Lab 2

2018-12-10

Contents

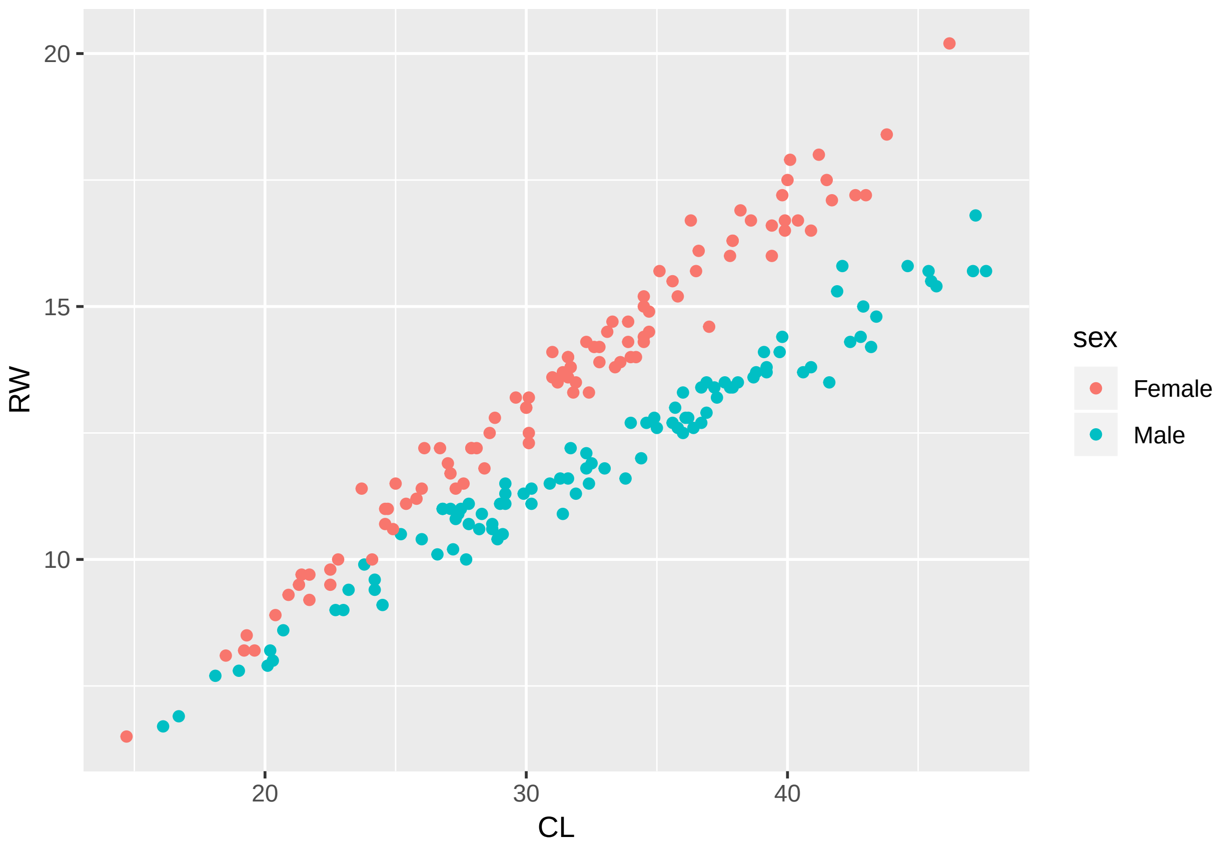
List of assignment contributions:

- Assignment 1: Erik Tedhamre
- Assignment 2: Rolf Sievert
- Assignment 3: Tobias Fjellström

Assignment 1

Assignment 1.1

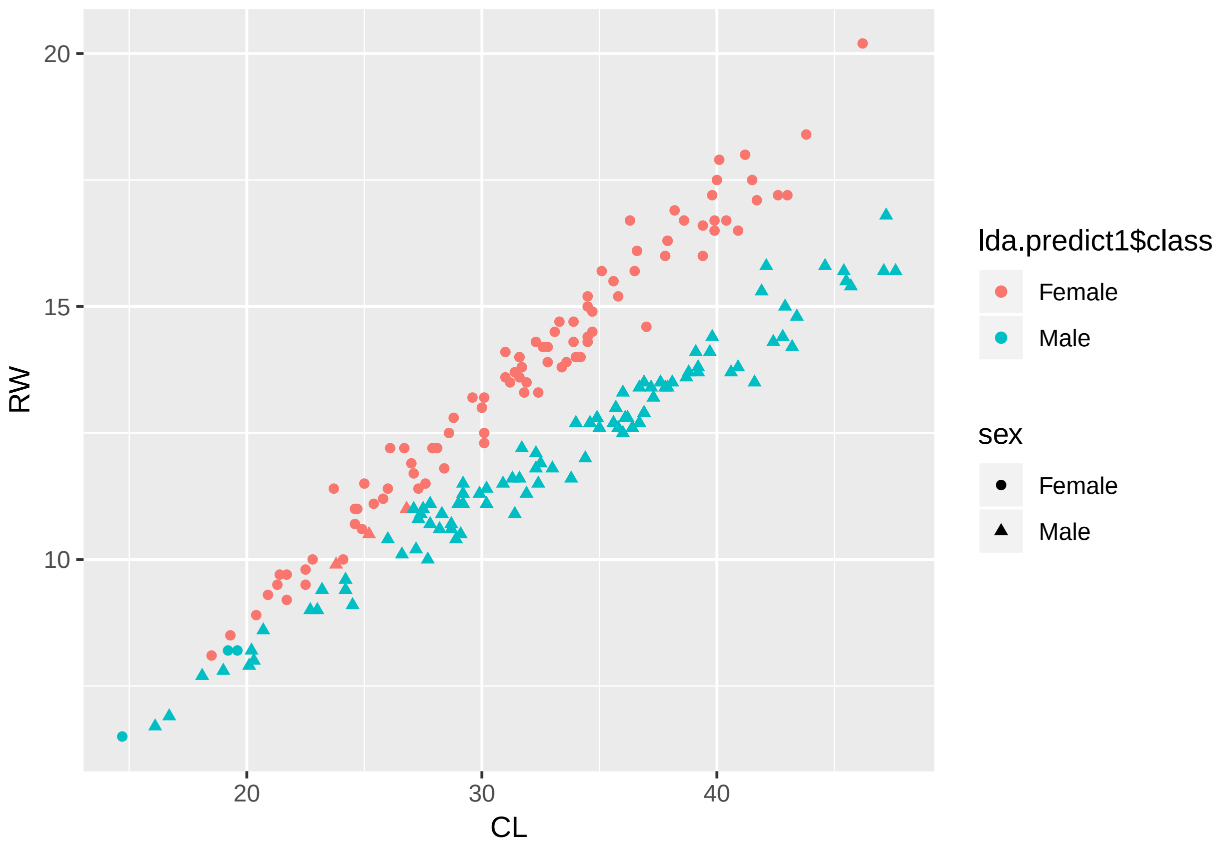
```r
library("ggplot2")
library("MASS")
data <- data.frame(read.csv("australian-crabs.csv"))
ggplot(data = data, mapping = aes(CL, RW, color = sex)) + geom_point()
```



A plot of carapace length versus rear width where the observations are colored by sex. Looking at the graph the data seems reasonably easy to classify by linear discriminant analysis. Because there seems to be a line between the two sexes. There's also a linear relationship betweem the two variables.

Assignment 1.2

```
lda.model1 <- lda(sex ~ CL + RW, data = data)
lda.predict1 <- predict(lda.model1, data)
ggplot.0.5 <- ggplot(data = data, mapping = aes(CL, RW, color = lda.predict1$class, shape = sex )) + ge
ggplot.0.5
```

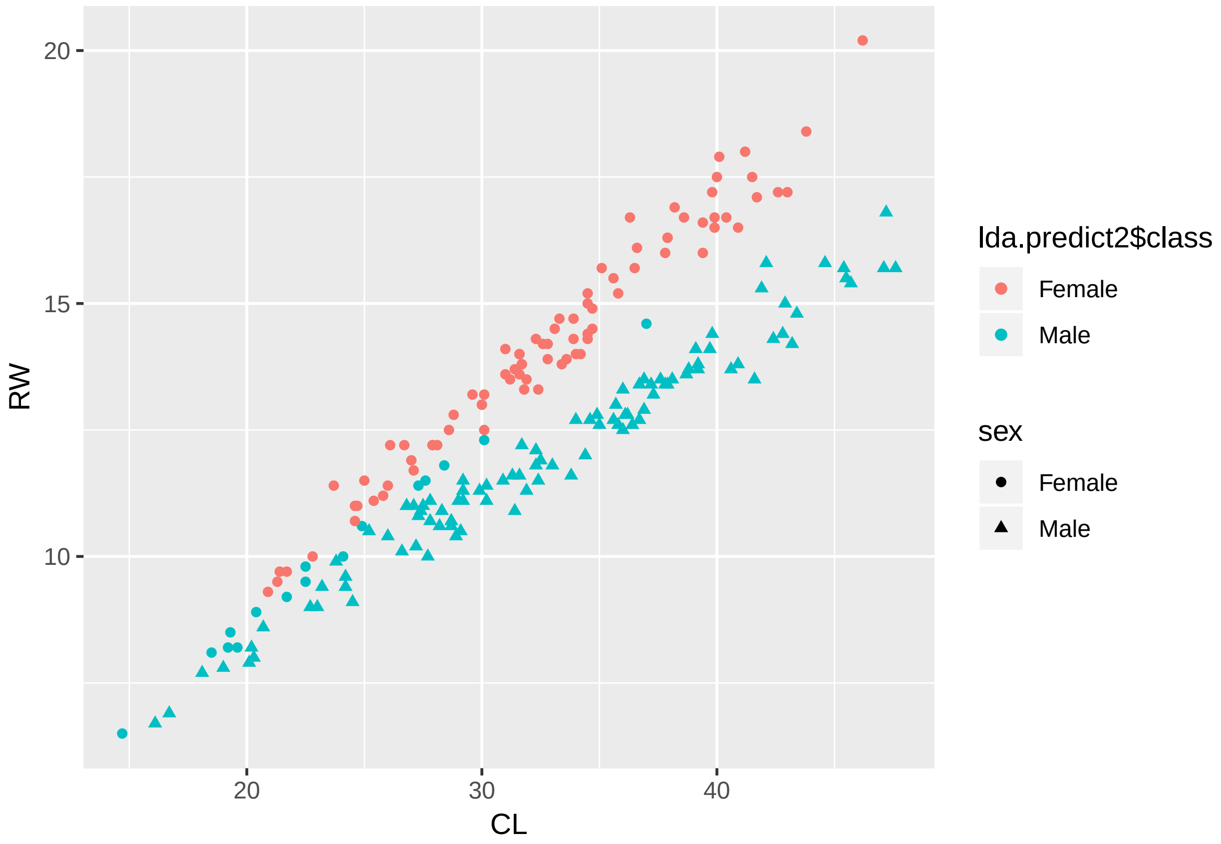


```
mcr.0.5 <- mean(lda.predict1$class != data$sex)
```

The missclassification rate for the linear discriminant analysis is **0.035**. This is pretty reasonable considering we saw on the original graph that there was one area with a bit of an overlap. If this is good enough for actual use is hard to say, it mostly depends on how much we would lose on an incorrect classification.

Assignment 1.3

```
lda.model2 <- lda(sex ~ CL + RW, data = data, prior = c(0.1, 0.9))
lda.predict2 <- predict(lda.model2, data)
ggplot(data = data, mapping = aes(CL, RW, color = lda.predict2$class, shape = sex )) + geom_point()
```



The number of males increased since we are assuming a wheighted distribution. Especially the areas containing both types of observations are now classified as only males instead of both.

```
mcr.0.9 <- mean(lda.predict2$class != data$sex)
```

The missclassification rate for the weighted linear discriminant analysis is **0.08**.

Assignment 1.4

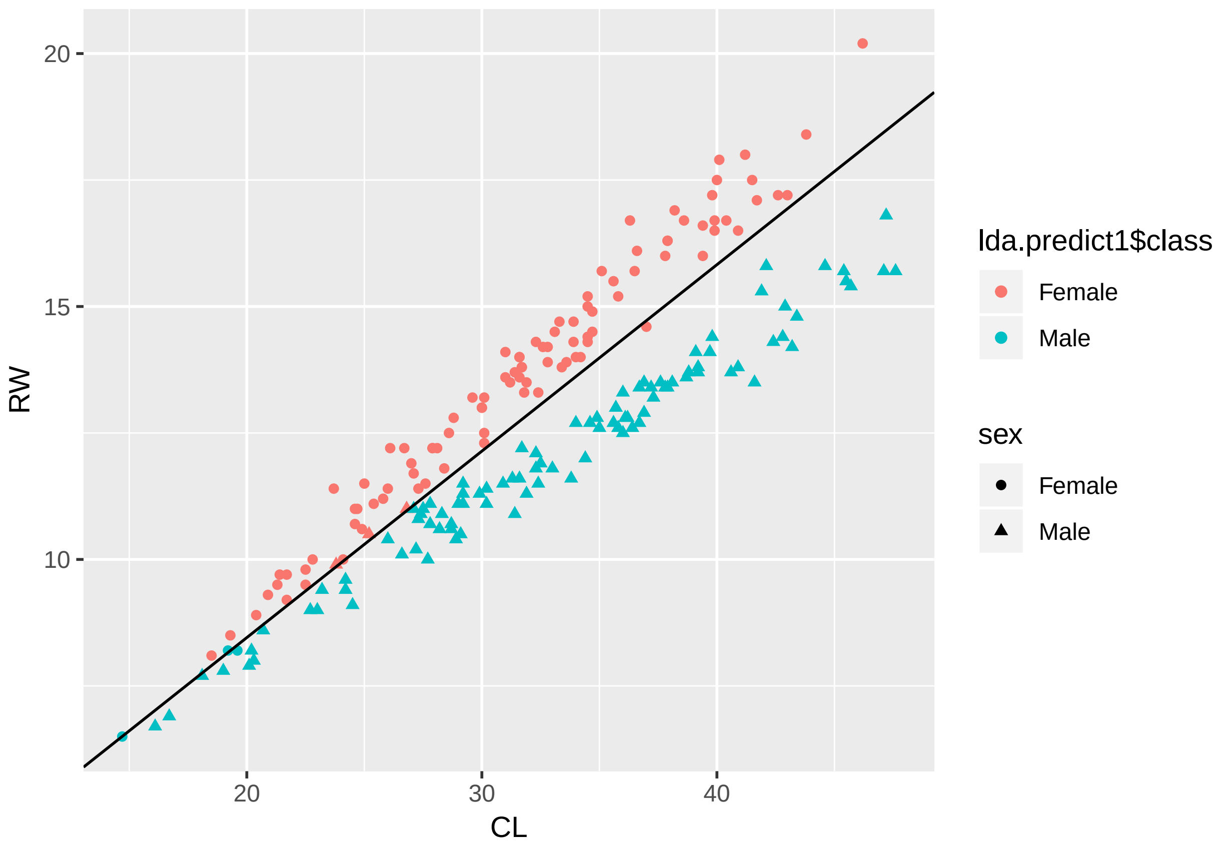
```
glm.model <- glm(as.factor(sex) ~ CL + RW, family = binomial, data = data)
```

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

```
glm.predict <- predict(glm.model, data, type = 'response')
mcr.glm <- mean(glm.predict != data$sex)
```

The missclassification rate is **0.035** which is the same as the original linear discriminant analysis.

```
glm.predict.0.5 <- ifelse(glm.predict > 0.5, "Male", "Female")
glm.slope <- coef(glm.model)[2]/(-coef(glm.model)[3])
glm.intercept <- coef(glm.model)[1]/(-coef(glm.model)[3])
ggplot.0.5 + geom_abline(slope = glm.slope, intercept = glm.intercept)
```



The decision line is drawn in the graph. It's described by **RW = 0.6CL + 1.08379**.

Assignment 2

Step 1

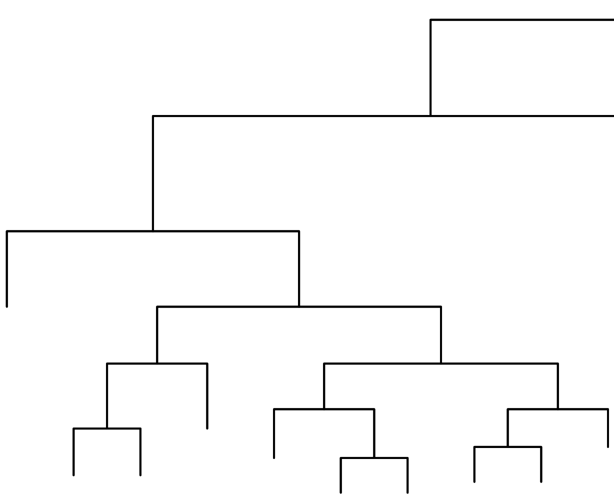
Firstly, the data was separated into training, validation and test data.

```r
# Add libraries
library("gridExtra")
library("ggplot2")
library("tree")
library("MASS")
library("e1071")

# Set working directory
setwd("~/courses/tdde01/lab2")

# Read data
scores = read.csv2("creditscoring.csv")
# Read as strings
scores$good_bad = as.factor(scores$good_bad)

# Split data into train/val/test
n=dim(scores)[1]
set.seed(12345)
```

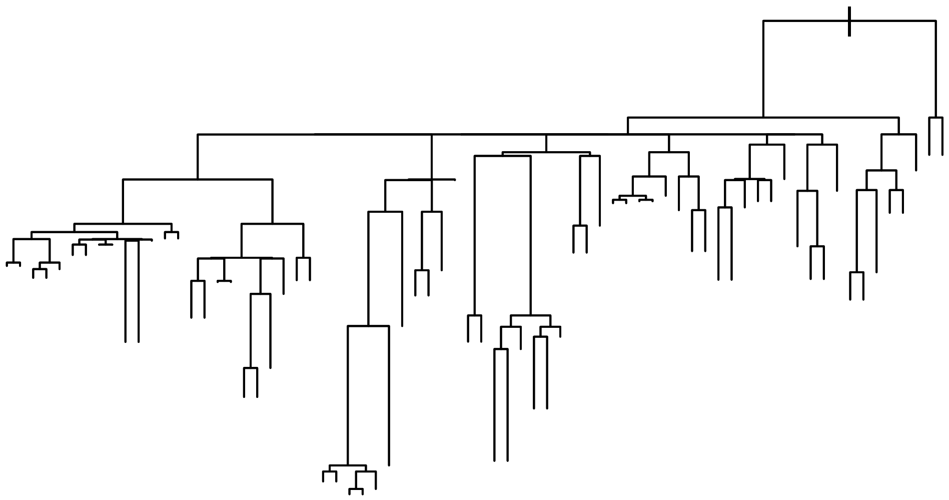
```r
id=sample(1:n, floor(n*0.5))
train=scores[id,]
id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.25))
val=scores[id2,]
id3=setdiff(id1,id2)
test=scores[id3,]
```

Step 2

```r
# Create tree
tree.deviance = tree(formula = good_bad~.,
                     data = train,
                     split="deviance")
tree.gini = tree(formula = good_bad~.,
                     data = train,
                     split="gini")
# Plot tree
plot(tree.deviance)
```



```r
plot(tree.gini)
```



```r
# Make predictions
predict.deviance.test = predict(tree.deviance, test, type = "class")
predict.deviance.train = predict(tree.deviance, train, type = "class")
predict.gini.test = predict(tree.gini, test, type = "class")
predict.gini.train = predict(tree.gini, train, type = "class")

# Calculate misclassification rates
misclass = function(predicted, true) {
    return(mean(predicted != true))
}

deviance.test.misclass = misclass(predict.deviance.test, test$good_bad)
deviance.train.misclass = misclass(predict.deviance.train, train$good_bad)
gini.test.misclass = misclass(predict.gini.test, test$good_bad)
gini.train.misclass = misclass(predict.gini.train, train$good_bad)

# Print misclassification rates
cat("Misclassification on deviance with test: ",
    deviance.test.misclass)
```

```
## Misclassification on deviance with test:  0.268
```

```r
cat("\nMisclassification on deviance with train: ",
    deviance.train.misclass)
```

```
##
## Misclassification on deviance with train:  0.212
```

```r
cat("\nMisclassification on gini with test: ",
    gini.test.misclass)
```

```
##
## Misclassification on gini with test:  0.368
```

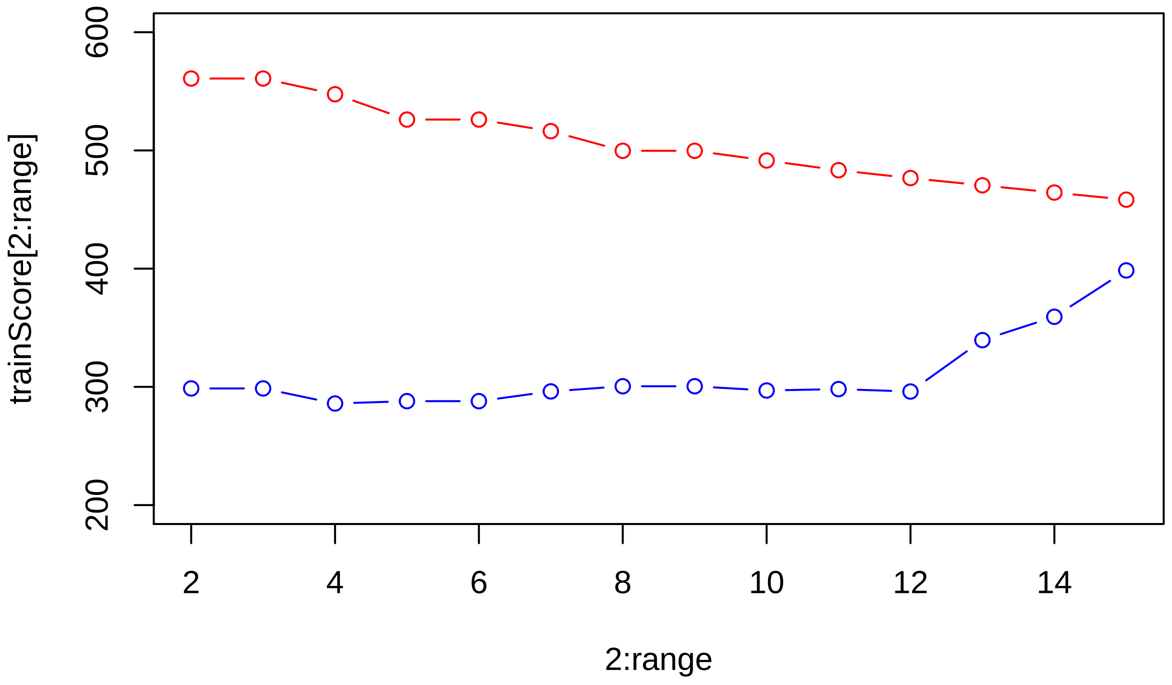
```r
cat("\nMisclassification on gini with train: ",
    gini.train.misclass, "\n")
```

```
##
## Misclassification on gini with train:  0.24
```

Here, misclassification rate is lower when using deviance as measure of impurity.

Step 3

```r
# Lists of scores
range = 15
pruned=rep(0, range)
trainScore=rep(0, range)
testScore=rep(0, range)

test.tree = tree(formula=good_bad~., data=train)
for(i in 2:range) {
    # Prune the tree
    prunedTree=prune.tree(test.tree, best=i)
    # Make trediction on validation data
    pred=predict(prunedTree, newdata=val, type="tree")
    # Append scores
    trainScore[i]=deviance(prunedTree)
    testScore[i]=deviance(pred)
}

# Plot the scores
plot(2:range, trainScore[2:range], type="b", col="red", ylim=c(200, 600))
points(2:range, testScore[2:range], type="b", col="blue")
```



```r
optLeaves = 4

# Info on optimal tree
optTree = prune.tree(tree.deviance, best=optLeaves)
summary(optTree)
```

```
##
## Classification tree:
## snip.tree(tree = tree.deviance, 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
```

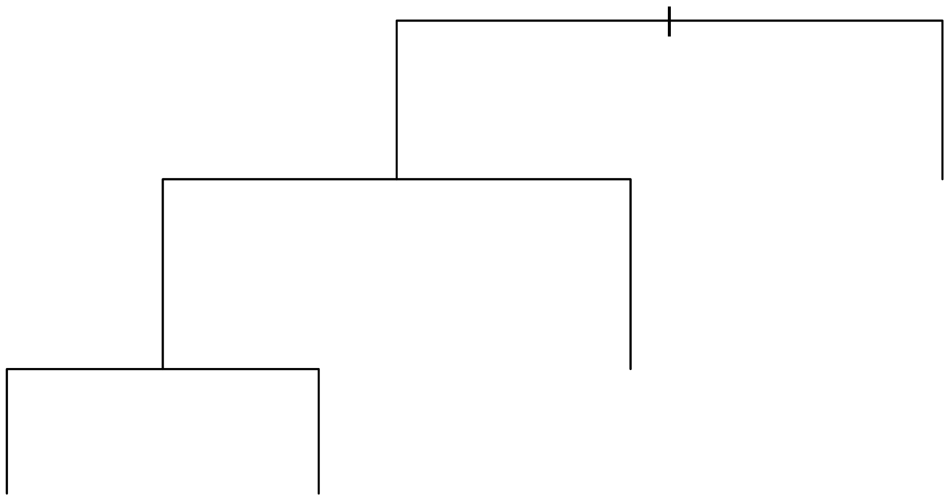
```r
optTree.predict = predict(optTree, newdata=test)
optTree.predict.string = ifelse(optTree.predict[2] > 0.5, "good", "bad")
optTree.misclass = misclass(optTree.predict.string, test$good_bad)
cat("\nMisclassification on optimal tree: ",
    optTree.misclass)
```

```
##
## Misclassification on optimal tree:  0.696
```

```r
cat("\nTree depth is 5 as can be seen in the plot. \n")
```

```
##
## Tree depth is 5 as can be seen in the plot.
```

```r
plot(optTree)
```



```r
cat("\n\nUsed variables in optimal tree: \n'savings' 'duration' 'history' 'age' 'purpose'\n\n")
```

```
##
##
## Used variables in optimal tree:
## 'savings' 'duration' 'history' 'age' 'purpose'
```

Step 4

```r
# Create model and classify
model.bayes = naiveBayes(formula = good_bad~., data=train)
predict.bayes.test = predict(model.bayes, newdata=test, type="class")
predict.bayes.train = predict(model.bayes, newdata=train, type="class")

# Print info on classification with prediction on test
table.bayes = table(predict.bayes.test, test$good_bad)
cat("Confusion table of naïve bayes:")
```

```
## Confusion table of naïve bayes:
```

```
print(table.bayes)
```

```
##
## predict.bayes.test bad good
##               bad   46   49
##               good  30  125
```

```
misclass.bayes = mean(predict.bayes.test != test$good_bad)
cat("Misclassification with naïve bayes: ", misclass.bayes, "\n\n")
```

```
## Misclassification with naïve bayes:  0.316
```

```
# Print info on classification with prediction on test
table.bayes = table(predict.bayes.train, train$good_bad)
cat("Confusion table of naïve bayes:")
```

```
## Confusion table of naïve bayes:
```

```
print(table.bayes)
```

```
##
## predict.bayes.train bad good
##                bad   95   98
##                good  52  255
```

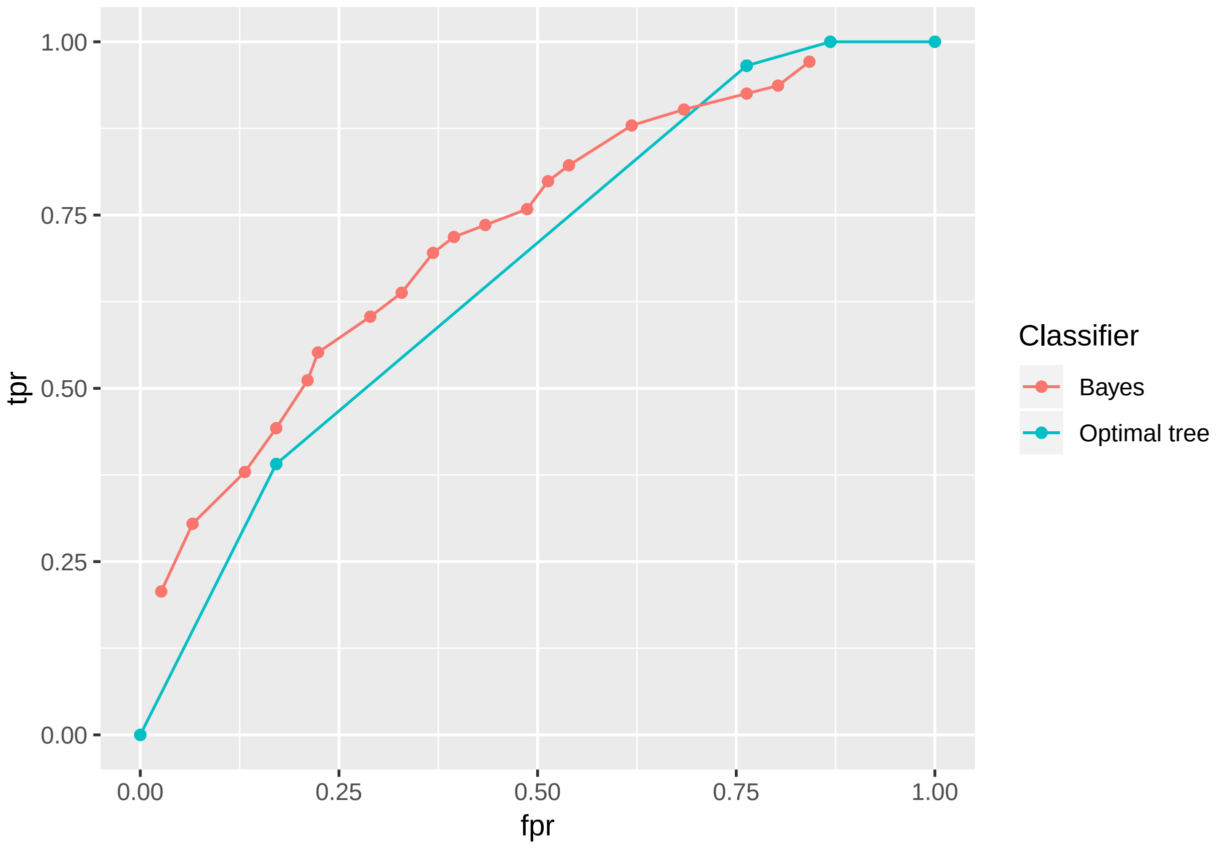
```
misclass.bayes = mean(predict.bayes.train != train$good_bad)
cat("Misclassification with naïve bayes: ", misclass.bayes, "\n\n")
```

```
## Misclassification with naïve bayes:  0.3
```

Naïve Bayes has much better result than in step 3.

Step 5

```
# Calculate TPR and FPR of optimal tree and bayes
getROC = function(pred, pi) {
    tpr = c()
    fpr = c()
    for (p in pi) {
        # Change probabilities to strings
        tmp = ifelse(pred[,'good'] > p, "good", "bad")
        # Get confusion matrix
        cm = table(predicted=tmp, actual=test$good_bad)
        if('good' %in% rownames(cm)) {
          # Calculate TPR, first dim of cm is predicted
          t = cm['good', 'good'] / sum(cm[,'good'])
          # Calculate FPR
          f = cm['good', 'bad'] / sum(cm[, 'bad'])
          # Append to list of values
          tpr = c(tpr, ifelse(is.finite(t), t, 0))
          fpr = c(fpr, ifelse(is.finite(f), f, 0))
        } else {
          tpr = c(tpr, 0)
          fpr = c(fpr, 0)
        }
    }
    df = data.frame(tpr, fpr)
    return(df)
}

pi = seq(0.05, 0.95, 0.05)
pred.optTree = predict(optTree, newdata=test)
pred.bayes = predict(model.bayes, newdata=test, type='raw')
opt.roc = getROC(pred.optTree, pi)
bayes.roc = getROC(pred.bayes, pi)

#Plot
roc.plot = ggplot(mapping=aes(col=Classifier)) +
    geom_point(data=opt.roc, aes(x=fpr, y=tpr, col="Optimal tree")) +
    geom_line(data=opt.roc, aes(x=fpr, y=tpr, col="Optimal tree")) +
    geom_point(data=bayes.roc, aes(x=fpr, y=tpr, col="Bayes")) +
    geom_line(data=bayes.roc, aes(x=fpr, y=tpr, col="Bayes"))

grid.arrange(roc.plot)
```



Naïve Bayes has better ratio between TPR and FPR. The only exception is around $\pi = 0.75$, as can be seen in the graph.

Step 6

Using loss matrix with naïve bayes.

```
# Create model and classify
model.bayes = naiveBayes(formula = good_bad~., data=train)
predict.bayes.test = predict(model.bayes, newdata=test, type="raw")
predict.bayes.train = predict(model.bayes, newdata=train, type="raw")

# Print info on classification with prediction on test
loss = 10/1
predict.bayes.test.loss = ifelse(predict.bayes.test[, 'good'] /
                                 predict.bayes.test[, 'bad'] > loss,
                                 'good',
                                 'bad')
table.bayes = table(Predict=predict.bayes.test.loss, Actual=test$good_bad)
cat("Confusion table of naïve bayes (using test data):")
```

```
## Confusion table of naïve bayes (using test data):
```

```
print(table.bayes)
```

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

```
misclass.bayes = mean(predict.bayes.test.loss != test$good_bad)
cat("Misclassification with naïve bayes (using test data): ", misclass.bayes, "\n")
```

```
## Misclassification with naïve bayes (using test data):  0.508
```

```
# Print info on classification with prediction on train
predict.bayes.train.loss = ifelse(predict.bayes.train[, 'good'] /
                                  predict.bayes.train[, 'bad'] > loss,
                                 'good',
                                 'bad')
table.bayes = table(Predict=predict.bayes.train.loss, Actual=train$good_bad)
cat("Confusion table of naïve bayes (using train data): ")
```

```
## Confusion table of naïve bayes (using train data):
```

```
print(table.bayes)
```

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

```
misclass.bayes = mean(predict.bayes.train.loss != train$good_bad)
cat("Misclassification with naïve bayes (using train data): ", misclass.bayes, "\n")
```

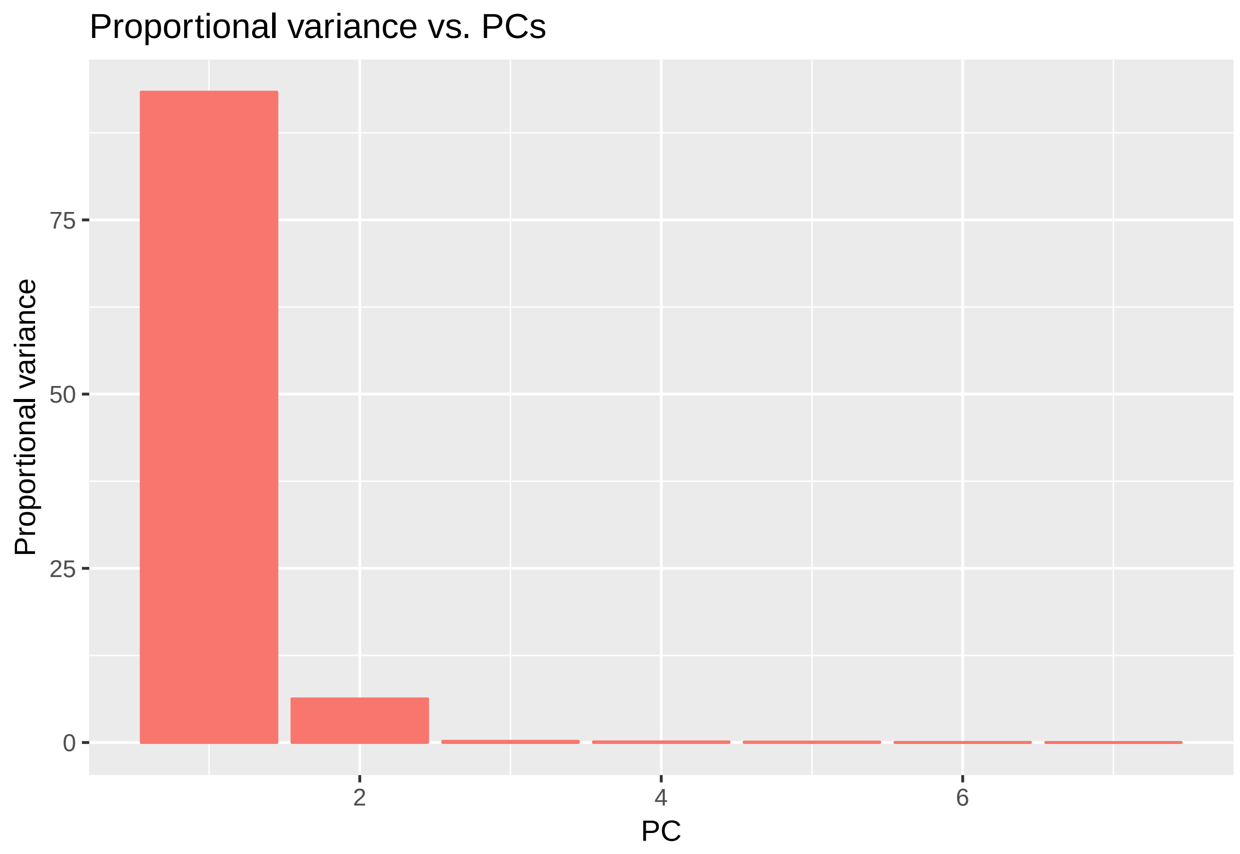
```
## Misclassification with naïve bayes (using train data):  0.546
```

The misclassification is much greater. But the confusion matrix is more favorable from an economic point of view for a company since less are predicted good that are actually bad.

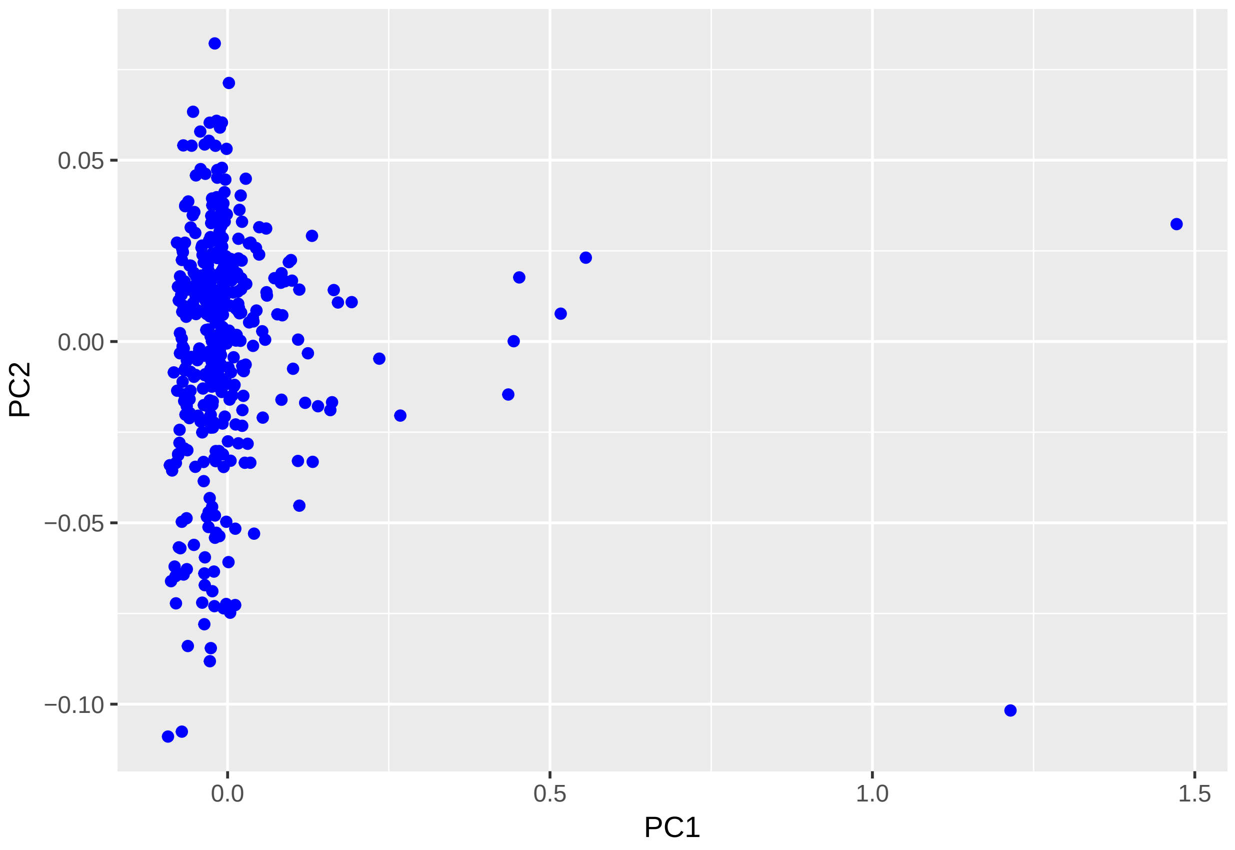
Assignment 4: Principal components

The assignment was to use near-infrared spectrometry data to predict the viscosity of diesel fuels.

4-1

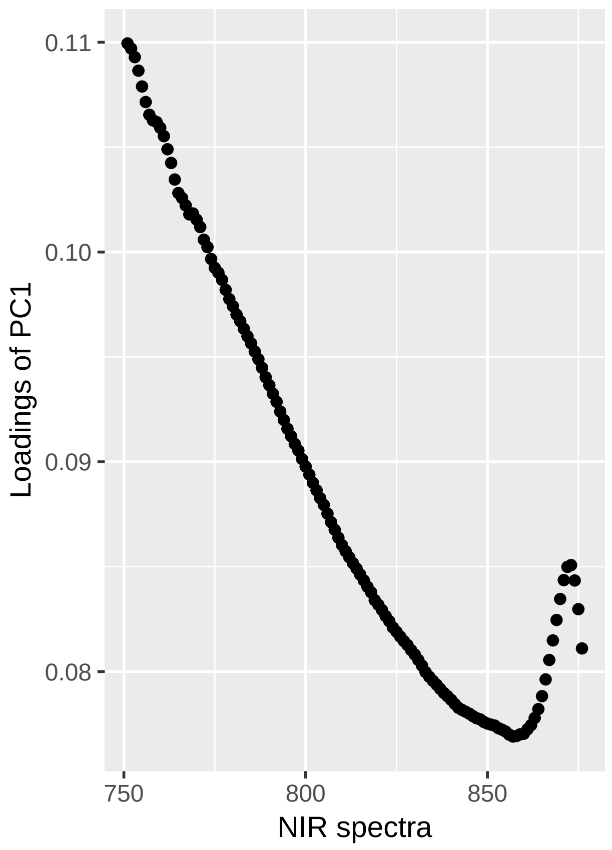


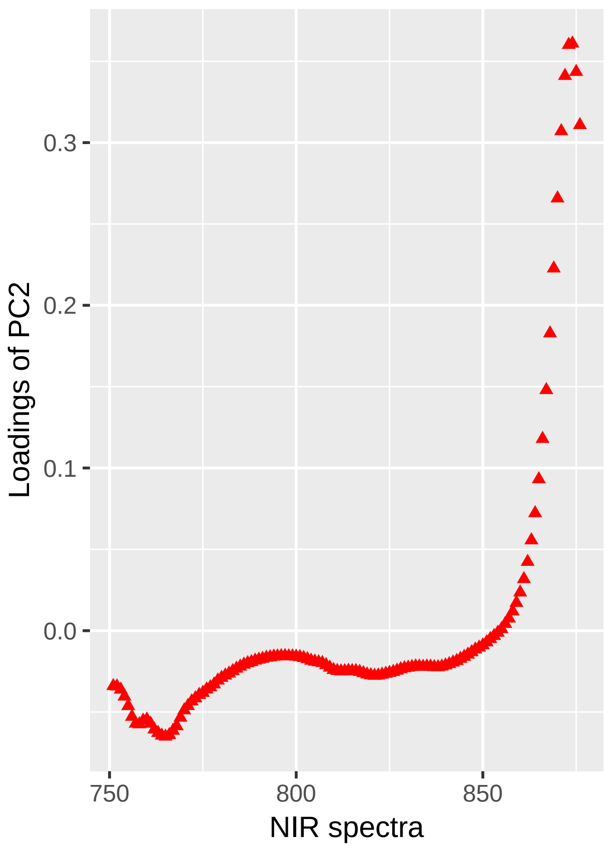
The figure above shows the proportion of the variance that is explained by the first seven principal components (PCs). 99.5957012% of the variation is explained by only PC1 and PC2, and they are selected for the upcoming tasks because very little information will be added by choosing a more complex model involving additional PCs. The data is shown in the space of PC1 and PC2 below. Some of the fuels seem unusual since there are some very dramatic outliers along PC1. However, maybe this is one of those fraction of percentages variations that we omit by only choosing PC1 and PC2 to describe the features.



4-2

Traceplots showing the loadings of PC1 (black) and PC2 (red triangles), i.e. the amount PC1 and PC2 depends on each of the 127 NIR channels are shown in the figures below. While PC1 seems to be a linear combination of small contributions from almost every channel PC2 is mostly explained by a handful of features in the region ~X810-X827.



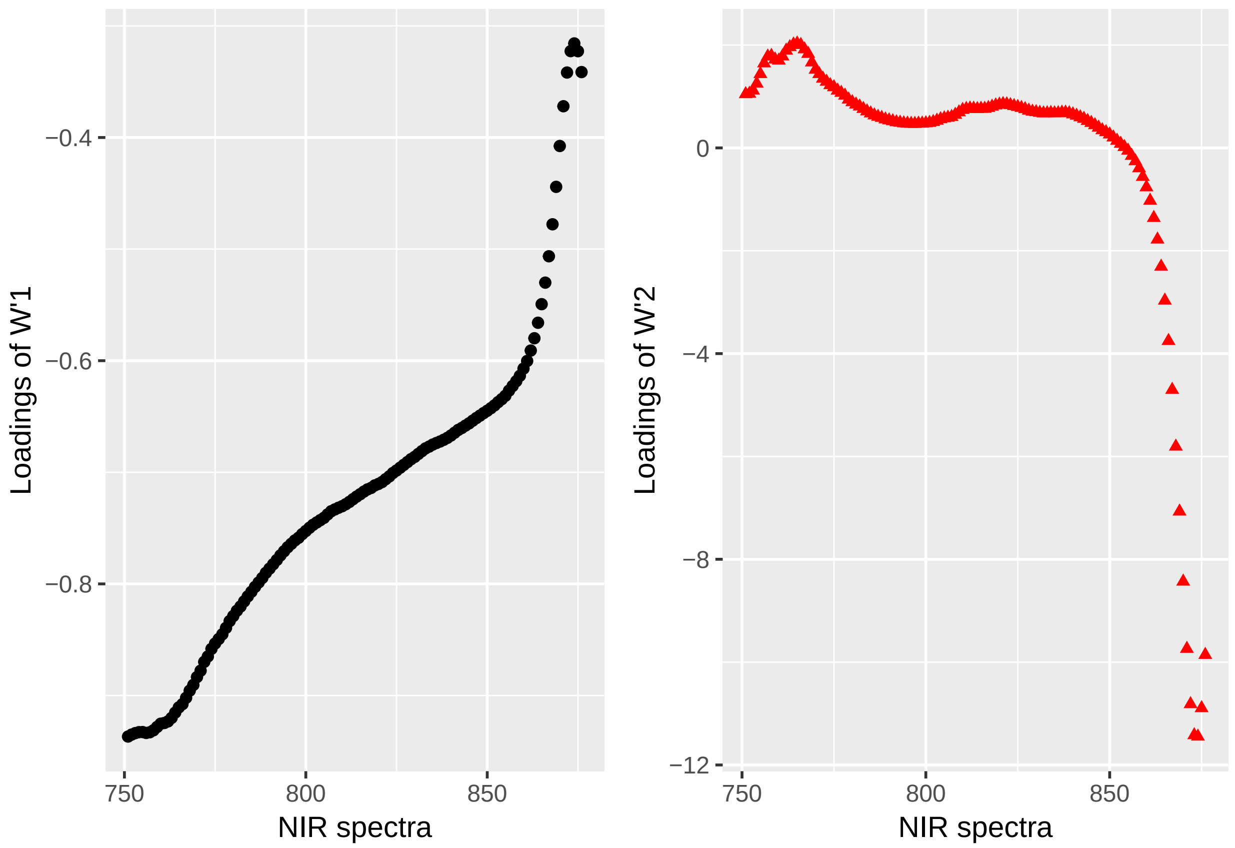


4-3

Independent component analysis (ICA) was performed to compare with the PCA results, after resetting the random number generator seed.

a)

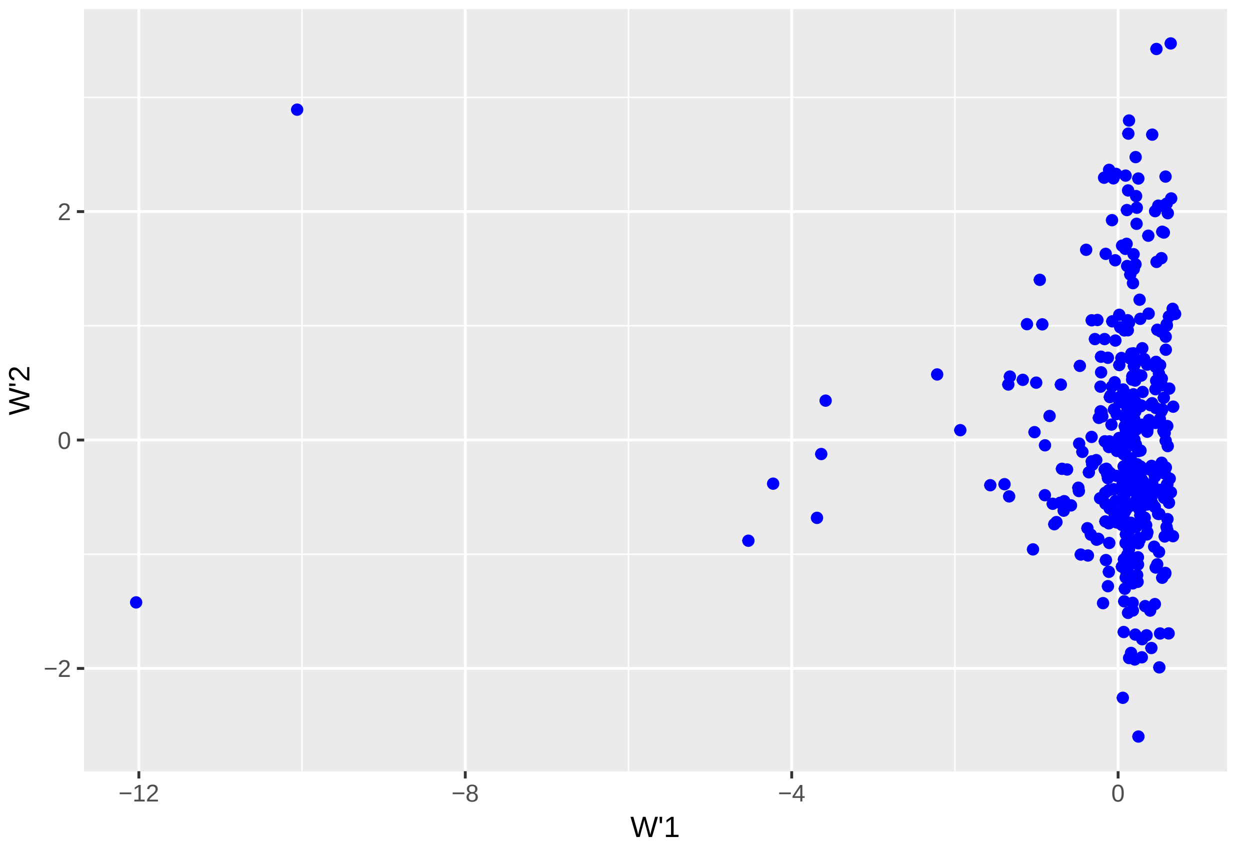
The loadings of the first two components of the ICA, computed as the first two columns of the matrix $W^{'} = KX$ (ouput vectors of the algorithm) are presented in the figure below:



The loadings carry some resemblance to the PC1 and PC2 components, but are actually not just inverses of the same if one looks closer. W_1' is totally unique and depends on mostly the lower end channels. W_2' is obviously an inverse of PC2 in the sense that it represents the negative mirror of every loading on PC2, however the amplitude of the loadings are also much higher than they were on PC2. The matrix W' thus represents another feature space produced from the input space than the PC algorithm generated, closely reminiscent but not quite the same. From theory we now this feature space is constructed to be as non-Gaussian as possible and to be rotation *variant*.

b)

The data plotted in the space of W_1' and W_2' is shown below:



We see clearly now that the data is plotted 'backwards and upside down' compared to the PC1-PC2 space. Conclusively both PCA and ICA extract the same features but by theory we know ICA identify the latent features uniquely, and apparently this manifests itself as a slightly different linear combination that explains the data a bit different but achieves the same amount of variance with its first two components.

Appendix

Assignment 1

```
cat ass1-kod-teddy.R
```

```
library("ggplot2")
library("MASS")
# 1.1
setwd("~/TDDE01/lab2")
data <- data.frame(read.csv("australian-crabs.csv"))
ggplot(data = data, mapping = aes(CL, RW, color = sex)) + geom_point()
# Yes, it seems to be two pretty distinct data sets, perhaps the beginning could be tricky
# 1.2
lda.model1 <- lda(sex ~ CL + RW, data = data)
lda.predict1 <- predict(lda.model1, data)
ggplot.0.5 <- ggplot(data = data, mapping = aes(CL, RW, color = lda.predict1$class, shape = sex )) + ge
mcr.0.5 <- mean(lda.predict1$class != data$sex)
# 1.3
lda.model2 <- lda(sex ~ CL + RW, data = data, prior = c(0.1, 0.9))
lda.predict2 <- predict(lda.model2, data)
```

```
ggplot.0.9 <- ggplot(data = data, mapping = aes(CL, RW, color = lda.predict2$class, shape = sex )) + geo
mcr.0.9 <- mean(lda.predict2$class != data$sex)
# It got worse since the distribution of the data is 50/50 and not 90/10
#1.4
glm.model <- glm(as.factor(sex) ~ CL + RW, family = binomial, data = data)
glm.predict <- predict(glm.model, data, type = 'response')
glm.predict.0.5 <- ifelse(glm.predict > 0.5, "Male", "Female")
mcr.glm <- mean(glm.predict.0.5 != data$sex)
summary(glm.model)
glm.slope <- coef(glm.model)[2]/(-coef(glm.model)[3])
glm.intercept <- coef(glm.model)[1]/(-coef(glm.model)[3])
ggplot.0.5 + geom_abline(slope = glm.slope, intercept = glm.intercept)
```

Assignment 2

```
cat ass2.R
```

```
# Step 1
# Add libraries
library("gridExtra")
library("ggplot2")
library("tree")
library("MASS")
library("e1071")

# Set working directory
setwd("~/courses/tdde01/lab2")

# Read data
scores = read.csv2("creditscoring.csv")
# Read as strings
scores$good_bad = as.factor(scores$good_bad)

# Split data into train/val/test
n=dim(scores)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=scores[id,]
id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.25))
val=scores[id2,]
id3=setdiff(id1,id2)
test=scores[id3,]

# Step 2
# Create tree
tree.deviance = tree(formula = good_bad~.,
                     data = train,
                     split="deviance")
tree.gini = tree(formula = good_bad~.,
                     data = train,
                     split="gini")
```

```
# Plot tree
plot(tree.deviance)
plot(tree.gini)

# Make predictions
predict.deviance.test = predict(tree.deviance, test, type = "class")
predict.deviance.train = predict(tree.deviance, train, type = "class")
predict.gini.test = predict(tree.gini, test, type = "class")
predict.gini.train = predict(tree.gini, train, type = "class")

# Calculate misclassification rates
misclass = function(predicted, true) {
    return(mean(predicted != true))
}

deviance.test.misclass = misclass(predict.deviance.test, test$good_bad)
deviance.train.misclass = misclass(predict.deviance.train, train$good_bad)
gini.test.misclass = misclass(predict.gini.test, test$good_bad)
gini.train.misclass = misclass(predict.gini.train, train$good_bad)

# Print misclassification rates
cat("Misclassification on deviance with test: ",
    deviance.test.misclass)
cat("\nMisclassification on deviance with train: ",
    deviance.train.misclass)
cat("\nMisclassification on gini with test: ",
    gini.test.misclass)
cat("\nMisclassification on gini with train: ",
    gini.train.misclass, "\n")

# Step 3
# Lists of scores
range = 15
pruned=rep(0, range)
trainScore=rep(0, range)
testScore=rep(0, range)

test.tree = tree(formula=good_bad~., data=train)
for(i in 2:range) {
    # Prune the tree
    prunedTree=prune.tree(test.tree, best=i)
    # Make trediction on validation data
    pred=predict(prunedTree, newdata=val, type="tree")
    # Append scores
    trainScore[i]=deviance(prunedTree)
    testScore[i]=deviance(pred)
}

# Plot the scores
plot(2:range, trainScore[2:range], type="b", col="red", ylim=c(200, 600))
points(2:range, testScore[2:range], type="b", col="blue")

optLeaves = 4
```

```
# Info on optimal tree
optTree = prune.tree(tree.deviance, best=optLeaves)
summary(optTree)
optTree.predict = predict(optTree, newdata=test)
optTree.predict.string = ifelse(optTree.predict[2] > 0.5, "good", "bad")
optTree.misclass = misclass(optTree.predict.string, test$good_bad)
cat("\nMisclassification on optimal tree: ",
    optTree.misclass)

cat("\nTree depth is 5 as can be seen in the plot. \n")
plot(optTree)
cat("\n\nUsed variables in optimal tree: \n'savings' 'duration' 'history' 'age' 'purpose'\n\n")

# Step 4
# Create model and classify
model.bayes = naiveBayes(formula = good_bad~., data=train)
predict.bayes.test = predict(model.bayes, newdata=test, type="class")
predict.bayes.train = predict(model.bayes, newdata=train, type="class")

# Print info on classification with prediction on test
table.bayes = table(predict.bayes.test, test$good_bad)
cat("Confusion table of naïve bayes:")
print(table.bayes)

misclass.bayes = mean(predict.bayes.test != test$good_bad)
cat("Misclassification with naïve bayes: ", misclass.bayes, "\n\n")

# Print info on classification with prediction on test
table.bayes = table(predict.bayes.train, train$good_bad)
cat("Confusion table of naïve bayes:")
print(table.bayes)

misclass.bayes = mean(predict.bayes.train != train$good_bad)
cat("Misclassification with naïve bayes: ", misclass.bayes, "\n\n")

# Step 5
# Calculate TPR and FPR of optimal tree and bayes
getROC = function(pred, pi) {
    tpr = c()
    fpr = c()
    for (p in pi) {
        # Change probabilities to strings
        tmp = ifelse(pred[,'good'] > p, "good", "bad")
        # Get confusion matrix
        cm = table(predicted=tmp, actual=test$good_bad)
        if('good' %in% rownames(cm)) {
          # Calculate TPR, first dim of cm is predicted
          t = cm['good', 'good'] / sum(cm[,'good'])
          # Calculate FPR
          f = cm['good', 'bad'] / sum(cm[, 'bad'])
          # Append to list of values
          tpr = c(tpr, ifelse(is.finite(t), t, 0))
          fpr = c(fpr, ifelse(is.finite(f), f, 0))
        } else {
          tpr = c(tpr, 0)
          fpr = c(fpr, 0)
        }
    }
    df = data.frame(tpr, fpr)
    return(df)
}

pi = seq(0.05, 0.95, 0.05)
pred.optTree = predict(optTree, newdata=test)
pred.bayes = predict(model.bayes, newdata=test, type='raw')
opt.roc = getROC(pred.optTree, pi)
bayes.roc = getROC(pred.bayes, pi)

#Plot
roc.plot = ggplot(mapping=aes(col=Classifier)) +
    geom_point(data=opt.roc, aes(x=fpr, y=tpr, col="Optimal tree")) +
    geom_line(data=opt.roc, aes(x=fpr, y=tpr, col="Optimal tree")) +
    geom_point(data=bayes.roc, aes(x=fpr, y=tpr, col="Bayes")) +
    geom_line(data=bayes.roc, aes(x=fpr, y=tpr, col="Bayes"))

grid.arrange(roc.plot)

# Step 6
# Create model and classify
model.bayes = naiveBayes(formula = good_bad~., data=train)
predict.bayes.test = predict(model.bayes, newdata=test, type="raw")
predict.bayes.train = predict(model.bayes, newdata=train, type="raw")

# Print info on classification with prediction on test
loss = 10/1
predict.bayes.test.loss = ifelse(predict.bayes.test[, 'good'] /
                                 predict.bayes.test[, 'bad'] > loss,
                                 'good',
                                 'bad')
table.bayes = table(Predict=predict.bayes.test.loss, Actual=test$good_bad)
cat("Confusion table of naïve bayes (using test data):")
print(table.bayes)

misclass.bayes = mean(predict.bayes.test.loss != test$good_bad)
cat("Misclassification with naïve bayes (using test data): ", misclass.bayes, "\n")

# Print info on classification with prediction on train
predict.bayes.train.loss = ifelse(predict.bayes.train[, 'good'] /
                                  predict.bayes.train[, 'bad'] > loss,
                                 'good',
                                 'bad')
table.bayes = table(Predict=predict.bayes.train.loss, Actual=train$good_bad)
cat("Confusion table of naïve bayes (using train data): ")
print(table.bayes)

misclass.bayes = mean(predict.bayes.train.loss != train$good_bad)
cat("Misclassification with naïve bayes (using train data): ", misclass.bayes, "\n")
```

Assignment 4, computer code

```
### LAB 2 ###

#-----------# 4: Principal components #-----------#

# Requirements
require("gridExtra") # (for plotting)
require("fastICA")
require("MASS")

# Import data
OrigSpectra <- read.csv2("NIRSpectra.csv")
spectra <- OrigSpectra
spectra$Viscosity = c() # Remove target
# Perform PCA
PCA=prcomp(spectra)
lambda=PCA$sdev^2
pr_var <- lambda/sum(lambda)*100
x = 1:7
var = pr_var[x]
DF <- data.frame(x, var) # Convert to data frame
ggplot(data=NULL)
+ geom_col(data = DF, aes(x=x, y=var, fill="red", colour = "red"))
+ labs(title="Proportional variance vs. PCs" ,x="PC", y="Proportional variance")
+ theme(legend.position="none")
sum((lambda/sum(lambda)*100)[1:2]) # -> 99.64 % of variation in PC1 and PC2

# Plot in coordinates of PC1 and PC2
PC1 = PCA$x[,1]
PC2 = PCA$x[,2]
PCsubset = data.frame(PC1, PC2) # Data projected along PC components
ggplot(data= PCsubset, aes(x = PC1, y = PC2)) + geom_point()

# Trace plots
L1 = PCA$rotation[,1]
L2 = PCA$rotation[,2]
L <- data.frame(L1, L2)
ggplot(data=L, aes(x=seq_along(L1),y=L1))+geom_point()
ggplot(data=L, aes(x=seq_along(L2),y=L2))+geom_point()

# Fast ICA.
set.seed(12345)
X = as.matrix(spectra)

ICA <- fastICA(X, n.comp = 2, fun = c("logcosh","exp"), alpha = 1.0,
               method = "R", row.norm = FALSE, maxit = 200, tol = 1e-04,
               verbose = FALSE, w.init = NULL)
W_prime = ICA$K%*%ICA$W # W' = KW
W1 = W_prime[,1]
W2 = W_prime[,2]
DF = data.frame(W1,W2)
ggplot(data=DF, aes(x=seq_along(W1),y=W1))+geom_point()
ggplot(data=DF, aes(x=seq_along(W2),y=W2))+geom_point()

# Score plots in W´ space
IC1 = ICA$S[,1]
IC2 = ICA$S[,2]
ICsubset = data.frame(IC1, IC2) # Data projected along IC components
ggplot(data= ICsubset, aes(x = IC1, y = IC2)) + geom_point()
```