# Lab2

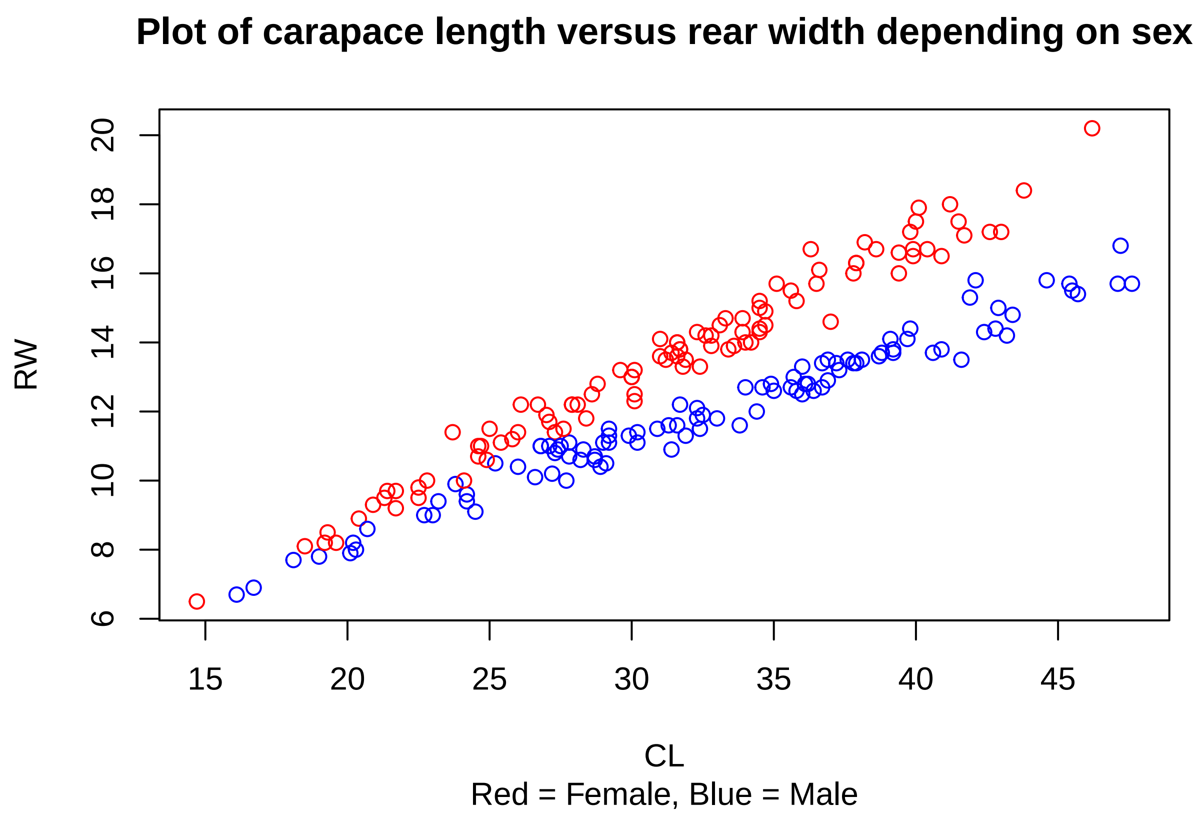
*Christian von Koch, Alice Velander and William Anzen*

*2019-12-08*

## Assignment 1 - Alice Velander

### Task 1

```r
Dataframe=read.csv("australian-crabs.csv")
n = length(Dataframe[,1])
CL = Dataframe$CL
RW = Dataframe$RW
plot(CL, RW, main="Plot of carapace length versus rear width depending on sex",
     sub="Red = Female, Blue = Male",
     col=c("red", "blue")[Dataframe$sex], xlab="CL", ylab="RW")
```



As seen in the scatter plot above, we can do an linear classification.

### Task 2

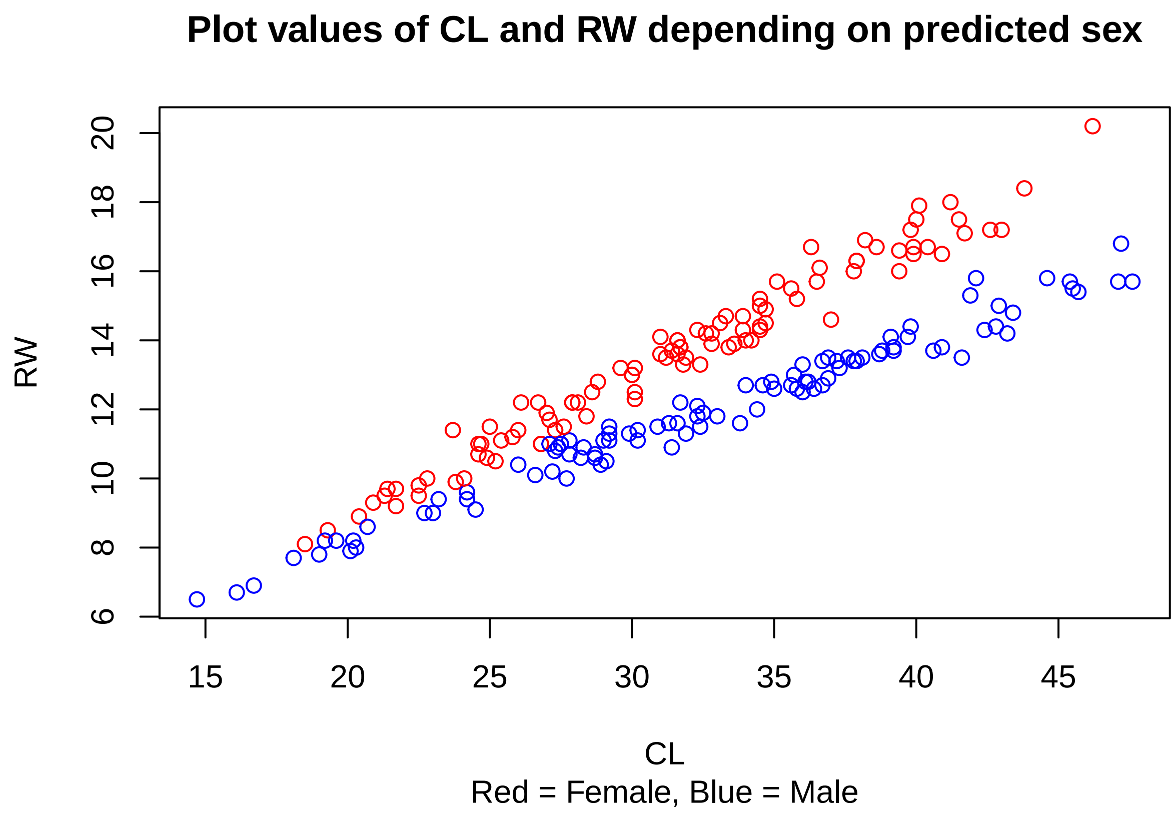
```r
#Create function for misclassification rate
missclass=function(conf_matrix, fit_matrix){
  n=length(fit_matrix[,1])
  return(1-sum(diag(conf_matrix))/n)
}

#LDA analysis with target Sex, and features CL and RW and proportional prior
library("MASS")
model = lda(sex ~ CL+RW, data=Dataframe)
predicted = predict(model, data=Dataframe)
confusion_matrix = table(Dataframe$sex, predicted$class)
misclass = missclass(confusion_matrix, Dataframe)
print(confusion_matrix)
```

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

```
print(misclass)
```

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

```
plot(CL, RW, main="Plot values of CL and RW depending on predicted sex",
     sub="Red = Female, Blue = Male",
     col=c("red", "blue")[predicted$class], xlab="CL", ylab="RW")
```



Misclassification rate = 0.035
Since we have very kind data that is easy to separate, we get a very similar plot as in step 1. We can also see that the prediction with LDA was good because of the low misclassification rate.

**Task 3**

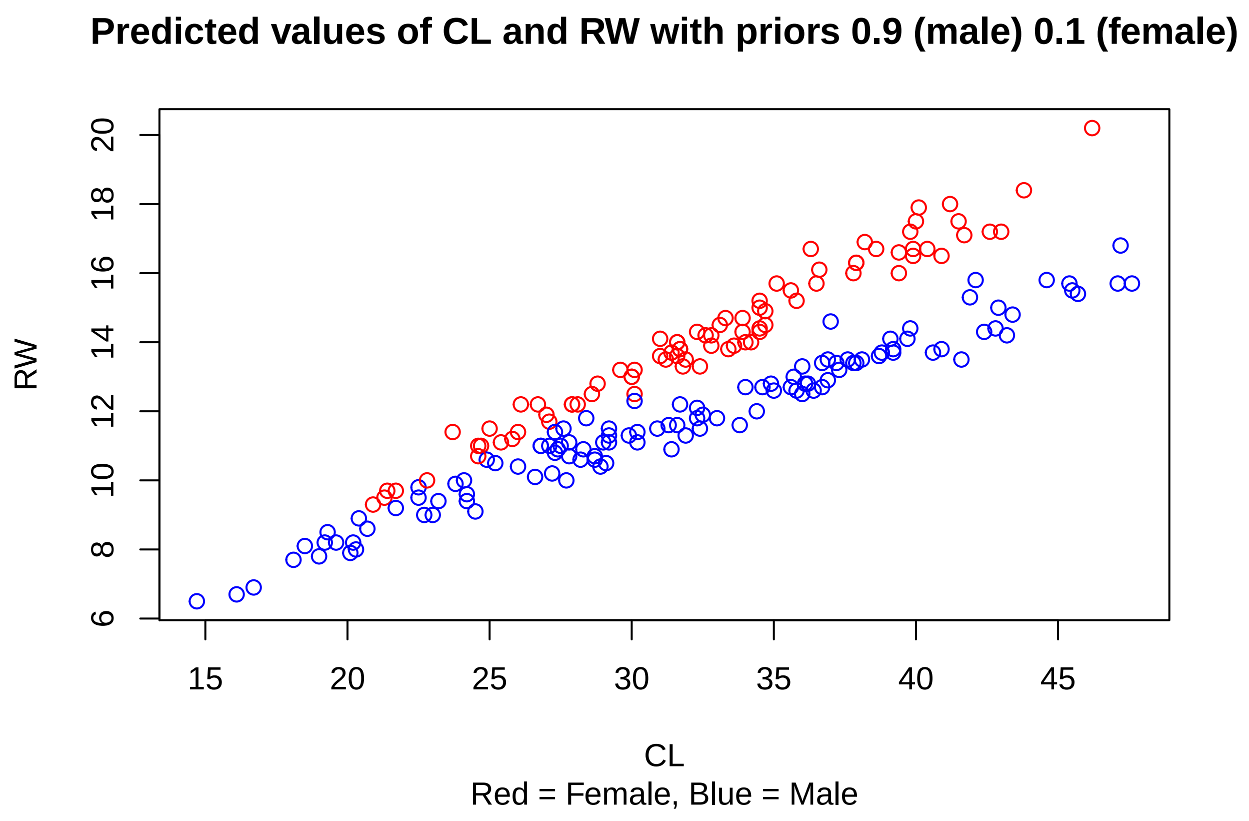
```
#Repeat step 2 but use priors p(Male)=0.9 and p(Female)=0.1
model2 = lda(sex ~ CL+RW, data=Dataframe, prior=c(1,9)/10)
predicted2 = predict(model2, data=Dataframe)
confusion_matrix2 = table(Dataframe$sex, predicted2$class)
misclass2 = missclass(confusion_matrix2, Dataframe)
print(confusion_matrix2)
```

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

```
print(misclass2)
```

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

```
plot(CL, RW, main="Predicted values of CL and RW with priors 0.9 (male) 0.1 (female)"
     , sub="Red = Female, Blue = Male", col=c("red", "blue")[predicted2$class], xlab="CL",
     ylab="RW")
```



Misclassification rate = 0.08
From the plots there are more males than females for the second prediction which is explained by the new prior. The misclassification rate is now slightly worse, making us think that the new prior does not help the prediction. This results in more females incorrectly predicted as males, and the likelihood of a male being falsely predicted as a female is low.

**Task 4**

```
#Repeat step 2 but now with logistic regression
model3 = glm(sex ~ CL+RW, data=Dataframe, family='binomial')
```

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

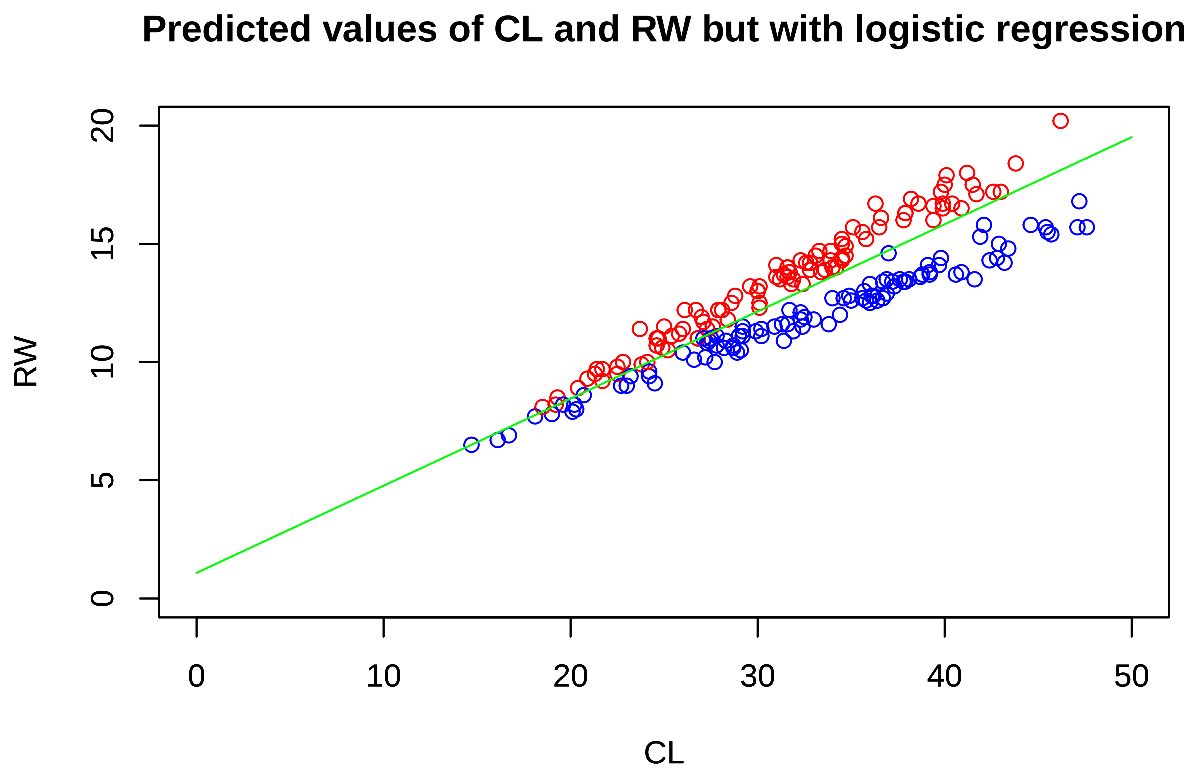
```
predicted3 = predict(model3, newdata=Dataframe, type='response')
sexvector = c()
for (i in predicted3) {
  if (i>0.5) {
    sexvector = c(sexvector, 'Male')
  } else {
    sexvector = c(sexvector, 'Female')
  }
}
sexvector_factor = as.factor(sexvector)
confusion_matrix3 = table(Dataframe$sex, sexvector_factor)
misclass3 = missclass(confusion_matrix3, Dataframe)
print(confusion_matrix3)
```

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

```
print(misclass3)
```

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

```
plot(CL, RW, main="Predicted values of CL and RW but with logistic regression",
     col=c("red", "blue")[sexvector_factor], xlab="CL", ylab="RW", xlim=c(0,50),
     ylim=c(0,20))

boundaryline = function(length, coefficientvector, prior) {
  return(-coefficientvector[1]/coefficientvector[3]-
           (coefficientvector[2]/coefficientvector[3])*length+
           log(prior/(1-prior))/coefficientvector[3])
}
par(new=TRUE)
curve(boundaryline(x, model3$coefficients, 0.5), xlab="CL", ylab="RW", col="green",
      from=0, to=50, xlim=c(0,50), ylim=c(0,20),
      sub="Red = Female, Blue = Male, Green = Boundaryline")
```



Red = Female, Blue = Male, Green = Boundaryline

When using logistic regression the results are similar as the first built model with LDA. This is simply a coincident and no real conclusion can be drawn regarding the exact same misclassification rate except from that the models seem to classify the data in the same way. When comparing which data points that are classified as females and males in the two models it can be concluded that the model using logistic regression classifies the data in a way which enables a boundary line more distincly. This is due to the characteristics of the logistic regression model. The equation for the decision boundary is as follows:

$R\hat{W}$=-($\beta_0$+$\beta_1$)/$\beta_2$*CL

## Assignment 2 - Christian von Koch

### Task 1

```
#1: Read data and divide into train, validation and test sets

library("tree")

data=read.csv2("creditscoring.csv")
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,]

#Create function for misclassification rate
misclass=function(conf_matrix, fit_matrix){
  n=length(fit_matrix[,1])
  return(1-sum(diag(conf_matrix))/n)
}
```

The data is divided into 50 % training data, 25 % validation data and 25 % test data.

**Task 2**

```
#2: Fit a decision tree to train data using the measures of impurity gini and deviance.
#Report misclass rates and choose optimal measure moving forward.

fit_deviance=tree(good_bad~., data=train, split="deviance")
predicted_deviance=predict(fit_deviance, newdata=test, type="class")
confusionmatrix_deviance=table(test$good_bad, predicted_deviance)
misclass_deviance=misclass(confusionmatrix_deviance, test)
print(confusionmatrix_deviance)
```

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

```
print(misclass_deviance)
```

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

```
fit_gini=tree(good_bad~., data=train, split="gini")
predicted_gini=predict(fit_gini, newdata=test, type="class")
confusionmatrix_gini=table(test$good_bad, predicted_gini)
misclass_gini=misclass(confusionmatrix_gini, test)
print(confusionmatrix_gini)
```

```
##       predicted_gini
##        bad good
##   bad   18   58
##   good  34  140
```

```
print(misclass_gini)
```

```
## [1] 0.368
```

```
#Deviance has best misclass score
```

It can be concluded from the misclassification rates that the split method deviance, classifies the data in a better way than the split method gini. Since the method deviance performed better it will be the choses splitting method in the following steps.

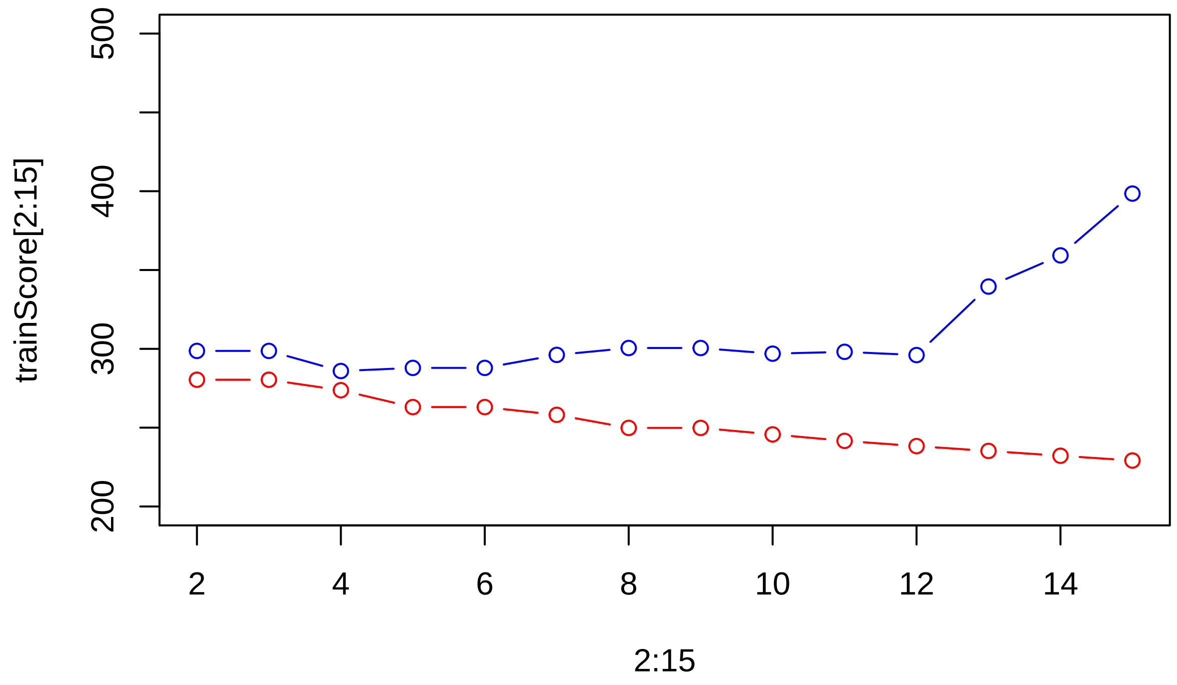
**Task 3**

```
#3: Use training and valid data to choose optimal tree depth. Present graphs of the
#dependence of deviances for training and validation data on the number of leaves.
#Report optimal tree, report it's depth and variables used bytree. Estimate
#misclassification rate for the test data.
```

```r
fit_optimaltree=tree(good_bad~., data=train, split="deviance")
summary(fit_optimaltree)
```

```
##
## Classification tree:
## tree(formula = good_bad ~ ., data = train, split = "deviance")
## Variables actually used in tree construction:
## [1] "savings"  "duration" "history"  "age"      "purpose"  "amount"   "resident"
## [8] "other"
## Number of terminal nodes:  15
## Residual mean deviance:  0.9569 = 458.3 / 479
## Misclassification error rate: 0.2105 = 104 / 494
```

```r
trainScore=rep(0,15)
testScore=rep(0,15)
for(i in 2:15){
  prunedTree=prune.tree(fit_optimaltree, best=i)
  pred=predict(prunedTree, newdata=valid, type="tree")
  #Divide by two since double of data points
  trainScore[i]=deviance(prunedTree)/2
  testScore[i]=deviance(pred)
}
plot(2:15, trainScore[2:15], type="b", col="red", ylim=c(200,500))
points(2:15, testScore[2:15], type="b", col="blue")
```



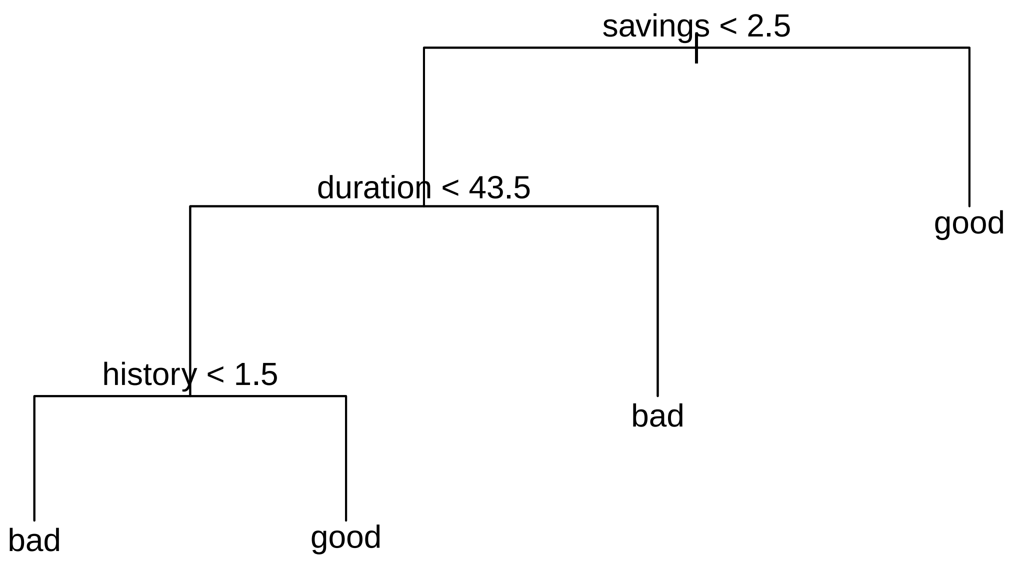
```r
min_deviance=min(testScore[2:15])
print(min_deviance)
```

```
## [1] 285.9425
```

```r
optimal_leaves=which(testScore[1:15] == min_deviance)
print(optimal_leaves)
```

```
## [1] 4
```

```r
#Optimal no of leaves is 4
finalTree=prune.tree(fit_optimaltree, best=4)
plot(finalTree)
text(finalTree, pretty=0)
```



```r
#Final tree contains variables savings, duration and history. Since 3 vars => Depth of
#tree is 3.
predicted_test=predict(finalTree, newdata=test, type="class")
confusionmatrix_test=table(test$good_bad, predicted_test)
misclass_test=misclass(confusionmatrix_test, test)
print(confusionmatrix_test)
```

```
##       predicted_test
##         bad good
##   bad    18   58
##   good    6  168
```

```
print(misclass_test)
```

```
## [1] 0.256
```

The tree with the lowest deviance used 4 leaves which is the optimal tree. The variables used by the tree is savings, duration and history, and the depth of the tree is 3. As we can see from the tree the classification checks the feature saving first and if the feature is above or equal to 2.5, the customer will be classified as *good*. Otherwise the model moves down in the tree to check for duration. If duration is above or equal to 43.5, the customer will be classified as *bad*, otherwise the model moves even further down in the tree. Finally, if history is above or equal to 1.5 the customer will be classified as *good*, otherwise the customer will be classified as *bad*. The misclassification rate for the test data is 0.256.

**Task 4**

```
#4: Use traning data to perform classification using Naives bayes and report the confusion
#matrices and misclassification rates for the traning and for the test data. Compare with
#results from previous steps.

#Load libraries
library(MASS)
library(e1071)
fit_naive=naiveBayes(good_bad~., data=train)
#Create function for predicting and creating confusion matrice and printing
#misclassification rate
compute_naive=function(model,data){
  predictedNaive=predict(model, newdata=data, type="class")
  confusionmatrixNaive=table(data$good_bad,predictedNaive)
  misclass = misclass(confusionmatrixNaive, data)
  print(confusionmatrixNaive)
  print(misclass)
  return(predictedNaive)
}
predictedNaive_train=compute_naive(fit_naive,train)
```

```
##       predictedNaive
##         bad good
##   bad    95   52
##   good   98  255
## [1] 0.3
```

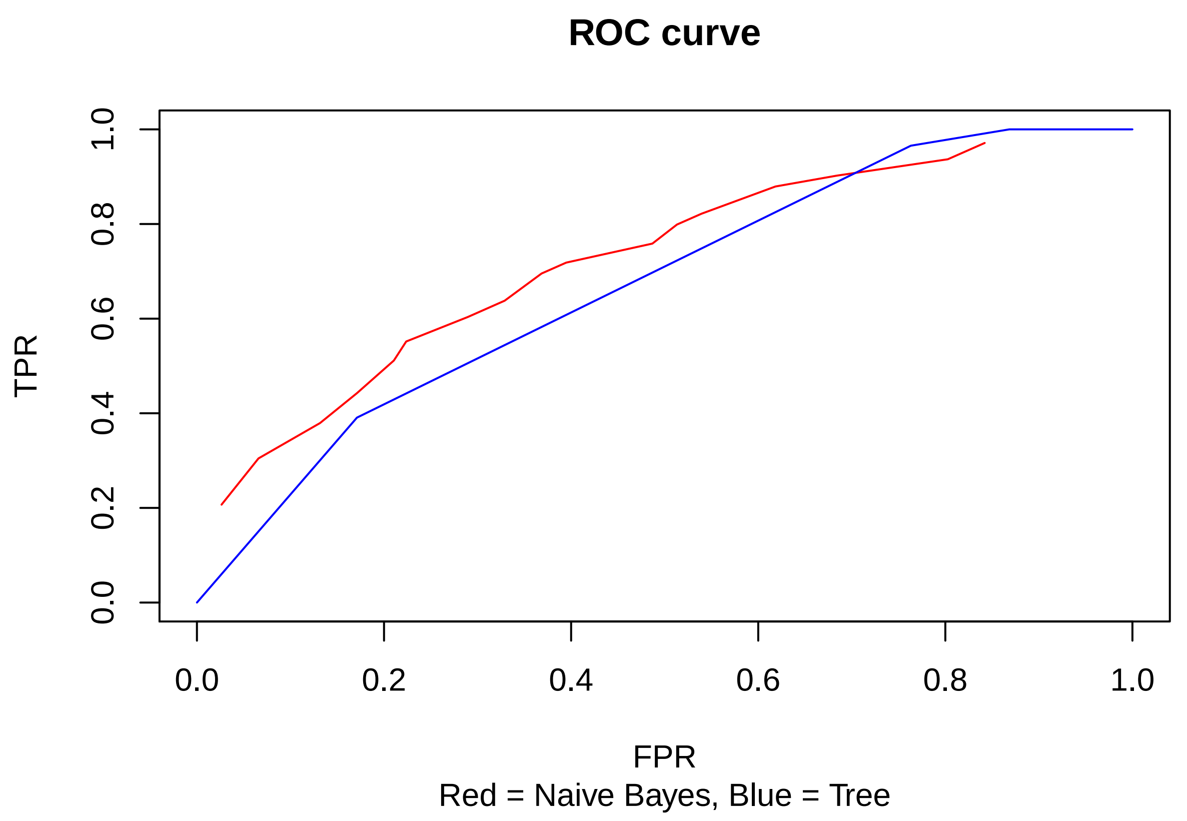
```
predictedNaive_test=compute_naive(fit_naive, test)
```

```
##       predictedNaive
##         bad good
##   bad    46   30
##   good   49  125
## [1] 0.316
```

With the Naïve Bayes method the misclassification rate is higher than what was concluded in step 3 (using decision trees). The misclassification rate for test data for the Naïve Bayes method is 0.316 and the misclassification rate for the decision tree method from step 3 is 0.256. This indicates that the decision tree method classifies the data more accurately than what the model which uses the Naïve Bayes method does.

**Task 5**

```r
#5: Use optimal tree and Naives Bayes to classify the test data by using principle:
#classified as 1 if bigger than 0.05, 0.1, 0.15, ..., 0.9, 0.95. Compute the TPR
#and FPR for two models and plot corresponsing ROC curves.

#Writing function for classifying data
class=function(data, class1, class2, prior){
  vector=c()
  for(i in data) {
    if(i>prior){
      vector=c(vector,class1)
    } else {
      vector=c(vector,class2)
    }
  }
  return(vector)
}

x_vector=seq(0.05,0.95,0.05)
tpr_tree=c()
fpr_tree=c()
tpr_naive=c()
fpr_naive=c()
treeVector=c()
treeConfusion = c()
naiveConfusion = c()
treeClass = c()
naiveClass = c()
#Reusing optimal tree found in task 3 but returntype is response instead
predictTree=data.frame(predict(finalTree, newdata=test, type="vector"))
predictNaive=data.frame(predict(fit_naive, newdata=test, type="raw"))
for(prior in x_vector){
  treeClass = class(predictTree$good, 'good', 'bad', prior)
  treeConfusion=table(test$good_bad, treeClass)
  if(ncol(treeConfusion)==1){
    if(colnames(treeConfusion)=="good"){
      treeConfusion=cbind(c(0,0), treeConfusion)
    } else {
      treeConfusion=cbind(treeConfusion,c(0,0))
    }
  }
  totGood=sum(treeConfusion[2,])
  totBad=sum(treeConfusion[1,])
  tpr_tree=c(tpr_tree, treeConfusion[2,2]/totGood)
  fpr_tree=c(fpr_tree, treeConfusion[1,2]/totBad)
  naiveClass=class(predictNaive$good, 'good', 'bad', prior)
  naiveConfusion=table(test$good_bad, naiveClass)
  if(ncol(naiveConfusion)==1){
    if(colnames(naiveConfusion)=="good"){
      naiveConfusion=cbind(c(0,0), naiveConfusion)
    } else {
      naiveConfusion=cbind(naiveConfusion,c(0,0))
    }
  }
  totGood=sum(naiveConfusion[2,])
  totBad=sum(naiveConfusion[1,])
  tpr_naive=c(tpr_naive, naiveConfusion[2,2]/totGood)
  fpr_naive=c(fpr_naive, naiveConfusion[1,2]/totBad)
}
#Plot the ROC curves
plot(fpr_naive, tpr_naive, main="ROC curve", sub="Red = Naive Bayes, Blue = Tree",
     type="l", col="red", xlim=c(0,1), ylim=c(0,1), xlab="FPR", ylab="TPR")
points(fpr_tree, tpr_tree, type="l", col="blue")
```



```
#Naive has greatest AOC => should choose Naive
```

From the ROC-curve we can see that the total area under the curve (AOC) is the biggest for the Naïve Bayes method. Therefore this method should be the one to use instead of the decision tree method.

**Task 6**

```
#6: Repeate Naive Bayes with loss matrix punishing with factor 10 if predicting good when
#bad and 1 if predicting bad when good.
naiveModel=naiveBayes(good_bad~., data=train)
train_loss=predict(naiveModel, newdata=train, type="raw")
test_loss=predict(naiveModel, newdata=test, type="raw")
confusion_trainLoss=table(train$good_bad, ifelse(train_loss[,2]/train_loss[,1]>10, "good",
                                                 "bad"))
misclass_trainLoss=misclass(confusion_trainLoss, train)
```

```r
print(confusion_trainLoss)
```

```
##
##        bad good
##   bad  137   10
##   good 263   90
```

```r
print(misclass_trainLoss)
```

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

```r
confusion_testLoss=table(test$good_bad, ifelse(test_loss[,2]/test_loss[,1]>10, "good",
                                               "bad"))
misclass_testLoss=misclass(confusion_testLoss, test)
print(confusion_testLoss)
```

```
##
##        bad good
##   bad   71    5
##   good 122   52
```

```r
print(misclass_testLoss)
```

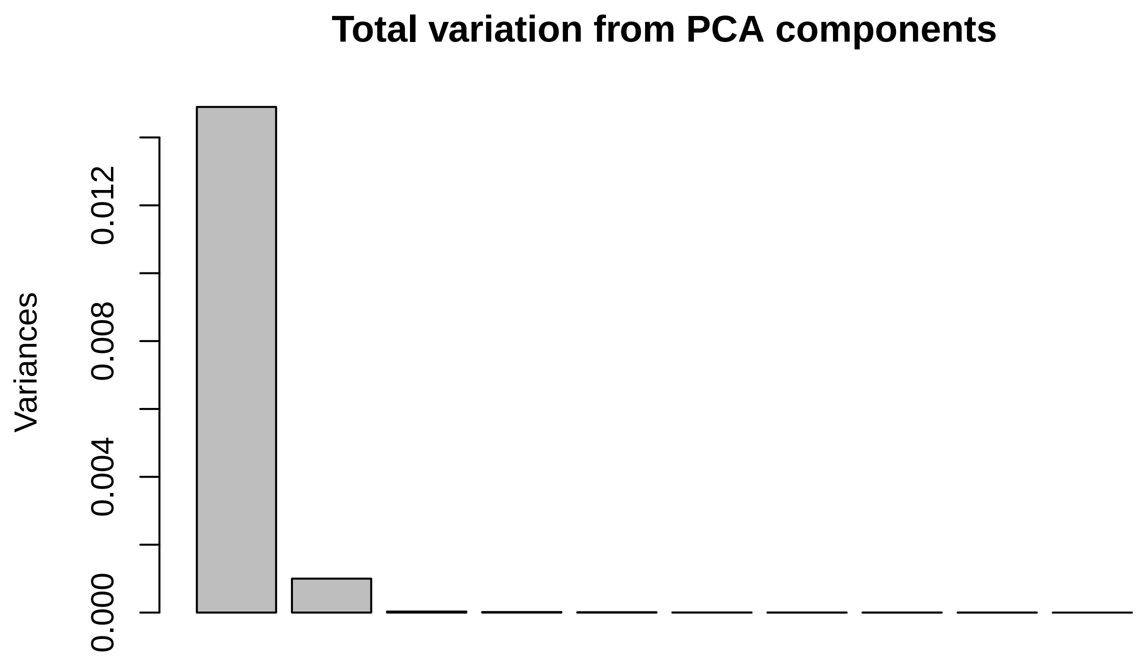
```
## [1] 0.508
```

From the confusion matrices and the misclassification rates it is noteable that the misclassification rates have gone up. More datapoints are being classified incorrectly. However, this is due to the defined loss matrix which describes a bigger loss when classifying a customer as *good* when the customer in fact was *bad*. This can also be seen in the confusion matrices where there are few customers being classified as *good* incorrectly (from 52 to 10 in train data and from 30 to 5 in test data). When the model in fact classifies a customer as *good* it is more often correct than not. In comparison it is noteable that the model is more often wrong regarding the classification *bad* which is due to the defined loss when misclassifying a customer as *bad* which is less than when misclassifying a customer as *good*.

## Assignment 4 - William Anzén

### Task 1

```r
#1: Read data

data=read.csv2("NIRspectra.csv")
data$Viscosity=c()
n=dim(data)[1]

#1: Conduct standard PCA using the feature space and provide a plot explaining how much
#variation is explained by each feature. Provide plot that show the scores of PC1 vs PC2.
#Are there unusual diesel fuels according to this plot.
pcaAnalysis=prcomp(data)
#Eigenvalues
lambda=pcaAnalysis$sdev^2
#Proportion of variation
propVar= lambda/sum(lambda)*100
screeplot(pcaAnalysis, main="Total variation from PCA components")
```



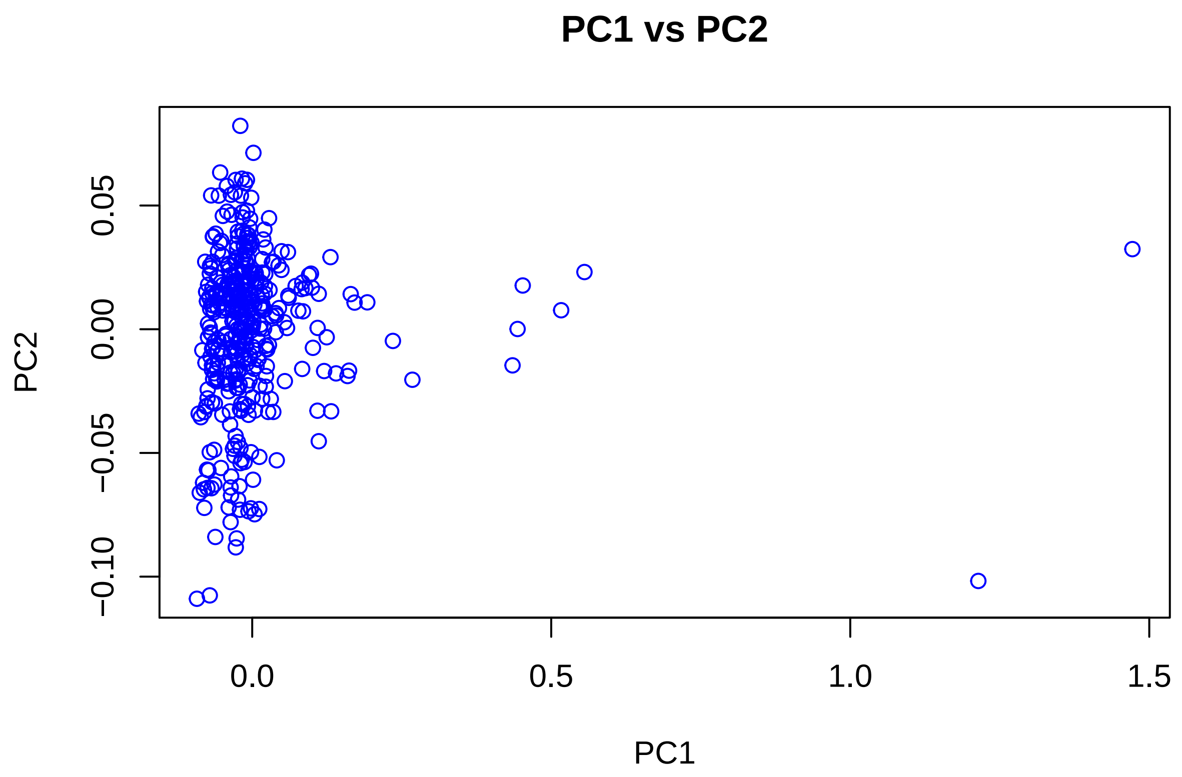
```
noOfVars=1
sumOfVariation=propVar[noOfVars]
while(sumOfVariation<99){
  noOfVars=noOfVars+1
  sumOfVariation=sumOfVariation+propVar[noOfVars]
}
#Print number of variables used and total variation
print(noOfVars)
```

```
## [1] 2
```

```
print(sumOfVariation)
```

```
## [1] 99.5957
```

```
#Print PC1 and PC2 in plot
plot(pcaAnalysis$x[,1],pcaAnalysis$x[,2], type="p", col="blue", main="PC1 vs PC2",
     xlab="PC1", ylab="PC2")
```

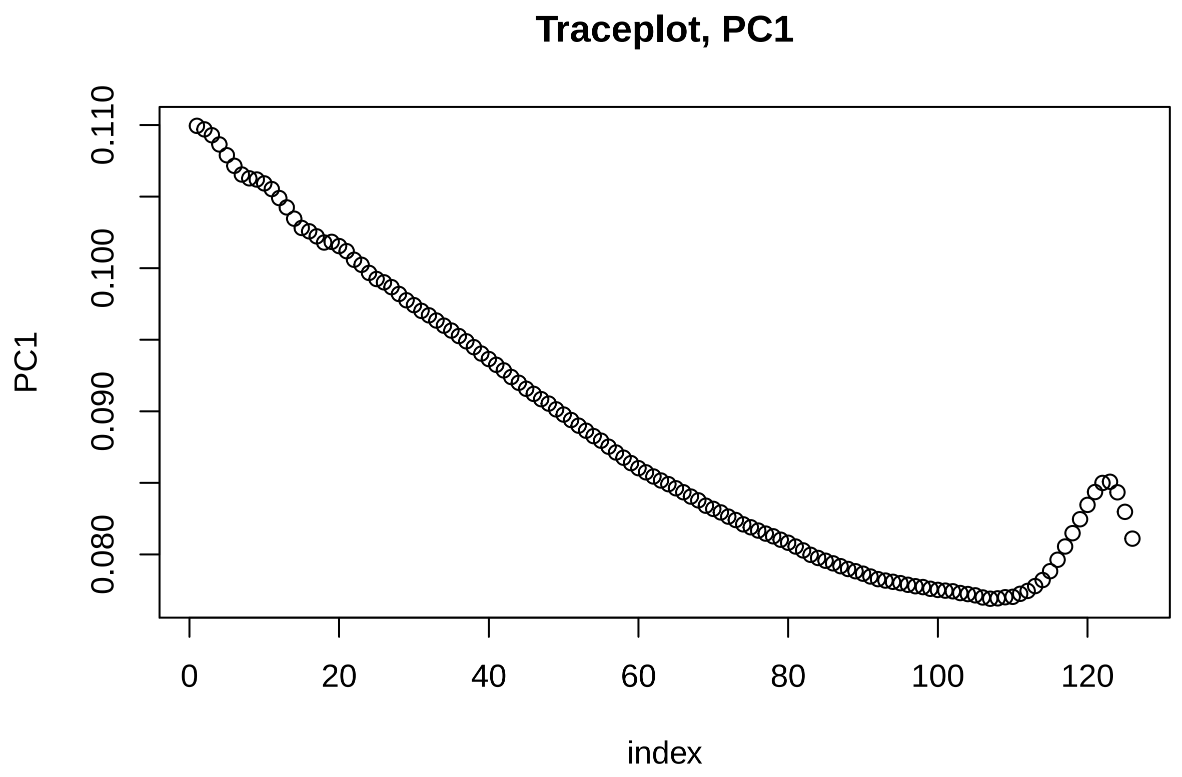


```
#We can see from the graph that the data is very accurately described by PC1.
```

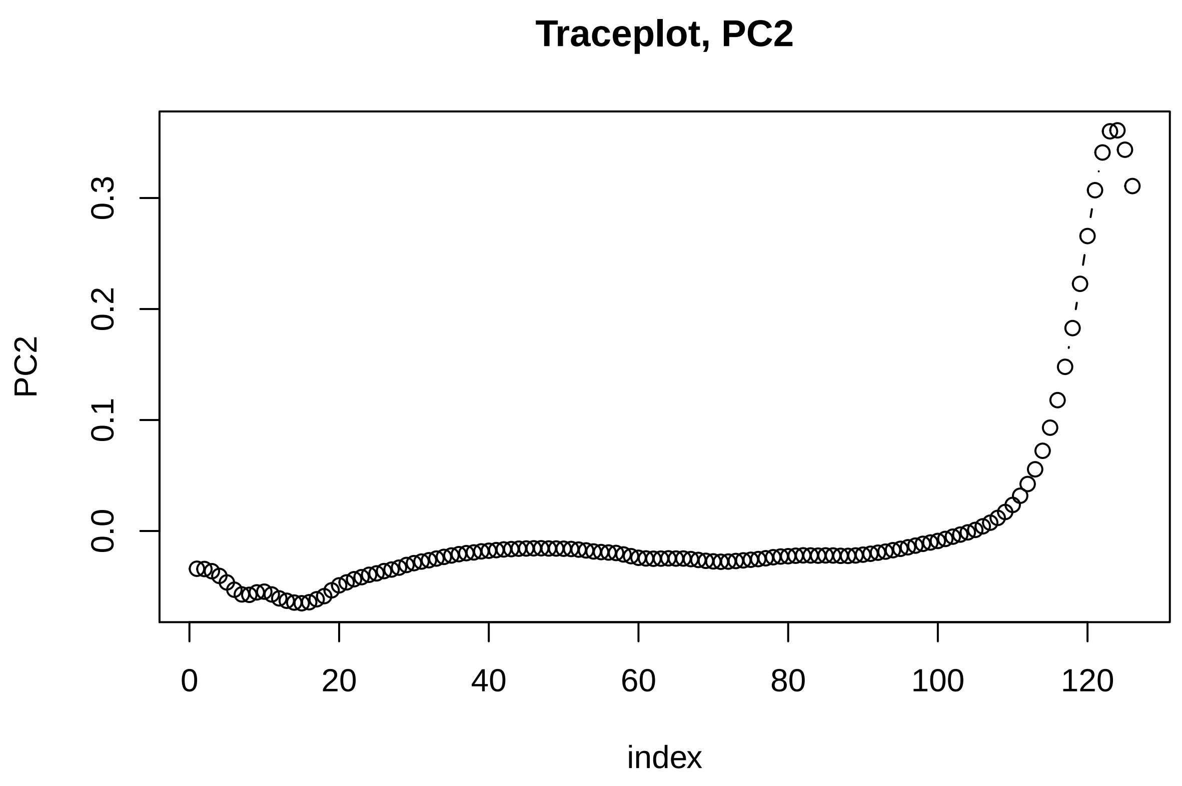
The histogram above shows that only PC1 and PC2 have a significant effect on the variance. This tells us that we need to extract these PCs. PC1 captures 93.3% of the variance and PC2 captures 6.2%, giving us a total over 99% of the variance captured with these 2 PCs.
We have a couple of outliers with a big variance on PC1. A couple of points around PC1= 0.5 and 2 points above PC=1.

**Task 2**

```
#2: Make trace plots of the loadings of the components selected in step 1. Is there any
#principle component that is explaines by mainly a few original features?

U=pcaAnalysis$rotation
plot(U[,1], main="Traceplot, PC1", xlab="index", ylab="PC1", type="b")
```



```r
plot(U[,2], main="Traceplot, PC2", xlab="index", ylab="PC2", type="b")
```



```
#We can see from graph that PC2 is not described by so many original features since it is
#close to zero for many of the features. The last 30 or so variables have an effect on PC2.
```

Most of the features have an impact on PC1 as we can see in the graph to the left. While most of the features do not have an impact on PC2 as most values for the different features are around 0. There are a couple of outliers that actually affect PC2 around feature number 115-126, as well as some in the beginning with values little below 0.

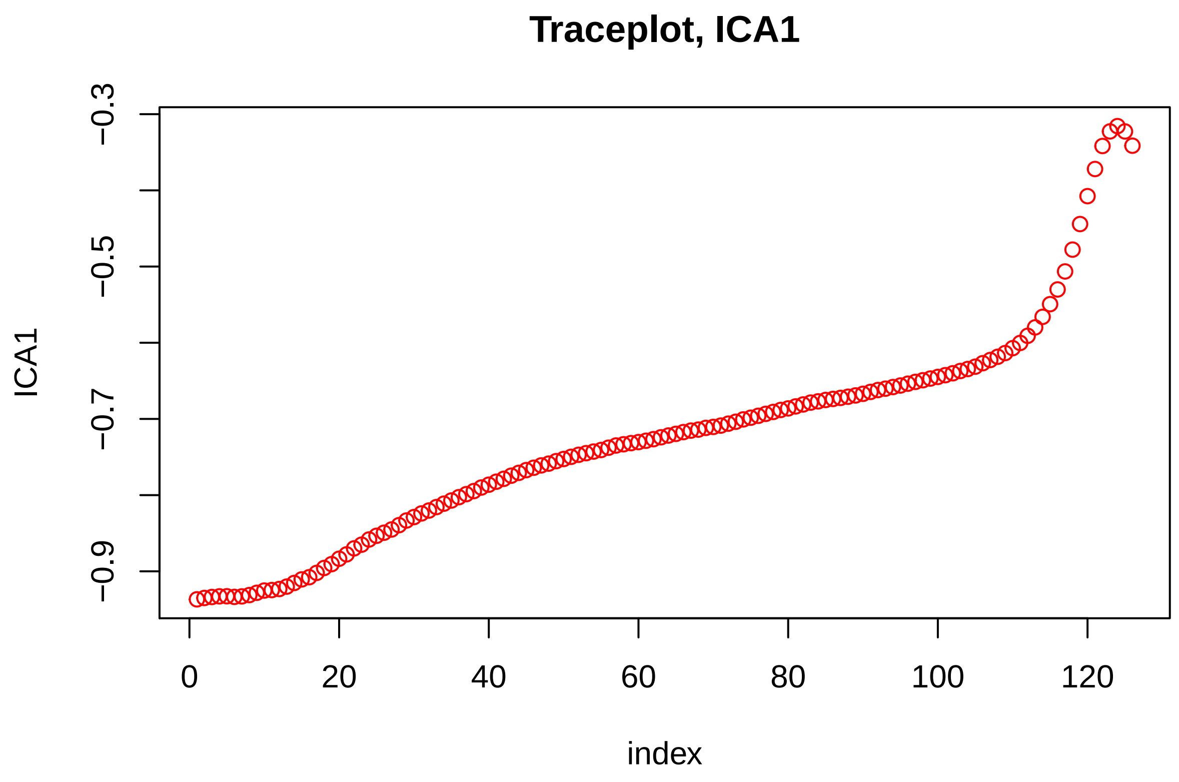
**Task 3**

```
#3: Perform independent Component Analysis (ICA) with no of components selected in step1
#(set seed 12345). Check the documentation of R for fastICA method and do following:
# Compute W'=K*W and present columns of W' in form of the trace plots. Compare with trace
# plots in step 2 and make conclusions. What kind of measure is represented by the matrix W'.
# Make a plot of the scores of the first two latent features and compare it with the score
# plot from step 1.

#Install package fastICa
#install.packages("fastICA")
library("fastICA")

set.seed(12345)
icaModel = fastICA(data, n.comp=2, verbose=TRUE)
```

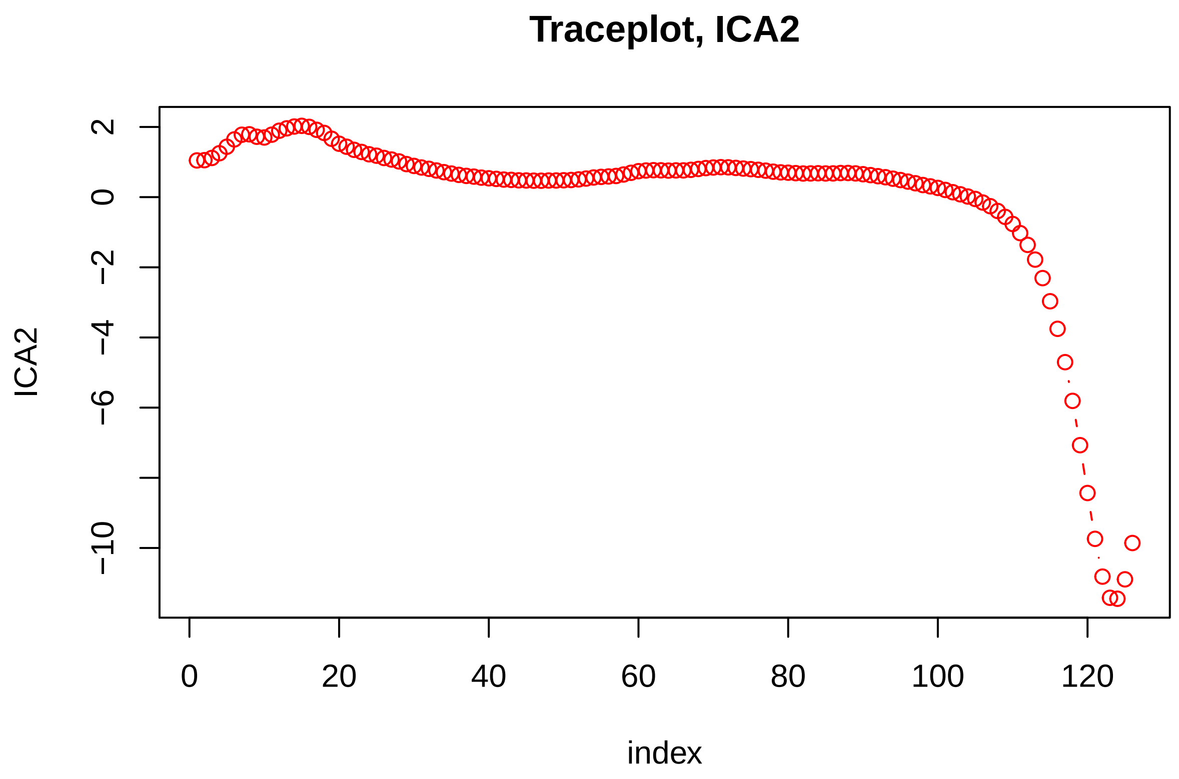
```
## Centering
```

```
## Whitening
## Symmetric FastICA using logcosh approx. to neg-entropy function

## Iteration 1 tol = 0.01930239

## Iteration 2 tol = 0.01303959

## Iteration 3 tol = 0.002393582

## Iteration 4 tol = 0.0006708454

## Iteration 5 tol = 0.0001661602

## Iteration 6 tol = 3.521604e-05
```

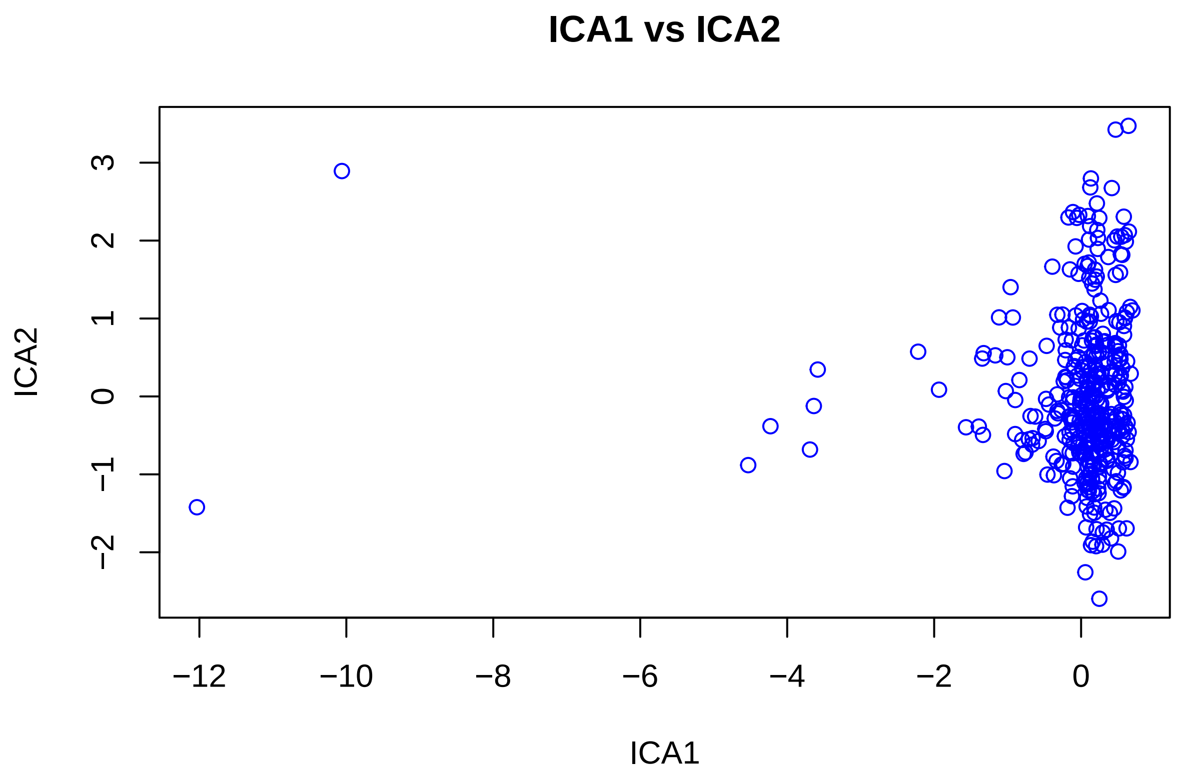
```
W=icaModel$W
K=icaModel$K
W_est=K%*%W
plot(W_est[,1], main="Traceplot, ICA1", xlab="index", ylab="ICA1", type="b", col="red")
```



```
plot(W_est[,2], main="Traceplot, ICA2", xlab="index", ylab="ICA2", type="b", col="red")
```



```
#Compared to the plots in step 2 the ICA1 follows in roughly the same pattern as PCA2
#and ICA2 the same as PCA1.
plot(icaModel$S[,1], icaModel$S[,2], main="ICA1 vs ICA2", xlab="ICA1", ylab="ICA2",
     type="p", col="blue")
```



```
#We can see from the plot that the dat is pretty well described by ICA2 whereas ICA1 is
#not that significant in describing the data (since it is close to 0 most of the cases).
#Some outliers are however described by ICA1.
```

Above we see the trace plots for the latent ICA components created from whitening the data by applying K (the pre-whitening matrix) onto W (our un-mixing matrix) to compute W'. This is done to “scale” the data. Compared with the trace plots in step 2 we see that the latent ICA component are mirrors of the PCA components. This is because PCA tries to find correlation between the features by maximizing the variance to allow reconstruction. ICA works maximizing independence through transforming the feature space into a new feature space where all features are mutually independent.

Just like in a) we see that also this is a mirror reflection of the scores from PCA but through performing ICA the features are scaled differently.

## Appendix

```
### Assignment 1

RNGversion('3.5.1')
Dataframe=read.csv("australian-crabs.csv")
n = length(Dataframe[,1])
CL = Dataframe$CL
RW = Dataframe$RW
plot(CL, RW, main="Plot of carapace length versus rear width depending on sex", sub="Red = Female, Blue
     col=c("red", "blue")[Dataframe$sex], xlab="CL", ylab="RW")

missclass=function(conf_matrix, fit_matrix){
  n=length(fit_matrix[,1])
  return(1-sum(diag(conf_matrix))/n)
}

library("MASS")
model = lda(sex ~ CL+RW, data=Dataframe)
predicted = predict(model, data=Dataframe)
confusion_matrix = table(Dataframe$sex, predicted$class)
misclass = missclass(confusion_matrix, Dataframe)
print(confusion_matrix)
print(misclass)
plot(CL, RW, main="Plot predicted values of CL and RW depending on sex", sub="Red = Female, Blue = Male
     col=c("red", "blue")[predicted$class], xlab="CL", ylab="RW")

model2 = lda(sex ~ CL+RW, data=Dataframe, prior=c(1,9)/10)
predicted2 = predict(model2, data=Dataframe)
confusion_matrix2 = table(Dataframe$sex, predicted2$class)
misclass2 = missclass(confusion_matrix2, Dataframe)
print(confusion_matrix2)
print(misclass2)
plot(CL, RW, main="Plot predicted values of CL and RW with priors p(Male)=0.9 and p(Female)=0.1"
     , sub="Red = Female, Blue = Male", col=c("red", "blue")[predicted2$class], xlab="CL", ylab="RW")

model3 = glm(sex ~ CL+RW, data=Dataframe, family='binomial')
predicted3 = predict(model3, newdata=Dataframe, type='response')
sexvector = c()
for (i in predicted3) {
  if (i>0.9) {
    sexvector = c(sexvector, 'Male')
  } else {
    sexvector = c(sexvector, 'Female')
  }
}
print(sexvector)
sexvector_factor = as.factor(sexvector)
confusion_matrix3 = table(Dataframe$sex, sexvector_factor)
misclass3 = missclass(confusion_matrix3, Dataframe)
print(confusion_matrix3)
print(misclass3)
plot(CL, RW, main="Plot predicted values of CL and RW but with logistic regression",
     col=c("red", "blue")[sexvector_factor], xlab="CL", ylab="RW", xlim=c(0,50), ylim=c(0,20))
```

```r
boundaryline = function(length, coefficientvector, prior) {
  return(-coefficientvector[1]/coefficientvector[3]-(coefficientvector[2]/coefficientvector[3])*length+
}
par(new=TRUE)
curve(boundaryline(x, model3$coefficients, 0.9), xlab="CL", ylab="RW", col="green", from=0, to=50, xlim=
      sub="Red = Female, Blue = Male, Green = Boundaryline")

### Assignment 2

library("tree")
RNGversion('3.5.1')

data=read.csv2("creditscoring.csv")
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,]

misclass=function(conf_matrix, fit_matrix){
  n=length(fit_matrix[,1])
  return(1-sum(diag(conf_matrix))/n)
}

fit_deviance=tree(good_bad~., data=train, split="deviance")
predicted_deviance=predict(fit_deviance, newdata=test, type="class")
confusionmatrix_deviance=table(test$good_bad, predicted_deviance)
misclass_deviance=misclass(confusionmatrix_deviance, test)
print(confusionmatrix_deviance)
print(misclass_deviance)
fit_gini=tree(good_bad~., data=train, split="gini")
predicted_gini=predict(fit_gini, newdata=test, type="class")
confusionmatrix_gini=table(test$good_bad, predicted_gini)
misclass_gini=misclass(confusionmatrix_gini, test)
print(confusionmatrix_gini)
print(misclass_gini)

fit_optimaltree=tree(good_bad~., data=train, split="deviance")
summary(fit_optimaltree)
trainScore=rep(0,15)
testScore=rep(0,15)
for(i in 2:15){
  prunedTree=prune.tree(fit_optimaltree, best=i)
  pred=predict(prunedTree, newdata=valid, type="tree")
  #Divide by two since double of data points
  trainScore[i]=deviance(prunedTree)/2
  testScore[i]=deviance(pred)
}
```

```r
plot(2:15, trainScore[2:15], type="b", col="red", ylim=c(200,500))
points(2:15, testScore[2:15], type="b", col="blue")
min_deviance=min(testScore[2:15])
print(min_deviance)
optimal_leaves=which(testScore[1:15] == min_deviance)
print(optimal_leaves)
finalTree=prune.tree(fit_optimaltree, best=4)
summary(finalTree)
plot(finalTree)
text(finalTree, pretty=0)
predicted_test=predict(finalTree, newdata=test, type="class")
confusionmatrix_test=table(test$good_bad, predicted_test)
misclass_test=misclass(confusionmatrix_test, test)
print(confusionmatrix_test)
print(misclass_test)

library(MASS)
library(e1071)
fit_naive=naiveBayes(good_bad~., data=train)
compute_naive=function(model,data){
  predictedNaive=predict(model, newdata=data, type="class")
  confusionmatrixNaive=table(data$good_bad,predictedNaive)
  misclass = misclass(confusionmatrixNaive, data)
  print(confusionmatrixNaive)
  print(misclass)
  return(predictedNaive)
}
predictedNaive_train=compute_naive(fit_naive,train)
predictedNaive_test=compute_naive(fit_naive, test)

class=function(data, class1, class2, prior){
  vector=c()
  for(i in data) {
    if(i>prior){
      vector=c(vector,class1)
    } else {
      vector=c(vector,class2)
    }
  }
  return(vector)
}

x_vector=seq(0.05,0.95,0.05)
tpr_tree=c()
fpr_tree=c()
tpr_naive=c()
fpr_naive=c()
treeVector=c()
treeConfusion = c()
naiveConfusion = c()
treeClass = c()
naiveClass = c()
set.seed(12345)
```

```r
predictTree=data.frame(predict(finalTree, newdata=test, type="vector"))
predictNaive=data.frame(predict(fit_naive, newdata=test, type="raw"))
for(prior in x_vector){
  treeClass = class(predictTree$good, 'good', 'bad', prior)
  treeConfusion=table(test$good_bad, treeClass)
  if(ncol(treeConfusion)==1){
    if(colnames(treeConfusion)=="good"){
      treeConfusion=cbind(c(0,0), treeConfusion)
    } else {
      treeConfusion=cbind(treeConfusion,c(0,0))
    }
  }
  totGood=sum(treeConfusion[2,])
  totBad=sum(treeConfusion[1,])
  tpr_tree=c(tpr_tree, treeConfusion[2,2]/totGood)
  fpr_tree=c(fpr_tree, treeConfusion[1,2]/totBad)
  print(fpr_tree)
  naiveClass=class(predictNaive$good, 'good', 'bad', prior)
  naiveConfusion=table(test$good_bad, naiveClass)
  if(ncol(naiveConfusion)==1){
    if(colnames(naiveConfusion)=="good"){
      naiveConfusion=cbind(c(0,0), naiveConfusion)
    } else {
      naiveConfusion=cbind(naiveConfusion,c(0,0))
    }
  }
  totGood=sum(naiveConfusion[2,])
  totBad=sum(naiveConfusion[1,])
  tpr_naive=c(tpr_naive, naiveConfusion[2,2]/totGood)
  fpr_naive=c(fpr_naive, naiveConfusion[1,2]/totBad)
}
plot(fpr_naive, tpr_naive, main="ROC curve", sub="Red = Naive Bayes, Blue = Tree", type="l", col="red",
     ylim=c(0,1), xlab="FPR", ylab="TPR")
points(fpr_tree, tpr_tree, type="l", col="blue")

naiveModel=naiveBayes(good_bad~., data=train)
train_loss=predict(naiveModel, newdata=train, type="raw")
test_loss=predict(naiveModel, newdata=test, type="raw")
confusion_trainLoss=table(train$good_bad, ifelse(train_loss[,2]/train_loss[,1]>10, "good", "bad"))
misclass_trainLoss=misclass(confusion_trainLoss, train)
print(confusion_trainLoss)
print(misclass_trainLoss)
confusion_testLoss=table(test$good_bad, ifelse(test_loss[,2]/test_loss[,1]>10, "good", "bad"))
misclass_testLoss=misclass(confusion_testLoss, test)
print(confusion_testLoss)
print(misclass_testLoss)

### Assignment 4

RNGversion('3.5.1')

data=read.csv2("NIRspectra.csv")
data$Viscosity=c()
```

```
n=dim(data)[1]

pcaAnalysis=prcomp(data)
lambda=pcaAnalysis$sdev^2
print(lambda)
propVar= lambda/sum(lambda)*100
screeplot(pcaAnalysis)
print(propVar)
noOfVars=1
sumOfVariation=propVar[noOfVars]
while(sumOfVariation<99){
  noOfVars=noOfVars+1
  sumOfVariation=sumOfVariation+propVar[noOfVars]
}
print(noOfVars)
plot(pcaAnalysis$x[,1],pcaAnalysis$x[,2], ylim=c(-10,10), type="p", col="blue", main="PC1 vs PC2", xlab

U=pcaAnalysis$rotation
plot(U[,1], main="Traceplot, PC1", xlab="index", ylab="PC1", type="b")
plot(U[,2], main="Traceplot, PC2", xlab="index", ylab="PC2", type="b")

library("fastICA")

set.seed(12345)
icaModel = fastICA(data, n.comp=2, verbose=TRUE)
W=icaModel$W
K=icaModel$K
W_est=K%*%W
plot(W_est[,1], main="Traceplot, ICA1", xlab="index", ylab="ICA1", type="b", col="red")
plot(W_est[,2], main="Traceplot, ICA2", xlab="index", ylab="ICA2", type="b", col="red")
plot(icaModel$S[,1], icaModel$S[,2], main="ICA1 vs ICA2", xlab="ICA1", ylab="ICA2", type="p", col="blue
```