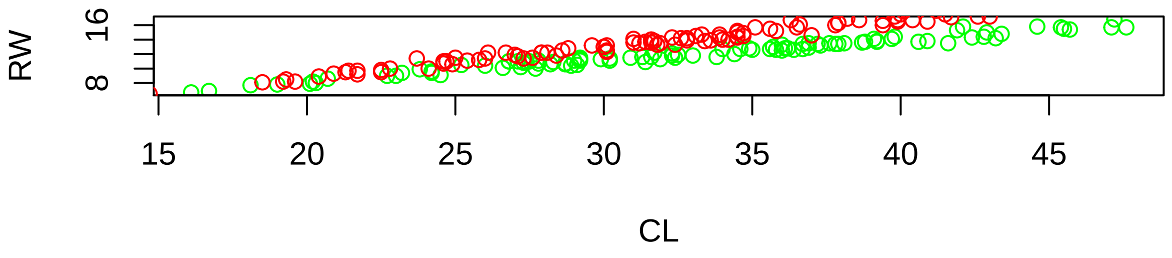
Performing a Linear Discriminant Analysis on the data through the function `lda()`, using the variables RW and CL yields an almost perfect separation of the data, which confirms what the graph indicated. The misclassification rate is only 0.035, an almost perfect score.

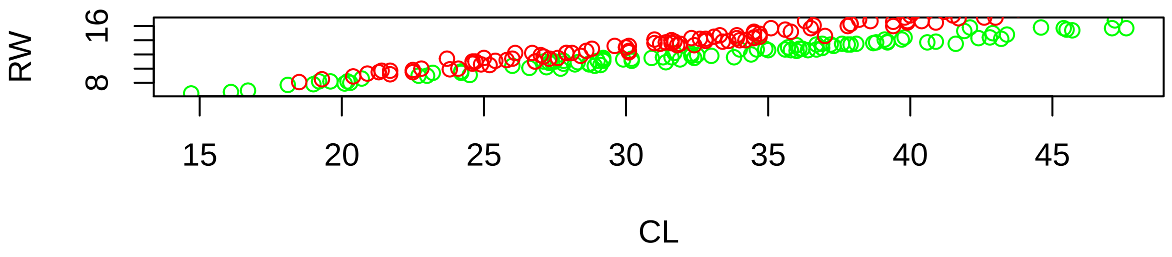
```
# Task 2

library(MASS)

fitModel = lda(formula = sex ~ RW + CL,
               data = data,
               prior = c(length(data$sex[data$sex == "Male"])
                         /nrow(data),length(data$sex[data$sex == "Female"])
                         /nrow(data)))

fits = predict(fitModel, data)
par(mfrow=c(2,1))
plot(data[data$sex == "Male",]$CL,
     data[data$sex == "Male",]$RW,
     col = "green", ylab = "RW", xlab = "CL")
points(data[data$sex == "Female",]$CL,
       data[data$sex == "Female",]$RW, col = "red")
plot(data[fits$class == "Male",]$CL,
     data[fits$class == "Male",]$RW,
     col = "green", ylab = "RW", xlab = "CL")
points(data[fits$class == "Female",]$CL,
       data[fits$class == "Female",]$RW,
       col = "red")
```





```
par(mfrow=c(1,1))

confMatrix = table(data$sex, fits$class)
confMatrix
```

```
##
##          Female Male
##   Female     97    3
##   Male        4   96
```

```
print("misclassificionation rate")
```

```
## [1] "misclassificionation rate"
```

```
1 - classificationRate(confMatrix)
```

```
## [1] 0.035
```

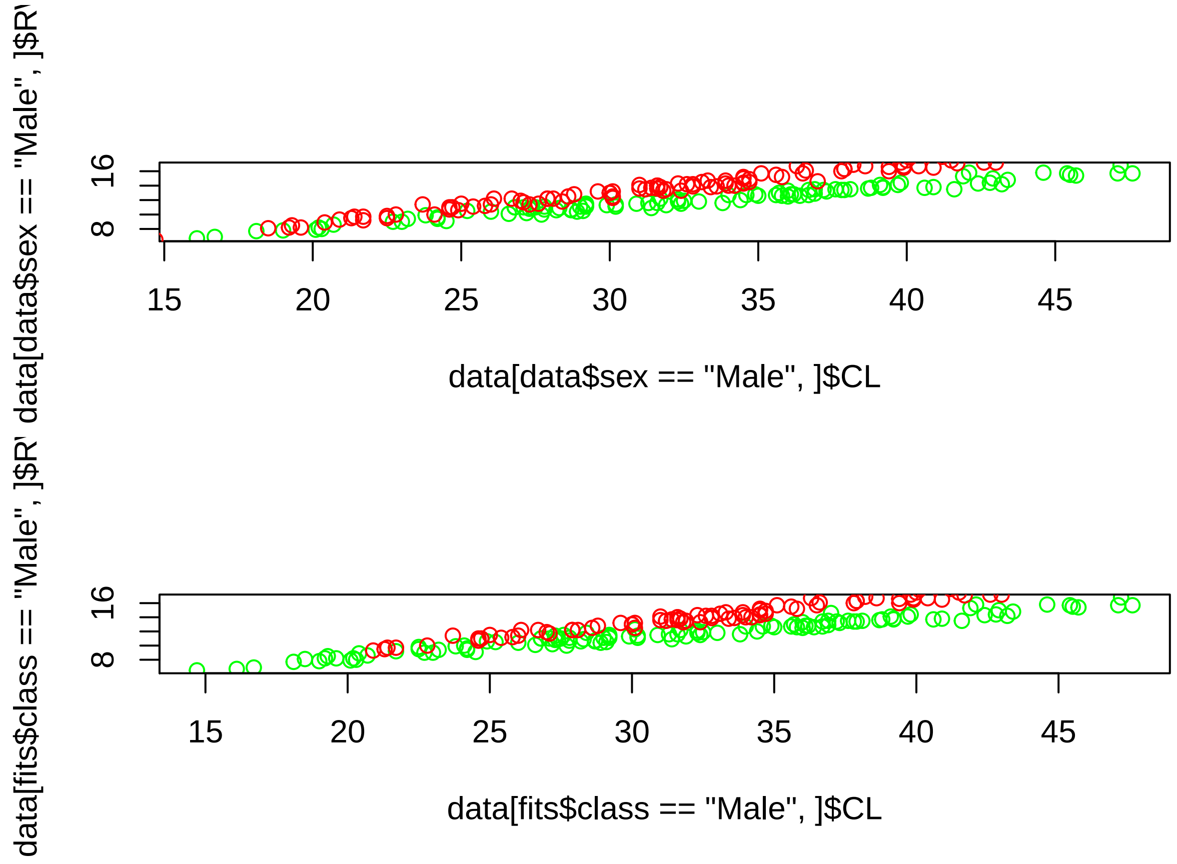
1.3 Task 3

In task 3, we changed the priors from $P(sex = Male) = 0.5$ and $P(sex = Female) = 0.5$ to $P(sex = Male) = 0.9$ and $P(sex = Female) = 0.1$. Although the result worsened to a misclassification rate to only 8 %, the linearly separable property of the data still yielded a high accuracy of 92 %. As can be seen in the confusion matrix, this skews some predictions of females to be predicted as males, as the priors skews the data towards this.

```
# Task 3

fitModel = lda(formula = sex ~ RW + CL, data = data, prior = c(0.1, 0.9))
fits = predict(fitModel, data)
par(mfrow=c(2,1))
plot(data[data$sex == "Male",]$CL,
```

```r
     data[data$sex == "Male",]$RW, col = "green")
points(data[data$sex == "Female",]$CL,
       data[data$sex == "Female",]$RW, col = "red")
plot(data[fits$class == "Male",]$CL,
     data[fits$class == "Male",]$RW, col = "green")
points(data[fits$class == "Female",]$CL,
       data[fits$class == "Female",]$RW, col = "red")
```



```r
par(mfrow=c(1,1))
confMatrix = table(data$sex, fits$class)
confMatrix
```

```
##
##           Female Male
##   Female      84   16
##   Male         0  100
```

```r
print("misclassificionation rate")
```

```
## [1] "misclassificionation rate"
```

```r
1 - classificationRate(confMatrix)
```

```
## [1] 0.08
```

1.4 Task 4

In task 4, we use logistic regression to fit the model. The results are here very good as well, only one less classified incorrectly compared to LDA. The plot shows the line separating the sexes, which appears separare the vast majority of the data points correctly.

Retrieving the separating line from the logistic is trivial. Consider the logistic regression:

$$p(Y=1|w,x) = \frac{1}{1+e^{w^Tx}} \text{ where } w^T = \beta_0 + \beta_{RW}x_{RW} + \beta_{CL}x_{CL}$$

Then, with a classification threshold of 0.5, we get that

$$0.5 = \frac{1}{1+e^{w^Tx}} \Leftrightarrow 0.5+0.5e^{w^Tx} = 1 \Leftrightarrow e^{w^Tx} = 1 \Leftrightarrow w^Tx = log(1) = 0 \Leftrightarrow w^Tx = \beta_0+\beta_{RW}x_{RW}+\beta_{CL}x_{CL} \Leftrightarrow$$

$$\Leftrightarrow -\frac{\beta_0+\beta_{CL}x_{CL}}{\beta_{RW}} = x_{RW}$$

Which is the line we are looking for.

```
# Task 4

fit = glm(sex ~ RW + CL, data = data, family = "binomial")
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
#summary(fit)
fits = predict(fit, data)


probabilities = fits
pLimit = 0.5
classifications = apply(as.matrix(probabilities), 1, function(row) {
  if (row > pLimit) {
    return(1)
  } else {
    return(0)
  }
})
#classificationsTest = log_reg(test, fit)


table(data$sex, classifications)
```

```
##         classifications
##           0  1
##   Female 98  2
##   Male    4 96
```

```
print("misclassification rate")
```

```
## [1] "misclassification rate"
```

```
1 - classificationRate(table(data$sex, classifications))
```

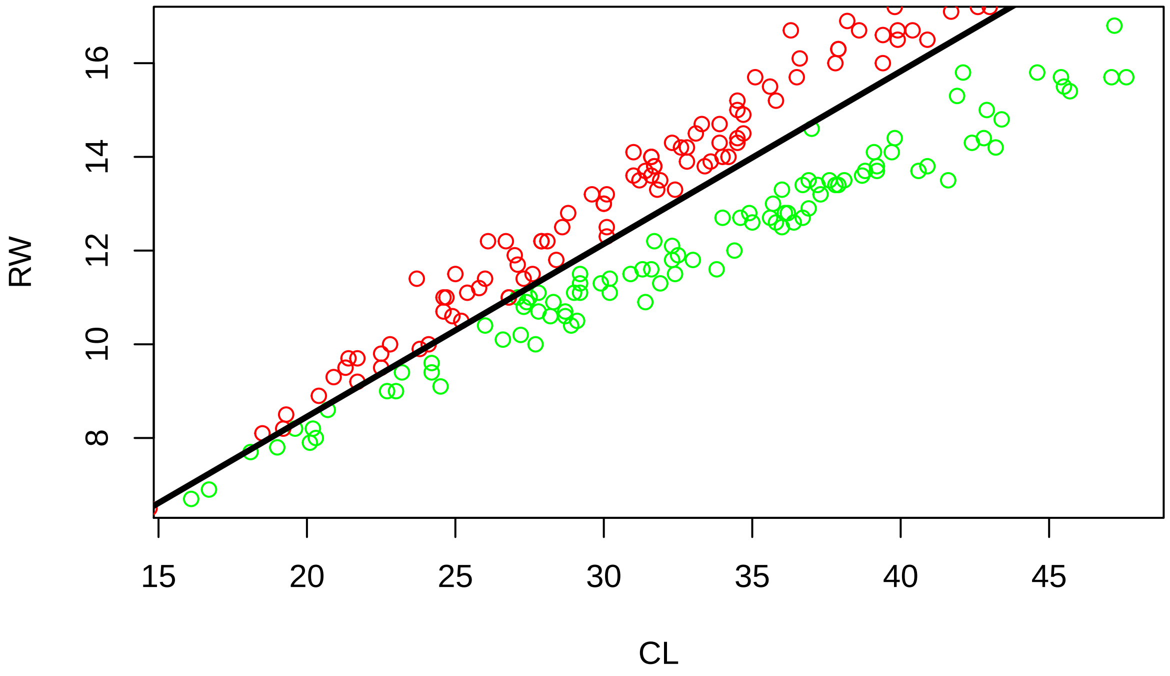
```
## [1] 0.03
```

```
fit$coefficients
```

```
## (Intercept)          RW          CL
##   13.616628  -12.563893    4.630747
```

```
# Report the equation of the decision boundary and draw it in the plot of the classified data
plot(data[classifications == 1,]$CL,
     data[classifications == 1,]$RW,
     col = "green", xlab = "CL", ylab ="RW")
points(data[classifications == 0,]$CL,
       data[classifications == 0,]$RW, col = "red")
# Equation of decision boundary: $0.5 = \frac{1}{1 + exp(-(intercept + beta1*RW + beta2*CL))}$
CLseq = seq(min(data$CL), max(data$CL), length = 1000)
Xseq = matrix(c(rep(1, length = 1000),CLseq), ncol = 2, nrow = 1000)

RWseq = -t(as.matrix(c(fit$coefficients[1],
                       fit$coefficients[3])))%*%t(Xseq)/fit$coefficients[2]
lines(x = CLseq, y = RWseq, lwd = 3)
```



2 Assignment 2

In assignment 2, our task was to use the data from the file `creditscoring.xls` to compare decision trees and Naïve in their abiity to predict whether a customer will be able to pay back the loan or not.

2.1 Task 1 and 2

Task 1 was to load and partition the data, a trivial task to perform given the lectures. Then, in task 2, we fit a decisision tree in two ways; one by using the Deviance measure, another by using the Gini measure.

```
classificationRate = function(confMatrix) {
  return(sum(diag(confMatrix))/sum(confMatrix))
```

```r
}
library(readxl)
data = read_excel("creditscoring.xls")
data$good_bad = factor(data$good_bad)
# Task 1

n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data[id,]
id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.25))
valid=data[id2,]
id3=setdiff(id1,id2)
test=data[id3,]

#install.packages("tree")
#library(rpart)

library(tree)

# Fitting with gini

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

trainClassifGini = factor(apply(predict(fitGini, train), 1, function(row) {
  return(names(which.max(row)))
}))
testClassifGini = factor(apply(predict(fitGini, test), 1, function(row) {
  return(names(which.max(row)))
}))

table(train$good_bad, trainClassifGini)
```

```
##       trainClassifGini
##         bad good
##   bad    72   75
##   good   45  308
```

```r
print("misclassification rate train gini")
```

```
## [1] "misclassification rate train gini"
```

```r
1 - classificationRate(table(train$good_bad, trainClassifGini))
```

```
## [1] 0.24
```

```r
table(test$good_bad, testClassifGini)
```

```
##       testClassifGini
##         bad good
##   bad    20   56
##   good   39  135
```

```
print("misclassification rate test gini")
```

```
## [1] "misclassification rate test gini"
```

```
1 - classificationRate(table(test$good_bad, testClassifGini))
```

```
## [1] 0.38
```

```
# fitting with deviance
fitDeviance = tree(good_bad ~., data = train, split = c("deviance"))

trainClassifDeviance = factor(apply(predict(fitDeviance, train), 1, function(row) {
  return(names(which.max(row)))
}))
testClassifDeviance = factor(apply(predict(fitDeviance, test), 1, function(row) {
  return(names(which.max(row)))
}))
table(train$good_bad, trainClassifDeviance)
```

```
##       trainClassifDeviance
##        bad good
##   bad   61   86
##   good  20  333
```

```
print("misclassification rate train deviance")
```

```
## [1] "misclassification rate train deviance"
```

```
1 - classificationRate(table(train$good_bad, trainClassifDeviance))
```

```
## [1] 0.212
```

```
table(test$good_bad, testClassifDeviance)
```

```
##       testClassifDeviance
##        bad good
##   bad   28   48
##   good  19  155
```

```
print("misclassification rate test deviance")
```

```
## [1] "misclassification rate test deviance"
```

```
1 - classificationRate(table(test$good_bad, testClassifDeviance))
```

```
## [1] 0.268
```

```
#Deviance is best => choose deviance!
```

We see that the deviance measure yields, based on the missclassification rate, significantly better results. Due to this, we shall use this in the upcoming tasks.

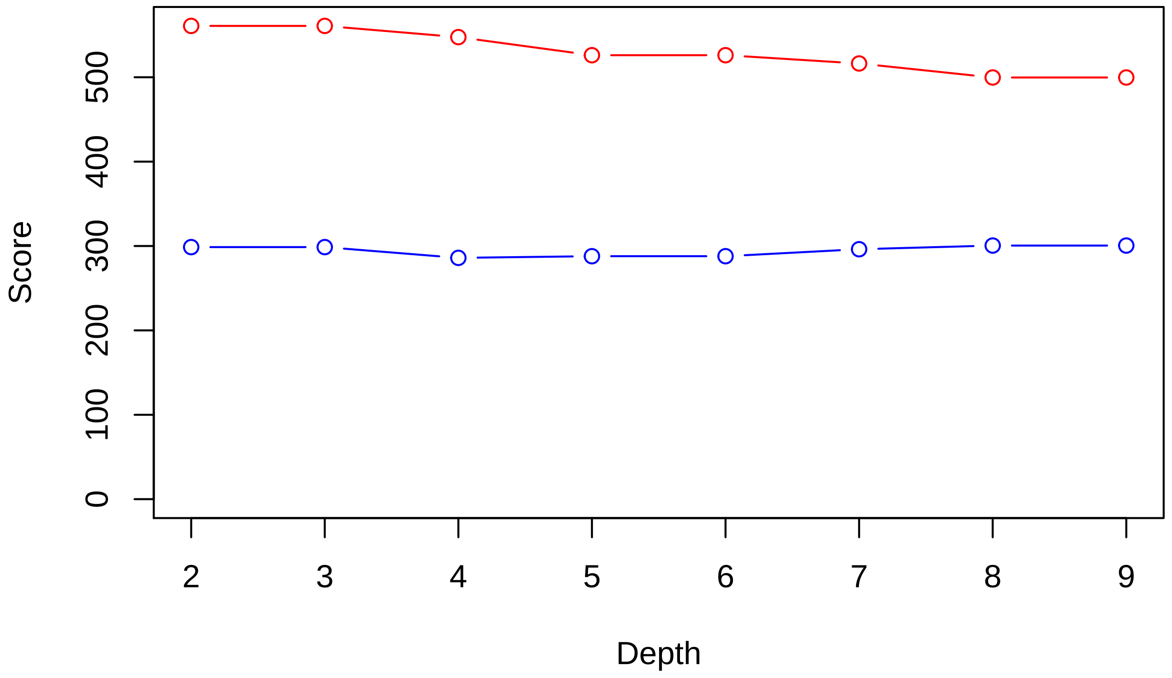
2.2 Task 3

Deciding the optimal tree depth, we can that it is 4. The optimal tree is printed below. We can also see that the misclassification rate is slightly improved, although not significantly.

```
# Task 3

fit = fitDeviance
```

```r
trainScore=rep(0,9)
testScore=rep(0,9)
for(i in 2:9) {
  prunedTree=prune.tree(fit,best=i)
  pred=predict(prunedTree, newdata=valid,
               type="tree")
  trainScore[i]=deviance(prunedTree)
  testScore[i]=deviance(pred)
}
plot(2:9, trainScore[2:9], type="b", col="red",
     ylim=c(0,max(trainScore, testScore)), xlab = "Depth", ylab = "Score")
points(2:9, testScore[2:9], type="b", col="blue")
```



```r
depth = which.min(testScore[2:9]) + 1
print(paste("Best depth is", depth))
```

```
## [1] "Best depth is 4"
```

```r
finalTree=prune.tree(fit, best=depth)
Yfit=predict(finalTree, newdata=valid,
             type="class")
table(valid$good_bad,Yfit)
```

```
##       Yfit
##        bad good
##   bad   23   54
##   good  12  161
```

```r
1 - classificationRate(table(valid$good_bad,Yfit))
```

```
## [1] 0.264
```

```r
print(finalTree)
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
## 1) root 494 598.00 good ( 0.2935 0.7065 )
##   2) savings < 2.5 346 446.70 good ( 0.3468 0.6532 )
##     4) duration < 43.5 325 405.90 good ( 0.3169 0.6831 )
##       8) history < 1.5 22  27.52 bad ( 0.6818 0.3182 ) *
##       9) history > 1.5 303 365.10 good ( 0.2904 0.7096 ) *
##     5) duration > 43.5 21  20.45 bad ( 0.8095 0.1905 ) *
##   3) savings > 2.5 148 134.40 good ( 0.1689 0.8311 ) *
```

2.3 Task 4

Using the training data to classify using Naïve Bayes, we can see that it is not as efficient as the tree using Deviance as measure of impurity.

```
# Task 4 - use naïve bayes
library(MASS)

#install.packages("e1071")
library(e1071)
fit = naiveBayes(good_bad ~.,data=train)
classTrain = predict(fit, train)
classTest = predict(fit, test)

table(train$good_bad, classTrain)
```

```
##       classTrain
##        bad good
##   bad   95   52
##   good  98  255
```

```
1 - classificationRate(table(train$good_bad, classTrain))
```

```
## [1] 0.3
```

```
table(test$good_bad, classTest)
```

```
##       classTest
##        bad good
##   bad   46   30
##   good  49  125
```

```
1 - classificationRate(table(test$good_bad, classTest))
```

```
## [1] 0.316
```

2.4 Task 5

Computing the ROC curves for the two different models, we can see in the plot that Naïve Bayes seem to perform slightly better. This is also confirmed by computing the AUC.

```
# Task 5 use optimal tree and naive bayes model to classify test data by using Y

Pi = as.matrix(seq(0,1, by = 0.05))

predictionsTest = predict(finalTree, test)
```

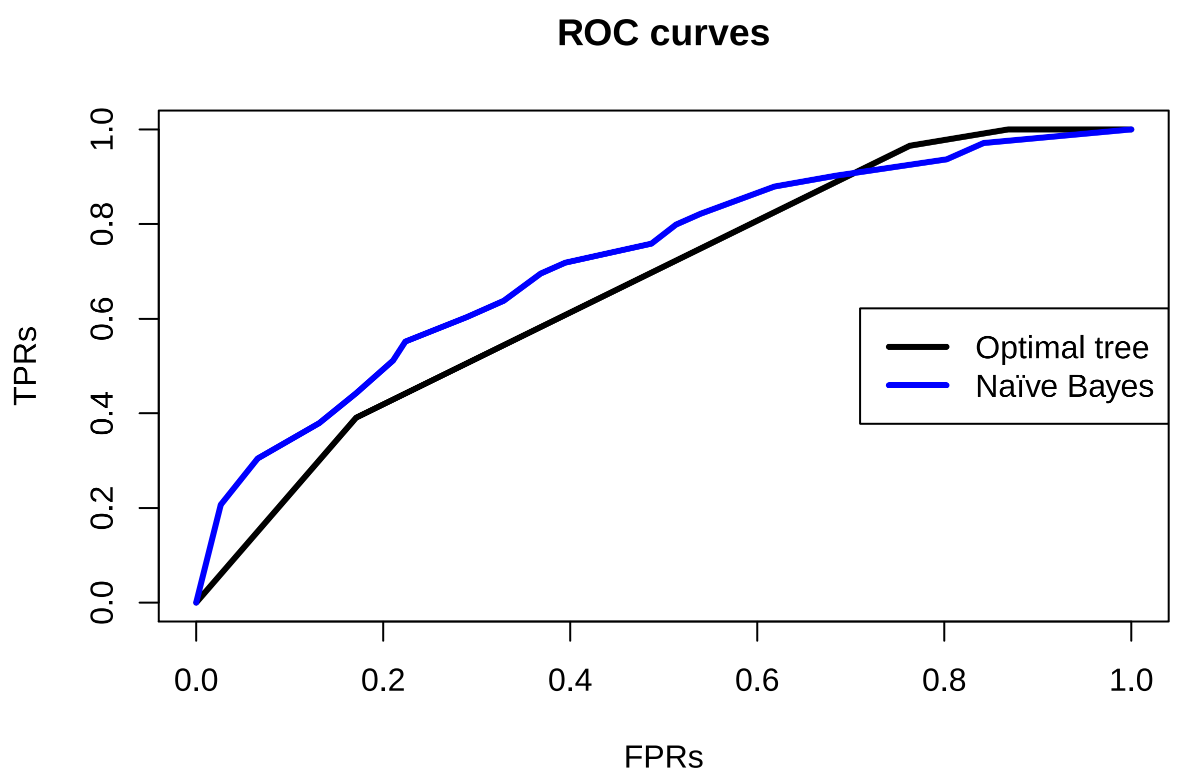
```r
predictionsTestBayes = predict(fit, test, "raw")

testClassifs = matrix(apply(as.matrix(Pi), 1, function(pi_val, preds) {
  return(ifelse(preds > pi_val, 2,1))
}, predictionsTest[,2]), nrow = nrow(predictionsTest),ncol = length(Pi))
testClassifsBayes = matrix(apply(as.matrix(Pi), 1, function(pi_val, preds) {
  return(ifelse(preds > pi_val, 2,1))
}, predictionsTestBayes[,2]), nrow = nrow(predictionsTestBayes),ncol = length(Pi))


TPR = function(predictions, labels) {
  N_plus = sum(labels == "good")
  labels = as.numeric(labels)
  return(sum(predictions == 2 & labels == predictions)/N_plus)
}

FPR = function(predictions, labels) {
  N_minus = sum(labels == "bad")
  labels = as.numeric(labels)
  return(sum(predictions == 2 & labels != predictions)/N_minus)
}
TPRs = apply(testClassifs,2,TPR, test$good_bad)
FPRs = apply(testClassifs,2, FPR, test$good_bad)
plot(x=FPRs, y = TPRs,
     type = "l", xlim = c(0,1), ylim = c(0,1),
     lwd = 3, main = "ROC curves")

TPRsBayes = apply(testClassifsBayes,2,TPR, test$good_bad)
FPRsBayes = apply(testClassifsBayes,2,FPR, test$good_bad)
lines(x=FPRsBayes, y = TPRsBayes, col = "blue", lwd = 3)
legend("right",
       lty = rep(1,2),
       col = c("black", "blue"),
       lwd = c(3,3),
       legend = c("Optimal tree","Naïve Bayes"))
```



```
AUC = function(TPR, FPR) {
  # TPR is y, FPR is x
  # Order after FPR
  xInd = order(FPR)
  x = FPR[xInd]
  y = TPR[xInd]
  area = 0
  for (i in 2:length(TPR)) {
    area = (x[i]-x[i-1])*(y[i] + y[i-1])/2 + area
  }
  return(area)
}

print("AUC bayes")
```

```
## [1] "AUC bayes"
```

```
AUC(TPRsBayes, FPRsBayes)
```

```
## [1] 0.7225499
```

```
print("AUC tree")
```

```
## [1] "AUC tree"
```

```
AUC(TPRs, FPRs)
```

```
## [1] 0.669994
```

2.5 Task 6

In task 6, the task was use a loss function and then use Naïve Bayes to decide. If one calculates on paper the decision rule, one arrives at

$$y_i = \begin{cases} good & \text{if } p(y_i = bad|x) < 10p(y_i = good|x) \\ 0 & \text{otherwise} \end{cases}$$

Since we want to avoid classifying bad ones as good incorrectly. This yields the result below.

```
# Task 6

# use loss matrix to decide.

# USE NAIVE BAYES - which was
predictionsTrain = predict(fit, train, "raw")
predictionsTest = predict(fit, test, "raw")

trainClassifsWLoss = apply(predictionsTrain, 1, function(row) {
  losses = c(1 - row[1], 10*(1 - row[2]))

  return(which.min(losses))
})

testClassifsWLoss = apply(predictionsTest, 1, function(row) {
  losses = c(1 - row[1], 10*(1 - row[2]))
  # c(bad, good)
  return(which.min(losses))
})

print("Training conf matrix rom task 4")
```

```
## [1] "Training conf matrix rom task 4"
```

```
table(train$good_bad, classTrain)
```

```
##       classTrain
##         bad good
##   bad    95   52
##   good   98  255
```

```
1 - classificationRate(table(train$good_bad, classTrain))
```

```
## [1] 0.3
```

```
print("Test conf matrix rom task 4")
```

```
## [1] "Test conf matrix rom task 4"
```

```
table(test$good_bad, classTest)
```

```
##       classTest
##         bad good
##   bad    46   30
##   good   49  125
```

```
1 - classificationRate(table(test$good_bad, classTest))
```

```
## [1] 0.316
```

```r
table(train$good_bad, trainClassifsWLoss)
```

```
##       trainClassifsWLoss
##          1   2
##   bad  137  10
##   good 263  90
```

```r
1 - classificationRate(table(train$good_bad, trainClassifsWLoss))
```

```
## [1] 0.546
```

```r
table(test$good_bad, testClassifsWLoss)
```

```
##       testClassifsWLoss
##          1   2
##   bad   71   5
##   good 122  52
```

```r
1 - classificationRate(table(test$good_bad, testClassifsWLoss))
```

```
## [1] 0.508
```

We see that we obtain a significantly worse result, with few bad classified as good, but a lot of good predicted as bad, since the loss function tends towards this.

3 Assignment 4

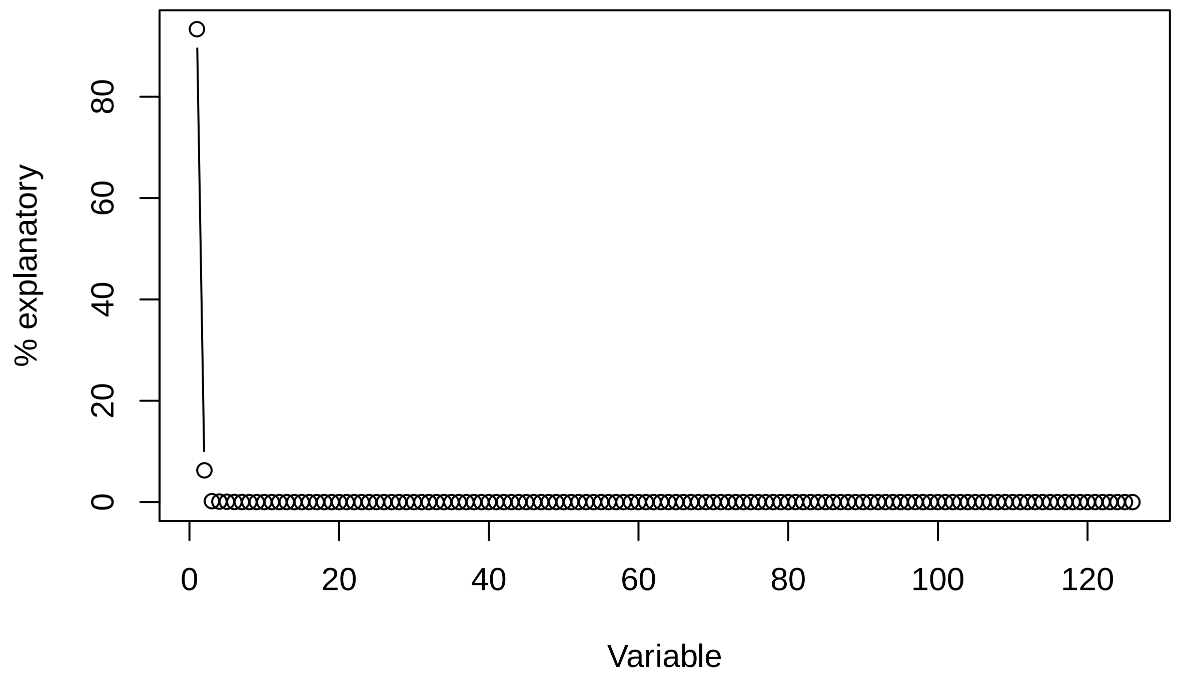
3.1 Task 1

Here, the task was to study how the near-infrared spectra of diesel fuels affects the viscosity. Performing a standard PCA, we can clearly see how 93 % is explained by the hidden factor, and 6 % by the second one, PC2.

```r
############################
####### ASSIGNMENT 4 #######
############################

data = read.csv("NIRSpectra.csv", sep = ";", dec = ",")
featSpace = data[,-c(ncol(data))]
result = prcomp(featSpace)
lambda=result$sdev^2
explanationFactor = lambda/sum(lambda)*100

names(explanationFactor) = seq(1,length(lambda),1)
plot(explanationFactor, type = "b", xlab = "Variable", ylab = "% explanatory")
```



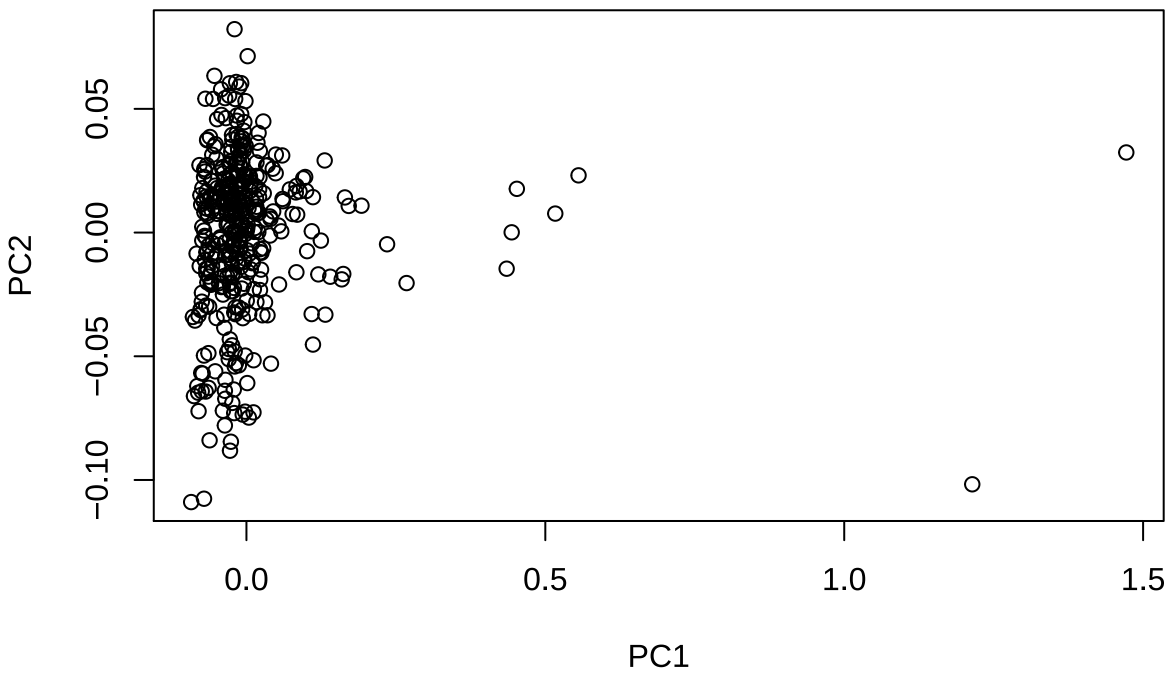
```r
explanationFactor = sort(explanationFactor, decreasing = T)

sumP = 0
i = 0
while (sumP < 99) {
  i = i + 1
  sumP = sumP + explanationFactor[i]


}

nrVarsToInvolve = i


plot(result$x[,1], result$x[,2], xlab = "PC1", ylab = "PC2")
```



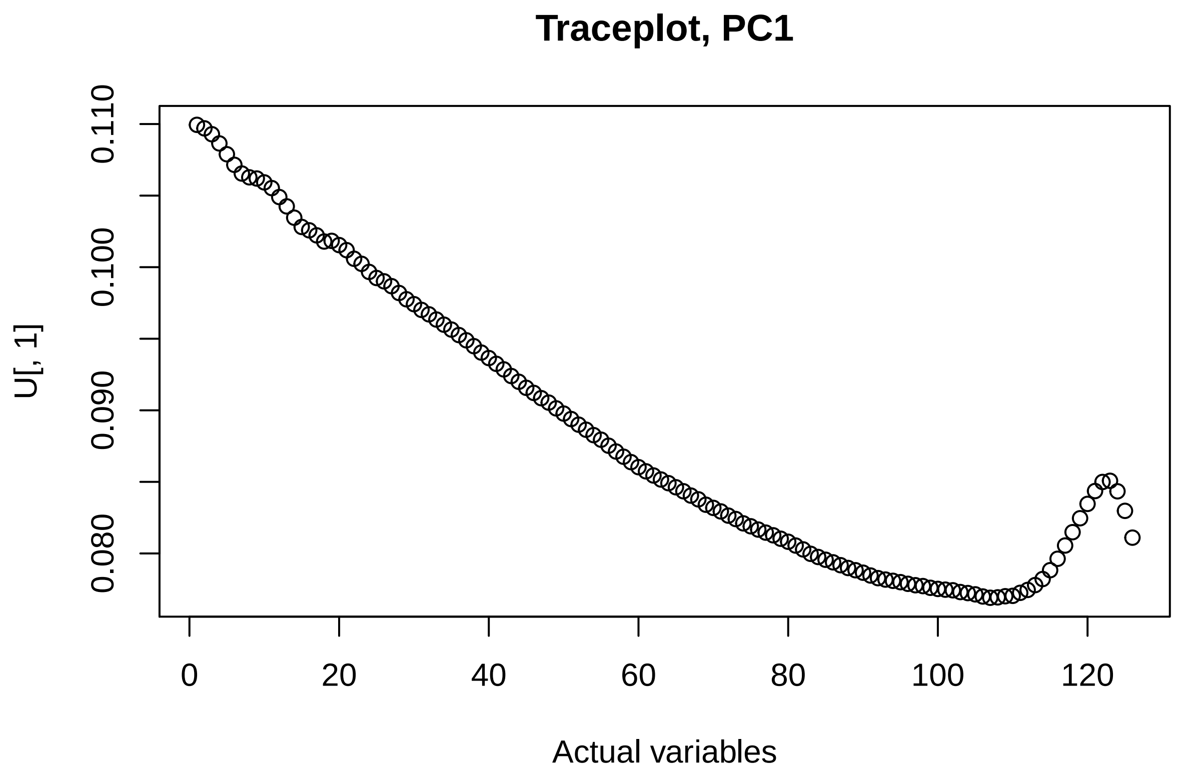
There are a few outliers as can be seen in the plot, although they are not many.

3.2 Task 2

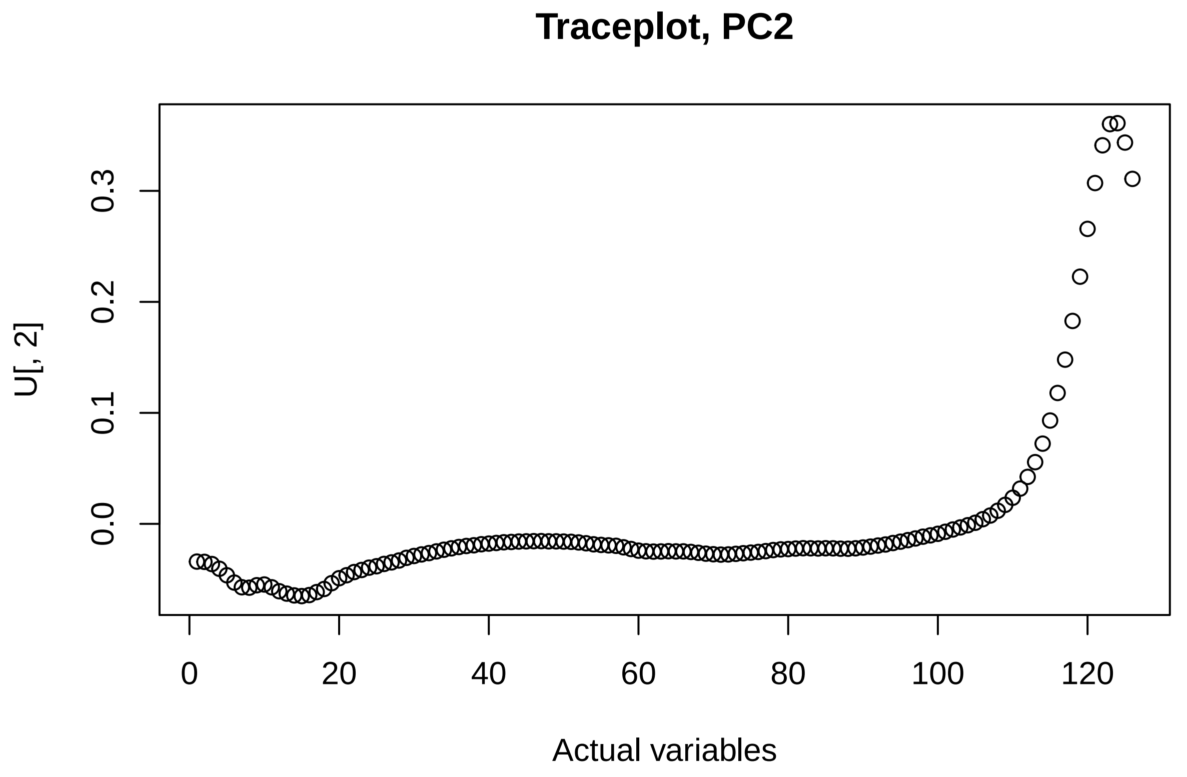
Investigating the trace plots of PC1 and PC2, we can clearly see that PC2 is mainly explained by the last variables. For PC1, we see a decreasing degree of explaining as the variable index increases, followed by a slight tip on the end.

```r
# Task 2: Make trace plots

U = result$rotation
plot(U[,1], main="Traceplot, PC1", xlab = "Actual variables")
```



```r
plot(U[,2],main="Traceplot, PC2", xlab = "Actual variables")
```



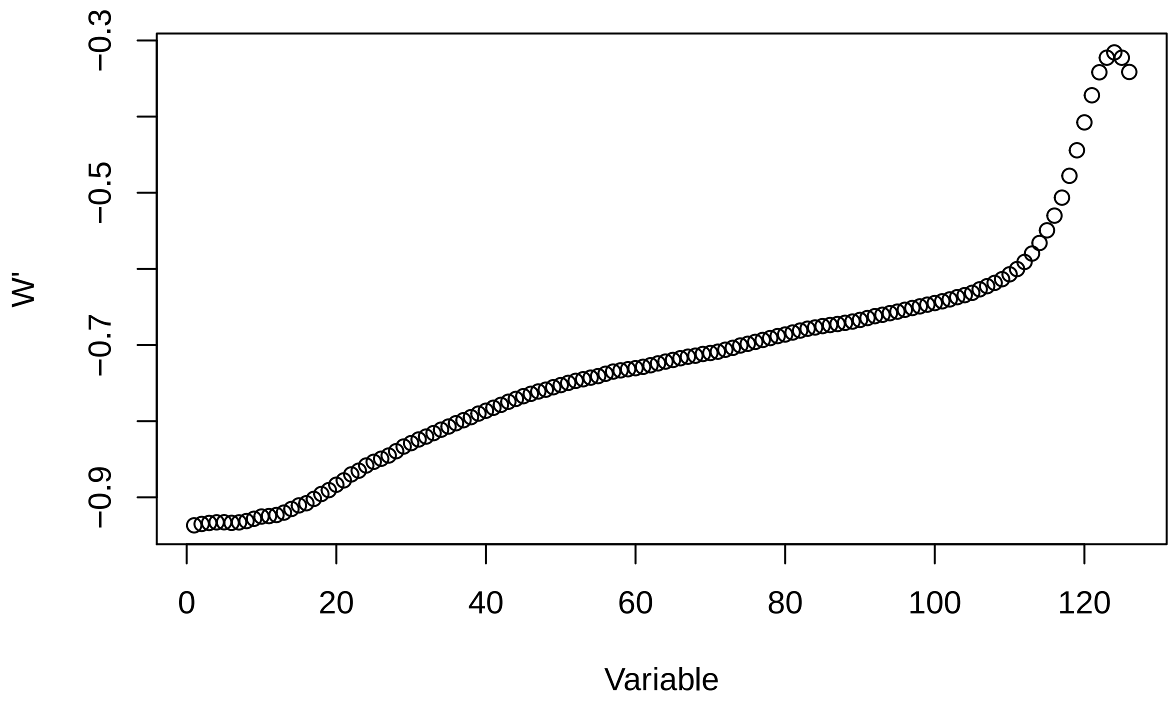
3.3 Task 3

Performing the last task, we can see that fastICA yields the same information, but becomes kind of the "inverse" to the PCA.

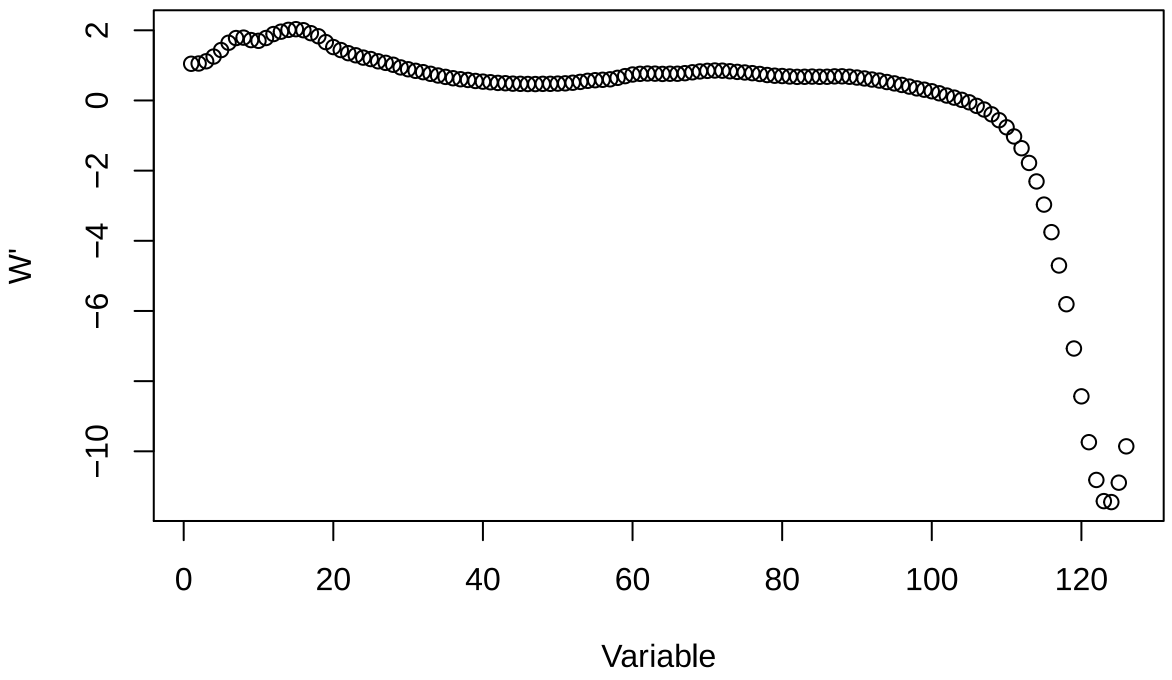
```r
# Task 3: Perform independent component analysis
#install.packages("fastICA")
library(fastICA)
#fICA = fastICA(as.matrix(featSpace),)
#X = as.matrix()

# a. Compute W' = K * W
set.seed(12345)
fICAResult = fastICA(as.matrix(featSpace), n.comp = nrVarsToInvolve)
W_prime = fICAResult$K%*%fICAResult$W

plot(W_prime[,1], xlab = "Variable", ylab = "W'")
```



```r
plot(W_prime[,2], xlab = "Variable", ylab = "W'")
```



W'
2
0
−2
−4
−6
−10
0
20
40
60
80
100
120
Variable