Lab Report in Machine Learning

Laboration 2

TDDE01

David Grönberg
Davgr686

06-12-2019

Contents

List of Figures

List of Tables

1. Assignments

1.1 Assignment 1

The data file *australian-crabs.csv* contains information about 200 grabs and their characteristics, consisting of 8 columns. The *Sex* column is plotted using carapace lenght (CL) as the x-axis and read width (RW) as the y-axis. Figure 1.1 shows the distribution of the *Sex* column. We can see that there is generally a clear separation between the Male distribution and the Female distribution. This gives us confidence that the data will be easy to classify using linear discriminant analysis (LDA).

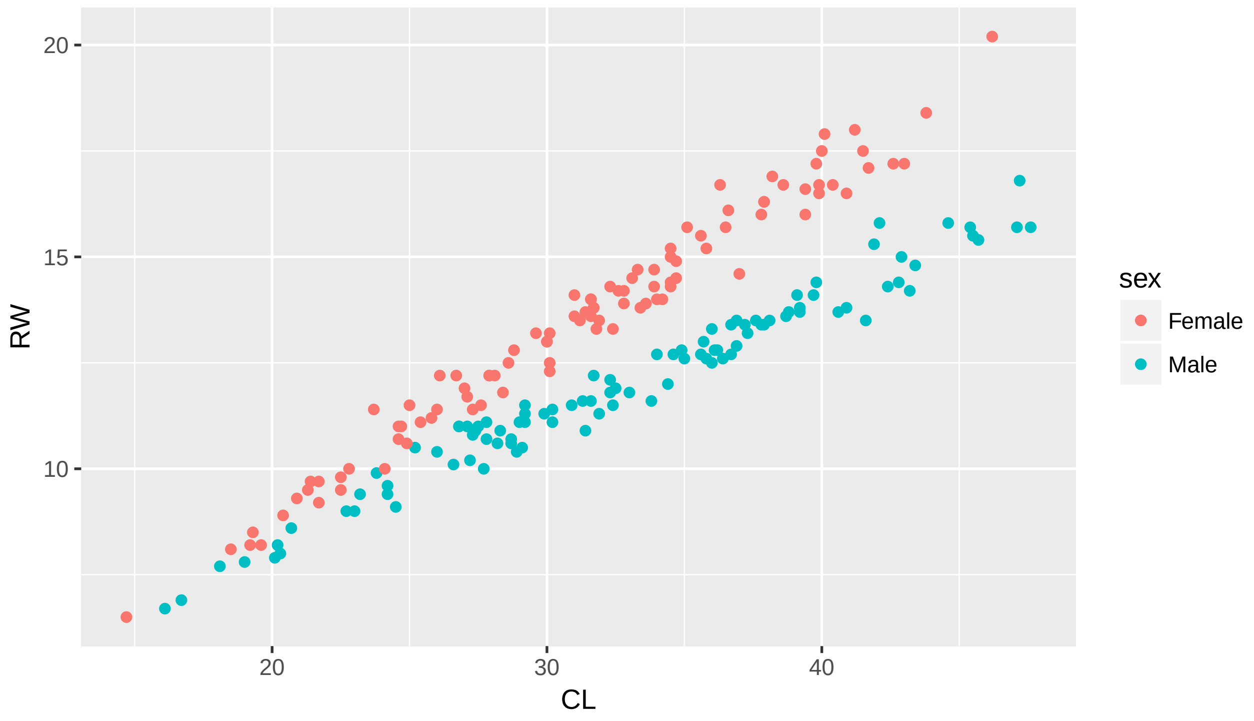

Figure 1.1: Scatterplot of Sex distribution

Next, we use *lda()* from the MASS library to perform LDA and assume a proportional prior (i.e prior probabilities is based on sample sizes). We use CL and RW as our features and the *Sex* column as our target.

```r
ldaModel <- lda(sex ~ CL + RW, data = data)
preds_sex <- predict(ldaModel, data)
```

When comparing figure 1.1 with figure 1.2, we can see that the model performs very well. There are only a few misclassified points, generally in the lower left region where the data is less separated. We observe a low misclassification rate of *0.035* which shows that the quality of the fit is high and the model predicts the *Sex* column with low error.

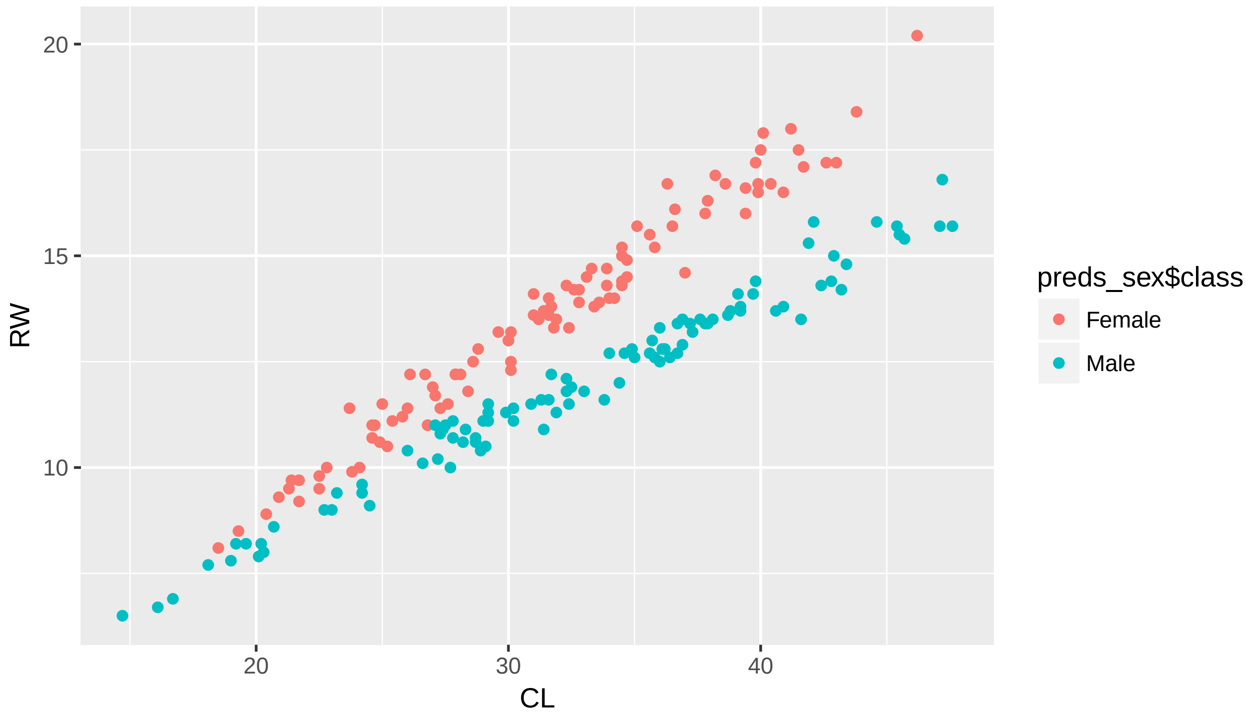

Figure 1.2: Scatterplot of distribution of predicted Sex

We repeat the previous step, but with the use of priors: $p(Male) = 0.9, p(Female) = 0.1$.

```r
ldaModel_prior <- lda(sex ~ CL + RW, data = data, prior=c(.1, .9))
preds_sex_prior <- predict(ldaModel_prior, data)
```

Figure 1.3 shows the distribution of predicted *Sex* using the previous mentioned priors. We can see that the model predicts "Male" to an higher extent, as an result of introducing the prior. the model is 100% accurate when predicting Male. However, it performs worse when predicting Female. The misclassification rate this model yields is *0.08*, which is twice as high compared to the model using a proportional prior.

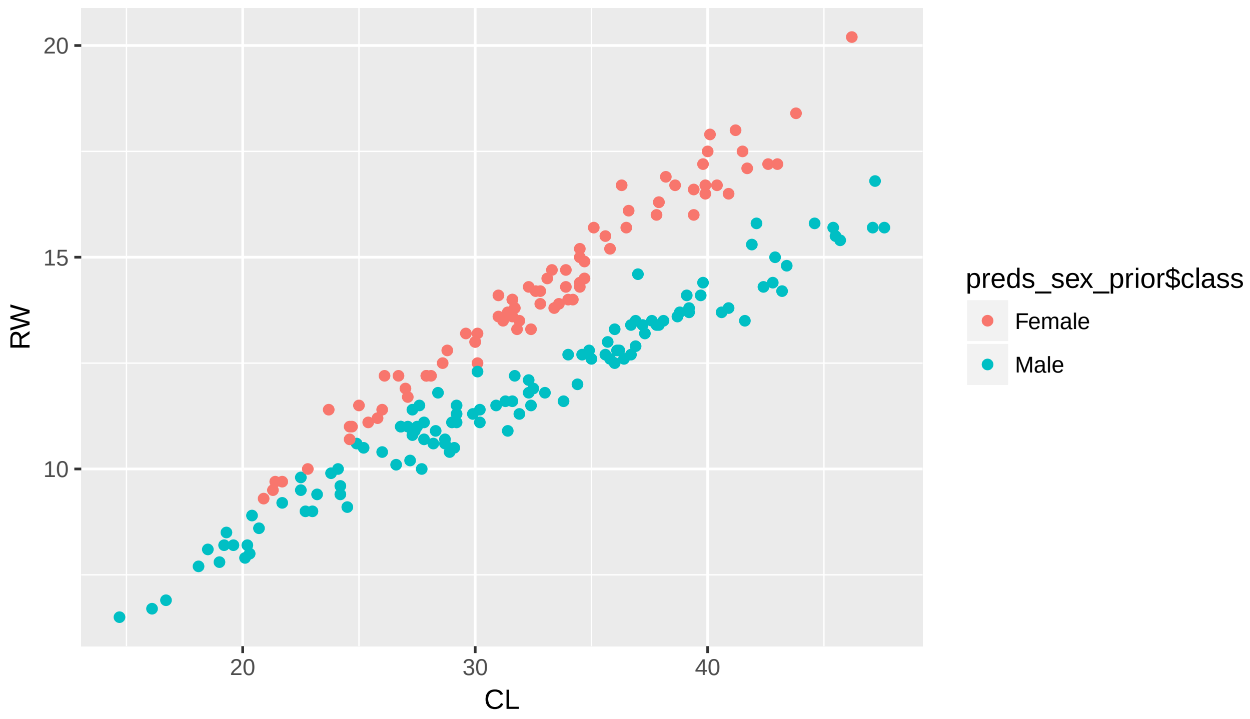

Figure 1.3: Scatterplot of distribution of predicted Sex with priors

We use *glm()* to fit a logistic regression model, calculate the misclassification rate and plot the predictions with a scatterplot.

When comparing figure 1.4 and figure 1.2, we can see that the models yield very similar predictions. The difference is in the bottom left where the different *Sex* are closely clustered. The logistic regression model and the LDA model yields the same misclassification rate (*0.035*).

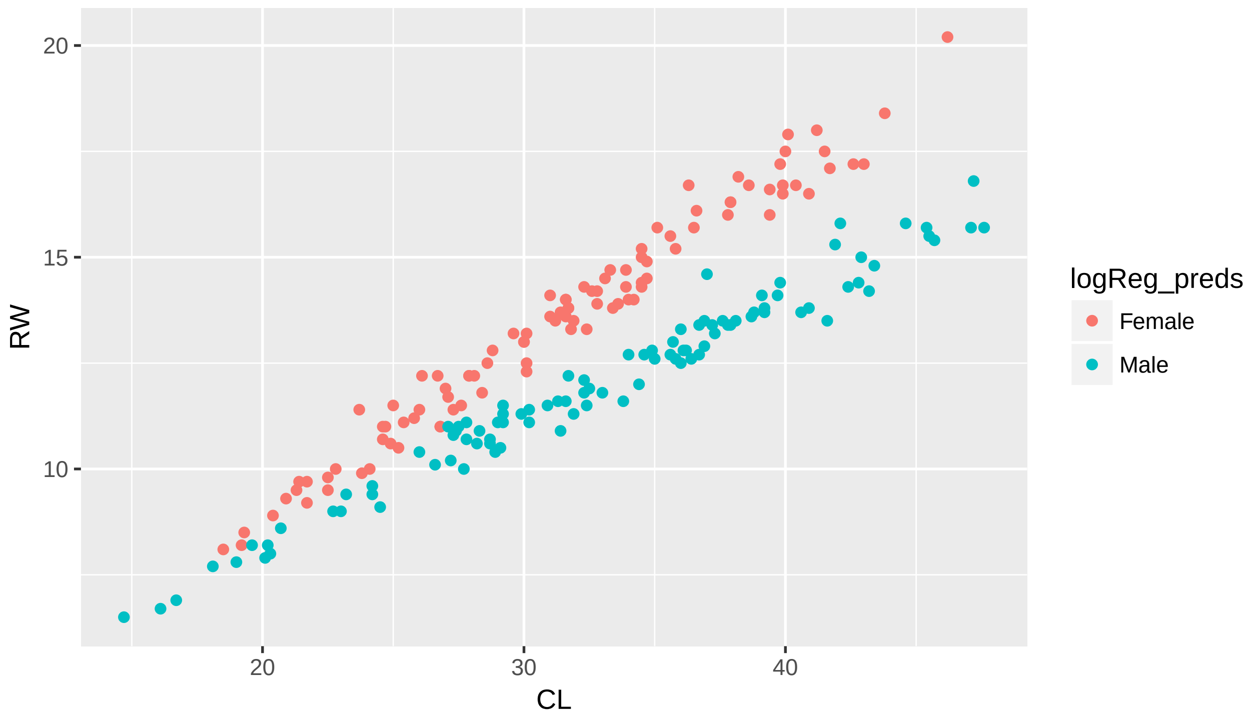

Figure 1.4: Scatterplot of distribution of predicted Sex using glm

To find the decision boundary for a logistic regression model with two predictors, we must find the points that satisfy

$$0 = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2$$

In this case, we want to have the decision boundary on the form of $y = kx + m$, where $m = \frac{-\beta_0}{\beta_2}$ and $k = \frac{-\beta_1}{\beta_2 x_1}$. $\hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2$ can be found by using the *coef* function to extract the coefficients from the model. Figure 1.5 shows the decision boundary for the logistic regression model.

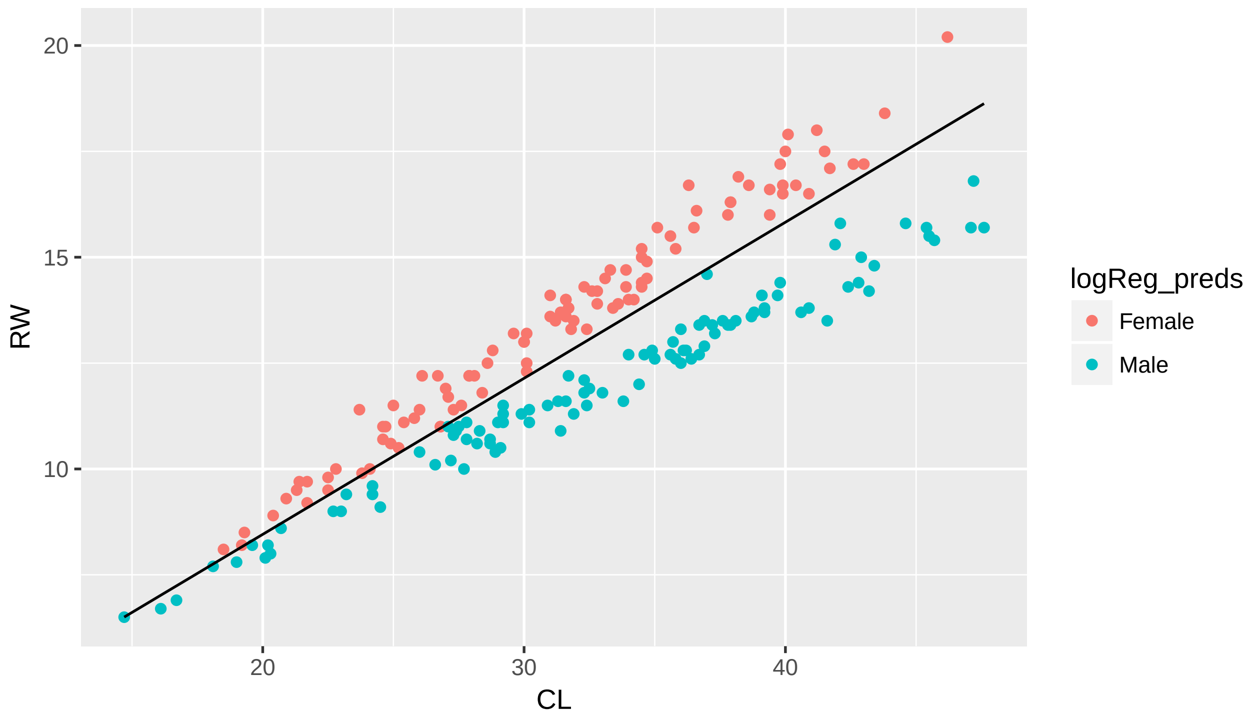

Figure 1.5: Decision boundary for the logistic regression model

1.2 Assignment 2

The excel file *creditscoring.xls* consisting of 20 columns, contain information about customers and their credit score. The task of this assignment is to derive a model that can be used to predict whether or not a customer is likely to pay back their loan.

First, the data is divided into a train, validation and test set. Then, a decision tree is fit using two measures of impurity: *Deviance* and *Gini index*. Figure 1.6 shows the structure of the decision tree when using *Deviance* as a measure of impurity. Here, the tree uses eight variables and the *savings* variable separates the *good bad* variable the best. Figure 1.7 shows the structure of the decision tree when using the *Gini index* when measuring separability. The tree structure is much more complex compared to figure 1.6, has 72 terminal nodes and uses 18 variables.

Table 1.1 shows the misclassification rates for the two tree models. When using the *Deviance* measure, the model has a lower misclassification rate on both the train and test set, when compared to using *Gini index*. Based on these results, the model using the *Deviance* measure is choosen for the following steps.

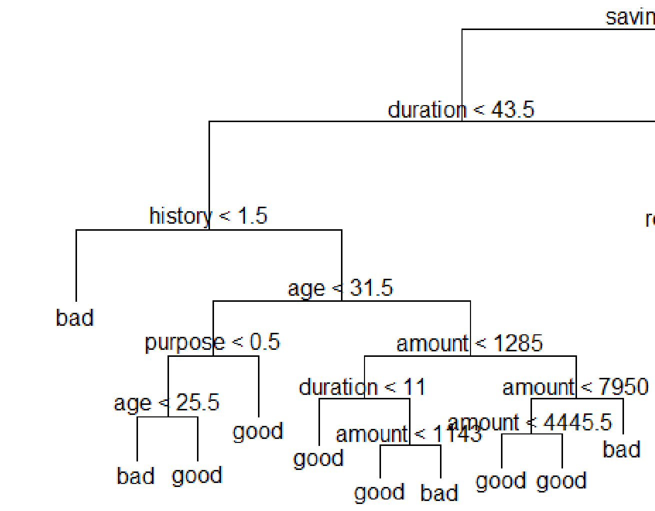

Figure 1.6: Tree structure of tree using Deviance measure

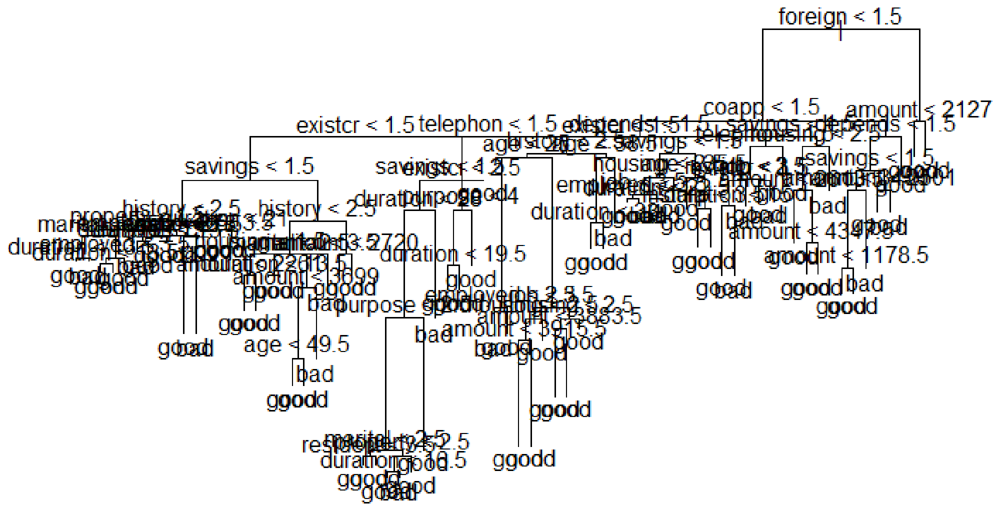

Figure 1.7: Tree structure of tree using Gini index measure

Table 1.1: Misclassification rate for the tree models

Misclassification Rate		
Impurity Measure	Train	Test
Deviance	0.212	0.268
Gini Index	0.240	0.368

After selecting the optimal impurity measure in the previous section, we use the selected model and find the optimal tree depth. The *prune.tree()* function is used to prune the tree and observe the deviance on the test and train set for different numbers of terminal nodes. The optimal number of terminal nodes chosen is based on the combined deviance for the test and train set.

Figure 1.8 shows the deviance for each number of terminal nodes (leaves). As the plot shows, the lowest combined deviance observed is when the model has 12 terminal nodes, thus 12 is chosen to be the optimal number of leaves.

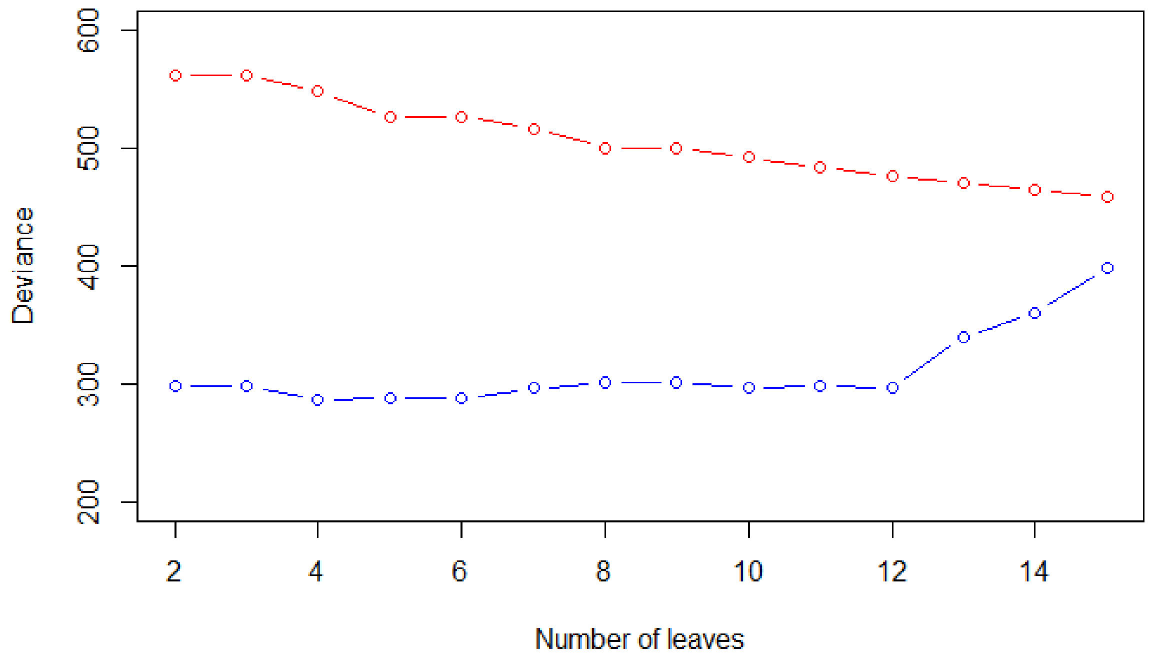

Figure 1.8: Deviance dependency on number of leaves.

The following code block presents relevant information about the optimal tree model using *Deviance* as its impurity measure with twelve terminal nodes and figure 1.9 presents its tree structure.

The model uses eight variables and the *savings* variable separates the target classes the best. The misclassification rate for the train set is *0.213* and *0.268* for the test set.

```
Classification tree:
snip.tree(tree = treeModel_deviance, nodes = c(5L, 77L, 78L))
Variables actually used in tree construction:
```

```
[1] "savings"  "duration" "history"  "age"       "purpose"  "amount"   "other"
[8] "resident"
Number of terminal nodes:  12
Residual mean deviance:  0.9889 = 476.6 / 482
Misclassification error rate (Train data): 0.2126 = 105 / 494
Misclassification error rate (Test data): 0.268
```

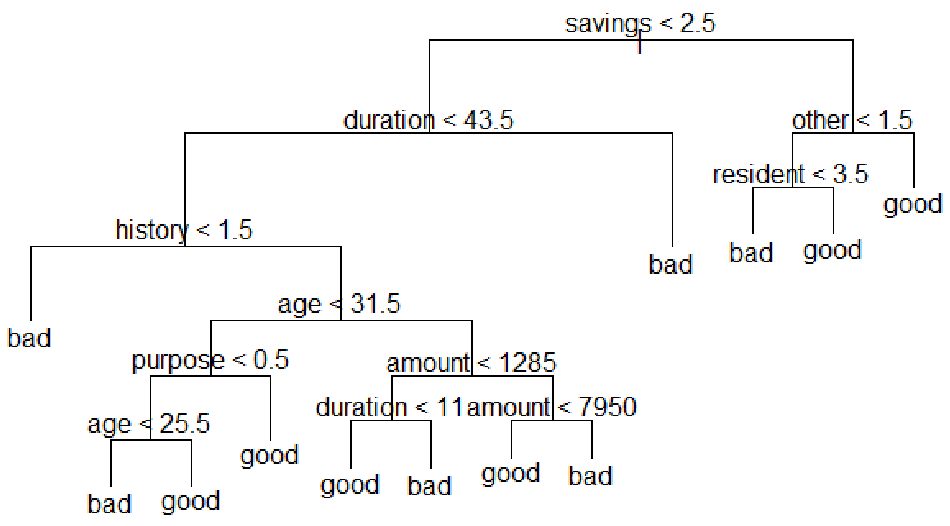

Figure 1.9: Tree structure of optimal tree

Next, a Naive Bayes model is fit using the train set and the *naiveBayes()* function. Table 1.4 presents the misclassification rate on the train and test set. When comparing the results with the optimal tree model in the previous section, we see that the Naive Bayes model has lower accuracy.

Table 1.2: Confusion matrix for the Naive bayes model on the train set

		Predicted	
		Bad	Good
Actual	Bad	95	52
	Good	98	255

Table 1.3: Confusion matrix for the Naive bayes model on the test set

		Predicted	
		Bad	Good
Actual	Bad	46	30
	Good	49	125

Table 1.4: Misclassification rate for the Naive bayes model

Misclassification Rate	
Train set	0.30
Test set	0.32

Next, we use the optimal tree model and the Naive Bayes model to classify the test data with the classification principle:

$$\hat{Y} = 1 \; if \; p(Y = "good" | X) > \pi, \; otherwise \; \hat{Y} = 0$$

where π range between 0.05 and 0.95.

Figure 1.10 shows the ROC curve for the two models, where the x-axis represents the *False Positive Rate (FPR)* and the y-axis represents the *True Positive Rate (TPR)*. The plot shows that the Naive Bayes model has a generally higher True Positive Rate - compared to the optimal tree model - when varying π.

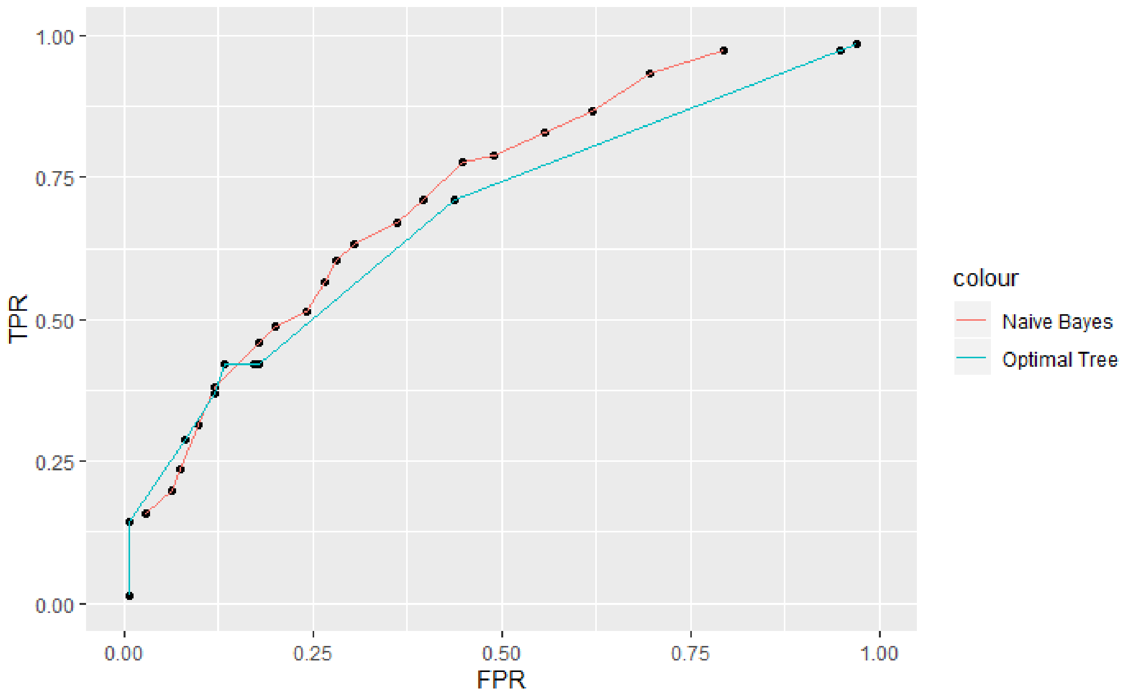

Figure 1.10: ROC curve for the Naive Bayes and Tree model

Lastly, we repeat Naive Bayes classification but introduce a loss matrix that counters the possibility of classifying a "bad" customer "good". This is done by introducing a classification principle in the predict stage:

$$\hat{Y} = "good" \; if \; p(Y = "good" | X) > 10 * p(Y = "bad" | X), \; otherwise \; \hat{Y} = "bad"$$

Figure 1.5 and 1.6 presents the confidence matrix after applying the loss matrix in the prediction stage. Now the model predicts "bad" to a higher extent for both the train and test set, thus minimizing the risk of misclassifying a "bad" customer as "good".

Table 1.5: Confusion matrix for the Naive Bayes model on the test set using a loss matrix

		Predicted	
		Bad	Good
Actual	Bad	71	5
	Good	122	52

Table 1.6: Confusion matrix for the Naive Bayes model on the train set using a loss matrix

		Predicted	
		Bad	Good
Actual	Bad	137	10
	Good	263	90

1.3 Assignment 4

The csv file *NIRspectra.csv* consisting of 127 columns that contain information about near-infrared spectra and viscosity levels for diesel fuels. The task of this assignment is to investigate how the spectra can be used to predict viscosity.

First, a standard Principle Component Analysis (PCA) is conducted using the *prcomp()* function.

Figure 1.11 presents the percent of variation that is explained by each Principle Component. We can see that PC1 accounts for almost all the variation in the data. The cumulative proportion of variance explained by PC1 and PC2 is more than 99%.

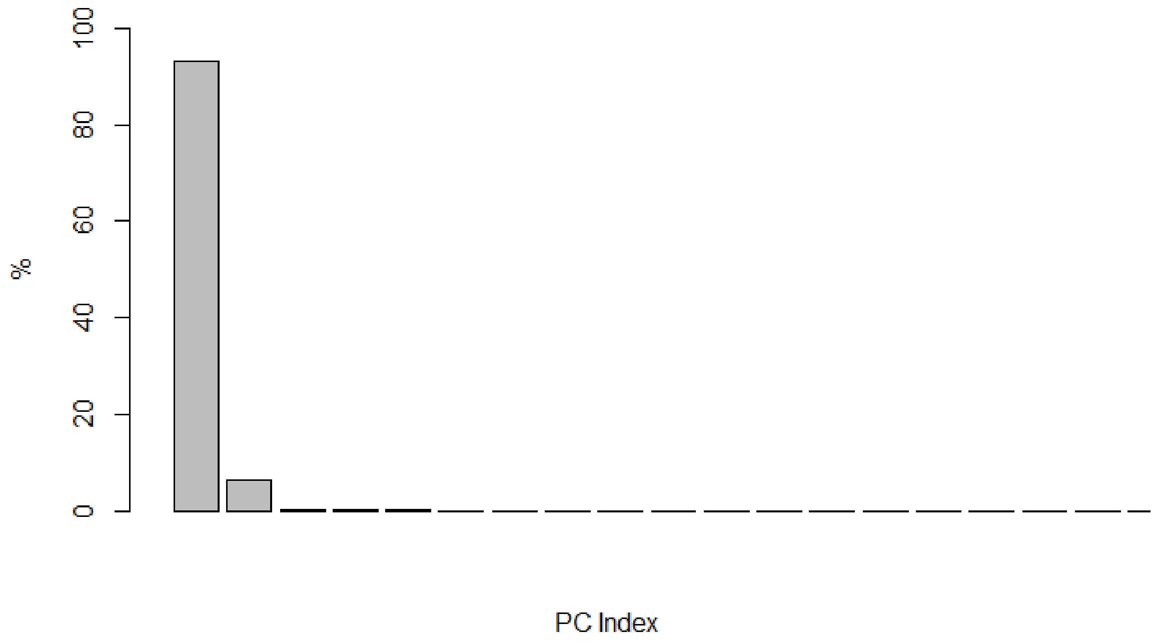

Figure 1.11: Percentage of variation explained by each PC

Figure 1.12 shows a plot of the diesel fuels in the PC1 (x-axis) and PC2 (y-axis) coordinate system. The fuels are generally clustered together on the left side of the graph, with some outliers observed in the middle and right side of the graph. Because PC1 accounts for almost all the variation in the data, points that are far from each other in the x-axis are very different from each other.

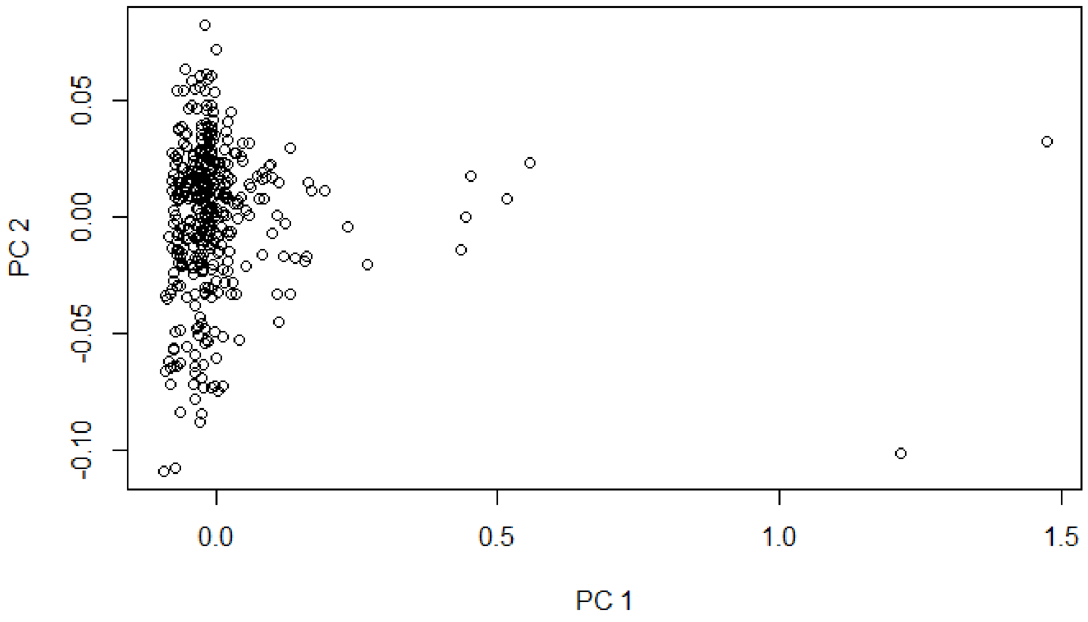

Figure 1.12: Scores in the PC1 PC2 coordinates

Figure 1.13 shows a traceplot for Principle Component 1 and 2. The x-axis represents the features and y-axis represents the weight. The plot shows how many of the features the Principle Component need to project the data. PC1 seems to use all the features (with weights ranging around 0.08 - 0.11), while PC2 seems to give features 115-127 high weights, and the rest around zero.

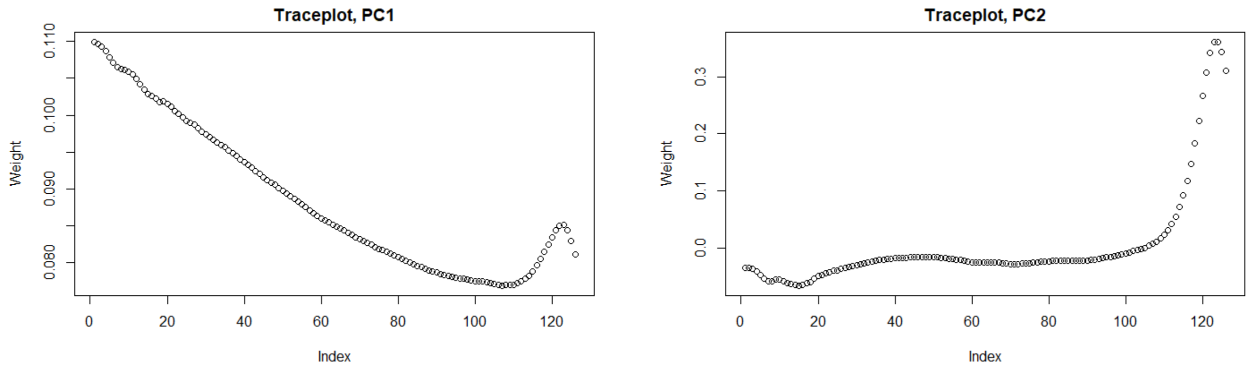

Figure 1.13: Traceplot of PC1 PC2

Next, we perform Independent Component Analysis (ICA) with two components using the *fastICA* package. We compute

$$W' = K \cdot W$$

where K is a pre-whitening matrix that projects data on the principle components, W is an estimated un-mixing matrix and *W'* is an estimated un-mixing matrix projected on the principle components. The columns of *W'* is plotted using a traceplot in figure 1.14. When comparing figure 1.14 to 1.13, it seems that the traces are mirrored, with different weights.

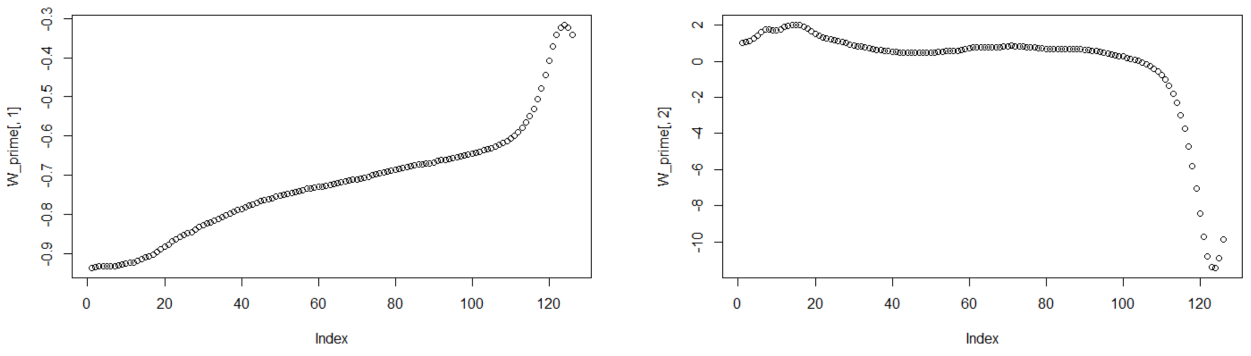

Figure 1.14: Traceplot of W' column 1 2

Figure 1.15 is similar to 1.12, but is mirrored with a different scale.

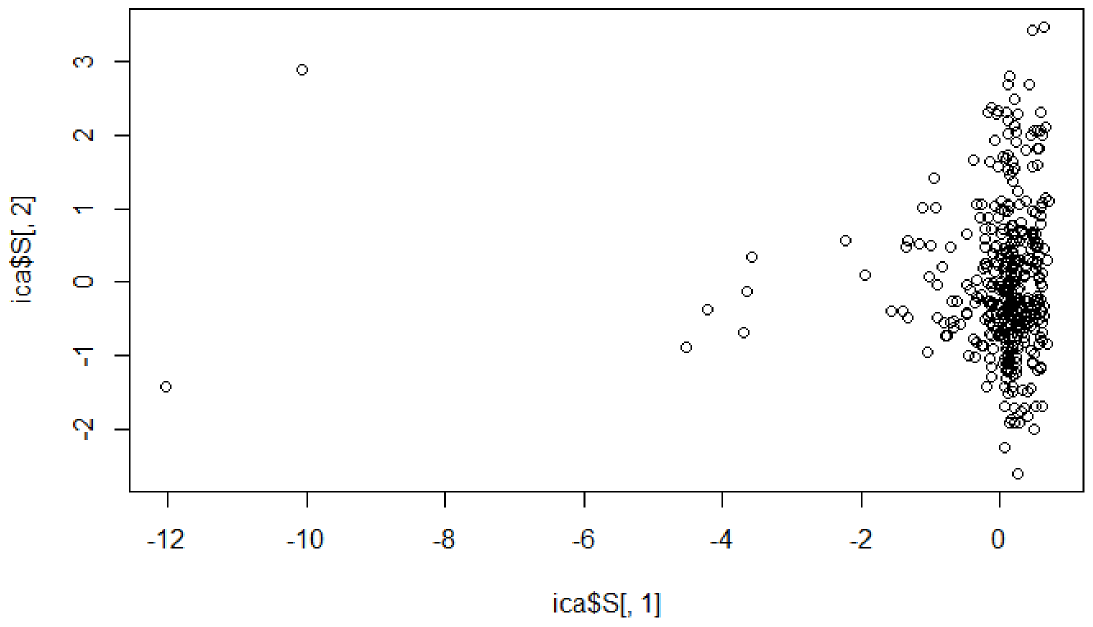

Figure 1.15: Scores in the PC1 PC2 coordinates

2. Code Appendix

2.1 Assignment 1

```r
library(ggplot2)
library(MASS)

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

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

ggplot(data, aes(x=CL, y=RW)) + geom_point(aes(color = sex))

ldaModel <- lda(sex ~ CL + RW, data = data)
preds_sex <- predict(ldaModel, data)
ggplot(data, aes(x=CL, y=RW)) + geom_point(aes(color = preds_sex$class))

table("Actual" = data$sex, "Predicted" = preds_sex$class)
mean(preds_sex$class != data$sex)

ldaModel_prior <- lda(sex ~ CL + RW, data = data, prior=c(.1, .9))
ldaModel_prior$prior
preds_sex_prior <- predict(ldaModel_prior, data)
ggplot(data, aes(x=CL, y=RW)) + geom_point(aes(color = preds_sex_prior$class))
table("Actual" = data$sex, "Predicted" = preds_sex_prior$class)
mean(preds_sex_prior$class != data$sex)

logRegModel <- glm(sex ~ CL + RW, data = data, family = binomial)

logReg_probs <- predict(logRegModel, data, type = "response")
logReg_preds <- ifelse(logReg_probs > 0.5, "Male", "Female")
table("Actual" = data$sex, "Predicted" = logReg_preds)
mean(logReg_preds != data$sex)
ggplot(data, aes(x=CL, y=RW)) + geom_point(aes(color = logReg_preds))

slope <- coef(logRegModel)[2]/(-coef(logRegModel)[3])
intercept <- coef(logRegModel)[1]/(-coef(logRegModel)[3])
ggplot(data, aes(x=CL, y=RW)) + geom_point(aes(color = logReg_preds)) +
    stat_function(fun=function(x) {intercept + slope*x}, color = "black", size = 1)
```

2.2 Assignment 2

```r
require(readxl)
library(tree)
library(e1071)
library(ggplot2)

```

```r
data <- read_excel("creditscoring.xls")
data$good_bad <- as.factor(data$good_bad)

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,]

treeModel_deviance <- tree(good_bad ~., data=train, split = "deviance")
treeModel_gini <- tree(good_bad ~., data=train, split = "gini")
plot(treeModel_deviance); text(treeModel_deviance)
plot(treeModel_gini); text(treeModel_gini)
summary(treeModel_deviance)
summary(treeModel_gini)

treeModel_deviance.pred_test <- predict(treeModel_deviance, test, type = "class")
treeModel_gini.pred_test <- predict(treeModel_gini, test, type = "class")

treeModel_deviance.pred_train <- predict(treeModel_deviance, train, type = "class")
treeModel_gini.pred_train <- predict(treeModel_gini, train, type = "class")

mean(treeModel_deviance.pred_test != test$good_bad)
mean(treeModel_gini.pred_test != test$good_bad)

mean(treeModel_deviance.pred_train != train$good_bad)
mean(treeModel_gini.pred_train != train$good_bad)


j = 15


trainScore=rep(0,j)
testScore=rep(0,j)
for(i in 2:j)
  {
    prunedTree=prune.tree(treeModel_deviance,best=i)
    pred=predict(prunedTree, newdata=valid, type="tree")
    trainScore[i]=deviance(prunedTree)
    testScore[i]=deviance(pred)
}

  plot(2:j, trainScore[2:j], xlab = "Number of leaves", ylab = "Deviance", type="b", col="red"
    , ylim=c(200,600))
  points(2:j, testScore[2:j], type="b", col="blue")

  combinedD <- (trainScore[2:j] + testScore[2:j])/2
  bestI <- match(min(combinedD), combinedD) + 1

 optimalLeavesTree = prune.tree(treeModel_deviance,best=bestI)
summary(optimalLeavesTree)
plot(optimalLeavesTree)
text(optimalLeavesTree)

optimalLeavesTree.prob_test <- predict(optimalLeavesTree, test, type = "class")
```

```r
mean(optimalLeavesTree.prob_test != test$good_bad)


NBclassfier = naiveBayes(good_bad~ ., data=train)
NBclassfier.pred_test <- predict(NBclassfier, test, type = "class")
NBclassfier.pred_train <- predict(NBclassfier, train, type = "class")

table("Actual" = test$good_bad, "Predicted" = NBclassfier.pred_test )
table("Actual" = train$good_bad, "Predicted" = NBclassfier.pred_train )
mean(NBclassfier.pred_test != test$good_bad)
mean(NBclassfier.pred_train != train$good_bad)


NBclassfier.prob_test <- predict(NBclassfier, test, type = "raw")
NBclassfier.pred_train <- predict(NBclassfier, train, type = "raw")
optimalLeavesTree.prob_test <- predict(optimalLeavesTree, test, type = "vector")

pi = seq(0.05, 0.95, 0.05)
TPR.nb <- rep(0, length(pi))
FPR.nb <- rep(0, length(pi))
TPR.tree <- rep(0, length(pi))
FPR.tree <- rep(0, length(pi))
for (i in 1:length(pi))
{
  optimalLeavesTree.pred_test <- ifelse(optimalLeavesTree.prob_test[,2] > pi[i], "good", "bad"
    )
  tab <-table("Predicted" = optimalLeavesTree.pred_test, "Actual" = test$good_bad )
  TP <- tab[1]
  FN <- tab[2]
  FP <- tab[3]
  TN <- tab[4]

  TPR.tree[i] <- TP/(TP + FN)
  FPR.tree[i] <- FP/(FP + TN)


  NBclassfierWCutoff.pred_test <- ifelse(NBclassfier.prob_test[,2] > pi[i], "good", "bad")
  tab <-table("Predicted" = NBclassfierWCutoff.pred_test, "Actual" = test$good_bad )
  TP <- tab[1]
  FN <- tab[2]
  FP <- tab[3]
  TN <- tab[4]

  TPR.nb[i] <- TP/(TP + FN)
  FPR.nb[i] <- FP/(FP + TN)
}

plotdf = data.frame("FPR" = FPR.nb, "TPR" = TPR.nb, "FPR_tree" = FPR.tree, "TPR_tree" = TPR.
    tree)


ggplot() + geom_point(data = plotdf, mapping = aes(x = FPR, y = TPR)) + geom_line(data =
    plotdf, mapping = aes(x = FPR, y = TPR, colour = "Naive Bayes")) + geom_point(data =
    plotdf, mapping = aes(x = FPR_tree, y = TPR_tree)) + geom_line(data = plotdf, mapping =
    aes(x = FPR_tree, y = TPR_tree, colour = "Optimal Tree")) +xlim(0,1) + ylim(0,1)



NBclassfier.pred_test.raw <- predict(NBclassfier, test, type = "raw")
NBclassfier.pred_train.raw <- predict(NBclassfier, train, type = "raw")

loss_train <- ifelse(NBclassfier.pred_train.raw[,2] > (NBclassfier.pred_train.raw[,1] * 10), "
    good", "bad")
loss_test <- ifelse(NBclassfier.pred_test.raw[,2]/NBclassfier.pred_test.raw[,1] > 10, "good",
    "bad")


table("Actual" = test$good_bad, "Predicted" = loss_test )
table("Actual" = train$good_bad, "Predicted" = loss_train )
```

2.3 Assignment 4

```r
library(ggplot2)
library(fastICA)

data <- read.csv("NIRspectra.csv", sep = ";", header = TRUE, dec=",")

pca <- prcomp(data[1:126])
summary(pca)

pca.var <- pca$sdev^2
pca.var.percent <- round(pca.var/sum(pca.var)*100, 1)
barplot(pca.var.percent, ylim = c(0,100), xlab = "PC Index", ylab = "%", xlim = c(0,20))
# PCA 1-3 explaines 99.6% of the total variance

plot(pca$x[,1], pca$x[,2], xlab = "PC 1", ylab = "PC 2")


U = pca$rotation
plot(U[,1], main = "Traceplot, PC1", ylab = "Weight")
plot(U[,2], main = "Traceplot, PC2", ylab = "Weight")

set.seed(12345)
ica <- fastICA(data[1:126], 2)
W_prime <- ica$K %*% ica$W

plot(W_prime[,1])
plot(W_prime[,2])

plot(ica$S[,1],ica$S[,2])
```