Machine Learning-Lab2Block1

Nastaran Meftahi, Mohsen Pirmoradiyan and Atieh Khaleghi

12/8/2019

Assignment 2. Analysis of credit scoring

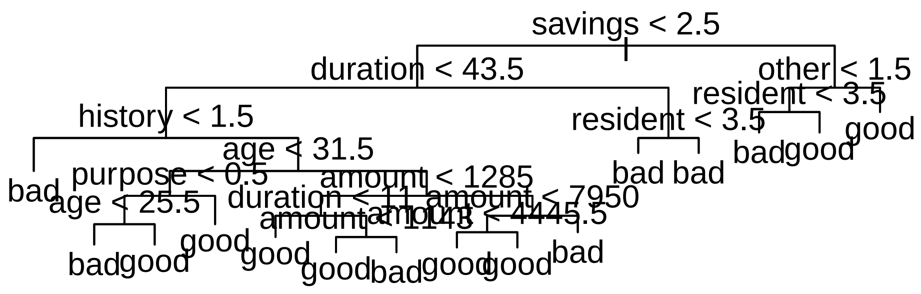
2.1. Import the data to R and divide into training/validation/test as 50/25/25:

At the first step data was imported and splitted to training, validation, and test sets.

2.2. Fit a decision tree

a) Deviance as the measure of impurity

A model was fitted to the training data and the deviance measurement was implemented.



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

Evaluating the model implementing the test data:

```
##       tree_dev_pred
##        bad good
##   bad   28   48
##   good  19  155
## Misclassification Rate: 0.268
```

b) Gini index as the measure of impurity

A model was fitted to the training data and the gini index was implemented.

```
##
## Classification tree:
## tree(formula = good_bad ~ ., data = train, split = "gini")
## Variables actually used in tree construction:
##  [1] "foreign"  "coapp"    "depends"  "telephon" "existcr"  "savings"
##  [7] "history"  "property" "marital"  "duration" "employed" "age"
## [13] "housing"  "amount"   "purpose"  "resident" "job"      "installp"
## Number of terminal nodes:  72
## Residual mean deviance:  1.015 = 428.5 / 422
## Misclassification error rate: 0.2368 = 117 / 494
```

As the result shows the misclassification rate for the training phase is 23.68%.

Evaluating the model implementing the test data:

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

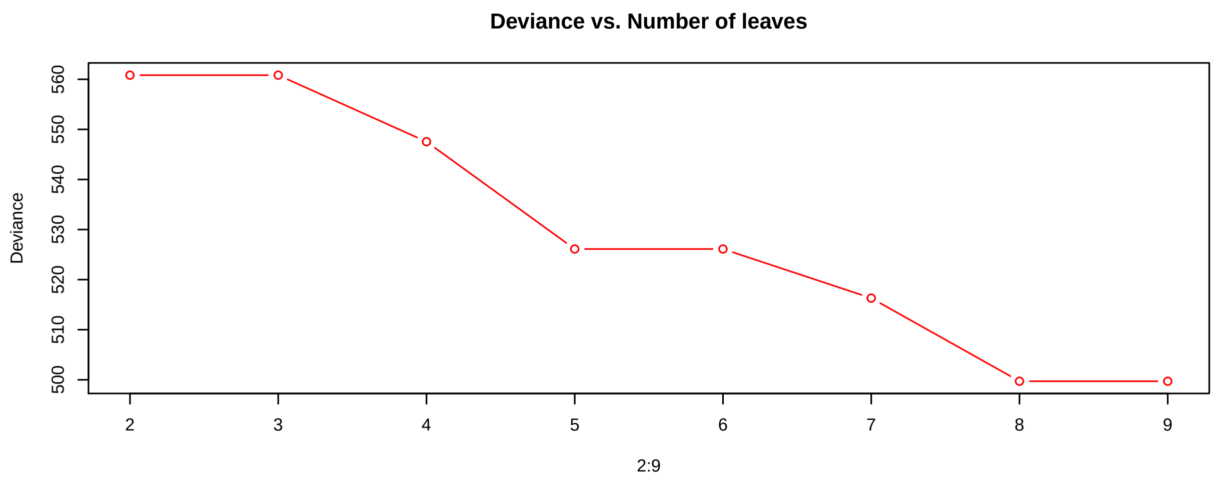
```
## Misclassification Rate: 0.368
```

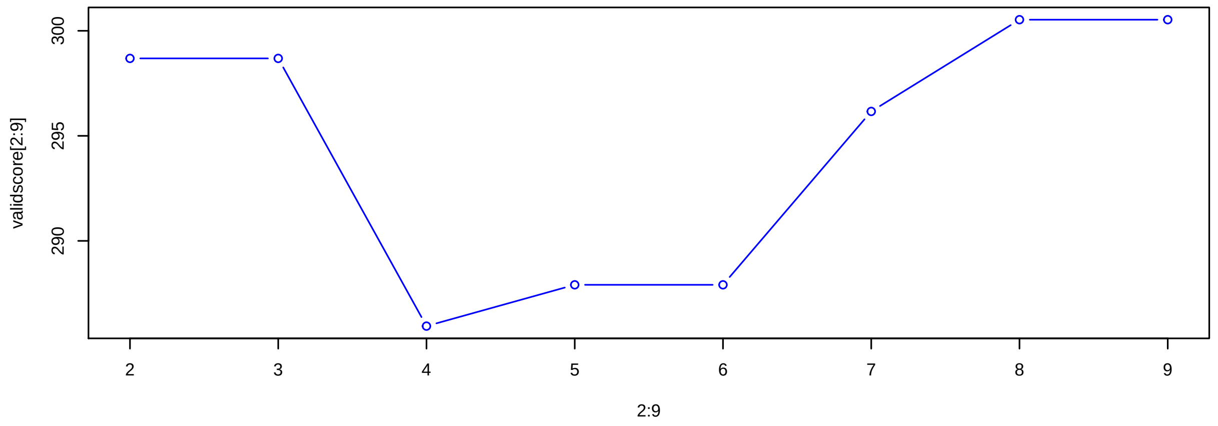
Misclassification rate resulted from the test data was 36.8%.

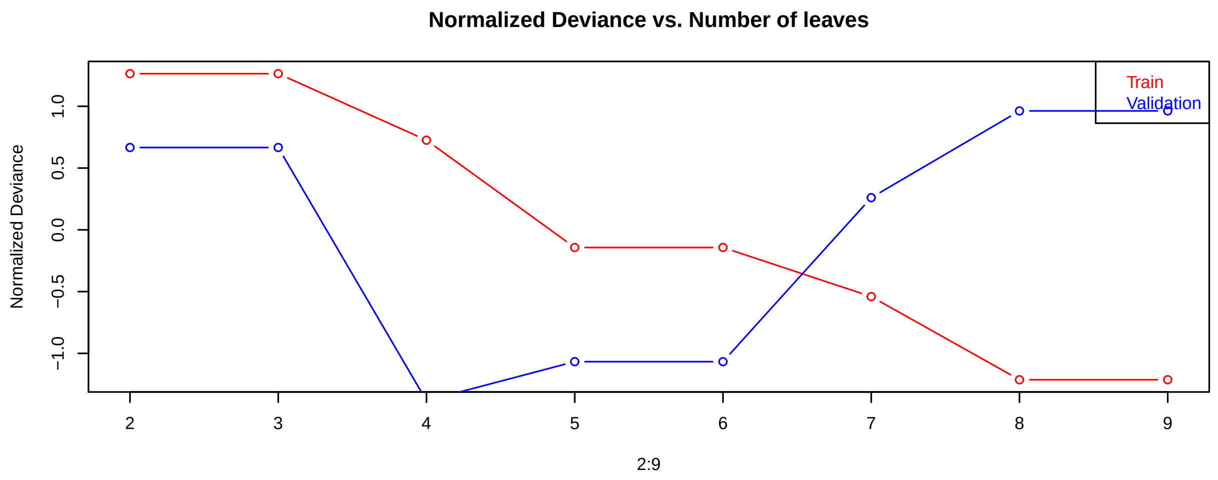
Overall the performance of "Deviance-based Model" can be concluded to be better than the Gini-based model. Therefor, for the next steps this model will be used. The results show that the training phase misclassification rates for both models were much less than those of the test stage. From this point we can conclude the overfitting problem within the decision tree algorithm.

2.3. Use training and validation sets to choose the optimal tree depth.

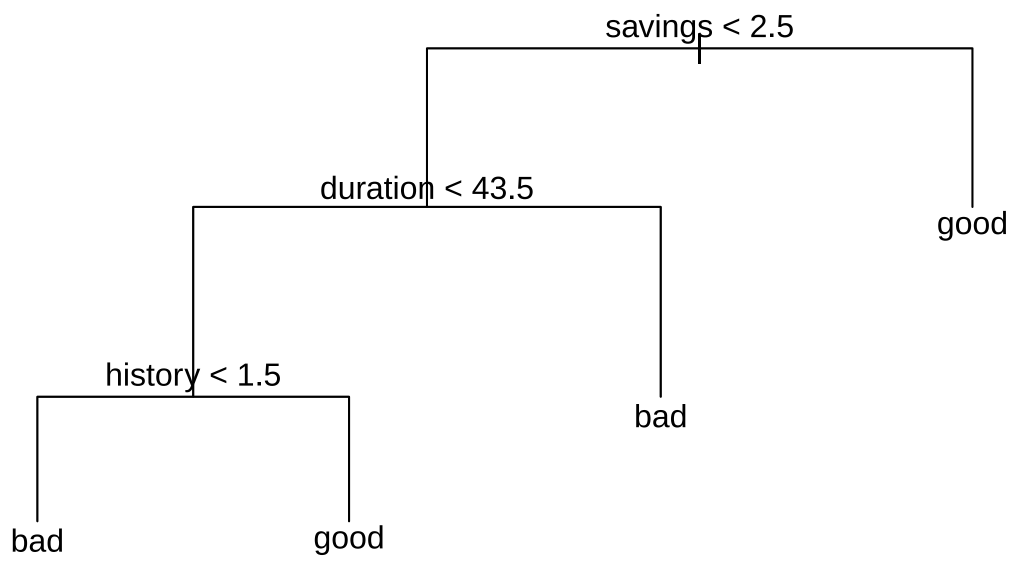
Function prune.tree was used to choose the optimal tree. Deviances versus number of trees were plotted for training and validation data sets:







The Deviances decreased for the training set with increasing the number of leaves. The trend, however, is different for the validation data. The graph shows a minimum at leaves of 4. As a result, this number was selected for the optimal tree.



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

As the results indicate, choosing a 4-leave tree as the optimal tree resulted in a model in which three valiables were used including: savings, duration, and history. This model ended in 4 terminal nodes. The misclassification for this tree were 25%. The feature "savings" was selected as the root node and based on the threshold value of 2.5 splitting at this node occured. Those with a saving more than 2.5 were classified as the good ones and classification of the rest was determined based on the "duration" variable. At this node the splitting was based on the value of 43.5, for which values more than that considered as bad, and the rest were splitted according to the feature"history". The decision at this node was made based on the value of 1.5.

The final model was evaluated using the test data:

```
##         testFit
## good_bad bad good
##     bad   18   58
##     good   6  168
```

```
## Misclassification Rate: 0.256
```

2.4. Use training data to perform classification using Naïve Bayes

A model fitted to the train data using the "naiveBayes" function from the package "e1071". The correspnding confusion matrix and misclassification rate are as follows:

```
##          fit_train
## good_bad bad good
##     bad   95   52
##     good  98  255
```

```
## Misclassification Rate: 0.3
```

The prediction procedure carried out on test data:

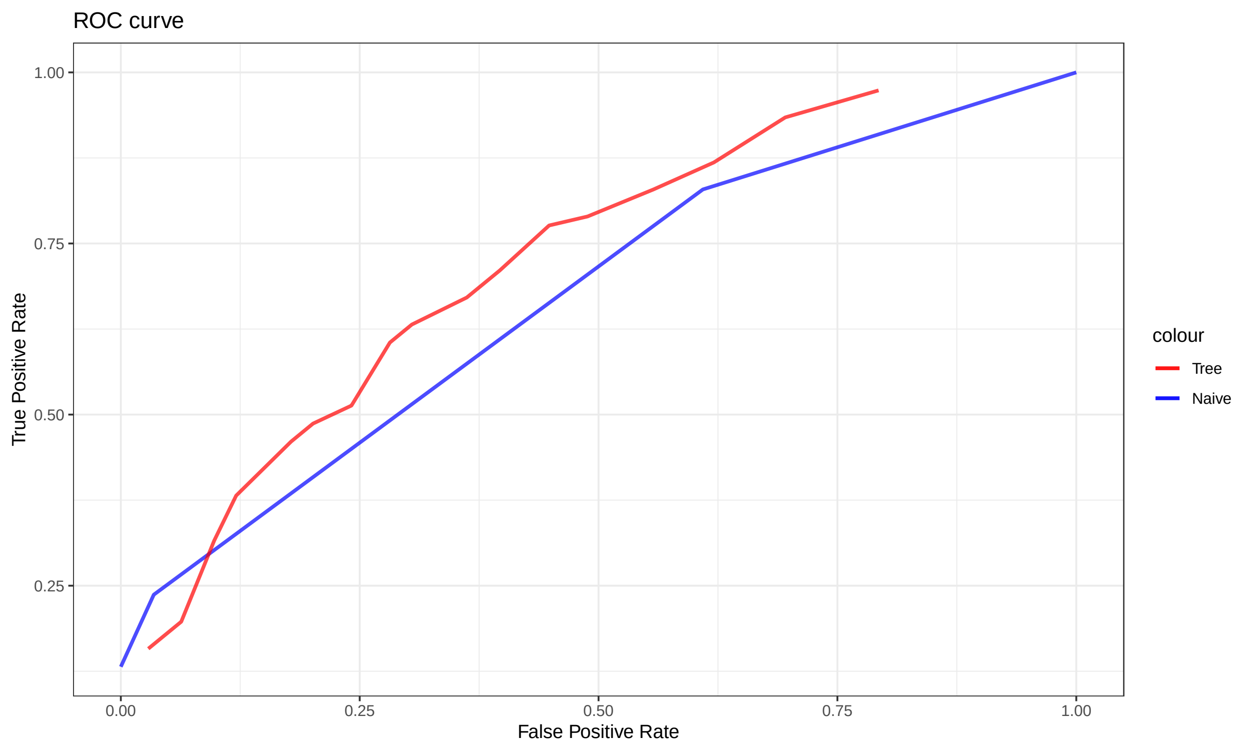
```
##          fit_test
## good_bad bad good
##     bad   46   30
##     good  49  125
```

```
## Misclassification Rate: 0.316
```

In comparison to the results obtaind in (2.3), it can be concluded that the decision tree had a better performance.

2.5. ROC curve

A vector of thresholds($\pi = .05, .1, ..., .095$) implemented to calculate the TPR and FPR for the optimal tree and Naive models. The red line is the ROC plot of tree model and the blue one corresponds to the Naive model. For a same FPR value the corresponding TPR value for the tree model is higher than that of the Naive model. In other words, the area under the curves is higher for the tree model. This indicates the better performance of the classification done by decision tree algorithm.



2.6. Naive model with introducing a loss matrix

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

```
## Misclassification Rate: 0.546
```

And for the test set:

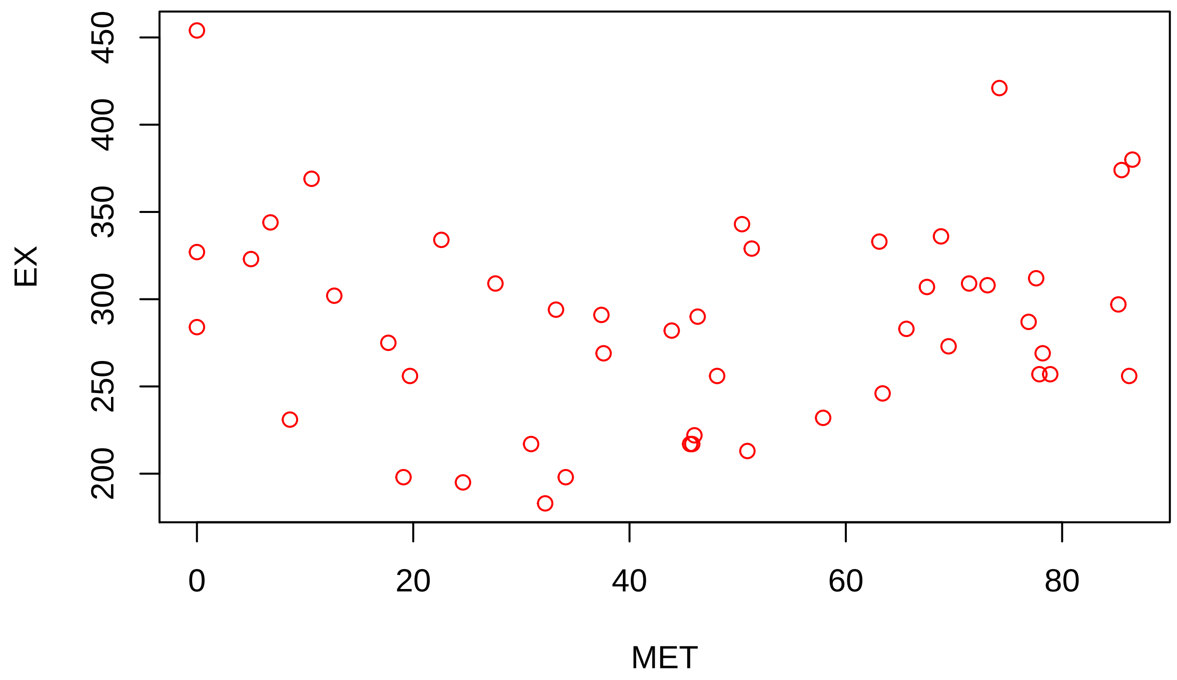
```
##          weighted_pred
## good_bad bad good
##      bad   71    5
##      good 122   52
```

```
## Misclassification Rate: 0.508
```

The loss matrix is such that we have to pay more cost if we categorized a bad one into "good" class. In fact, the loss we should apply for this case must be 10 times the loss we should apply for the opposite classifying, i.e. classifying a good one into the "bad" category. The way to do this is penalizing the probablity of classifying y to be "good" to be 10 times higher than the probability of classifying y to be "bad". Having applied this penalizing, we can identify that the number of misclassification of bad ones into good class ceclined sharply, although the overall misclassification worsened.

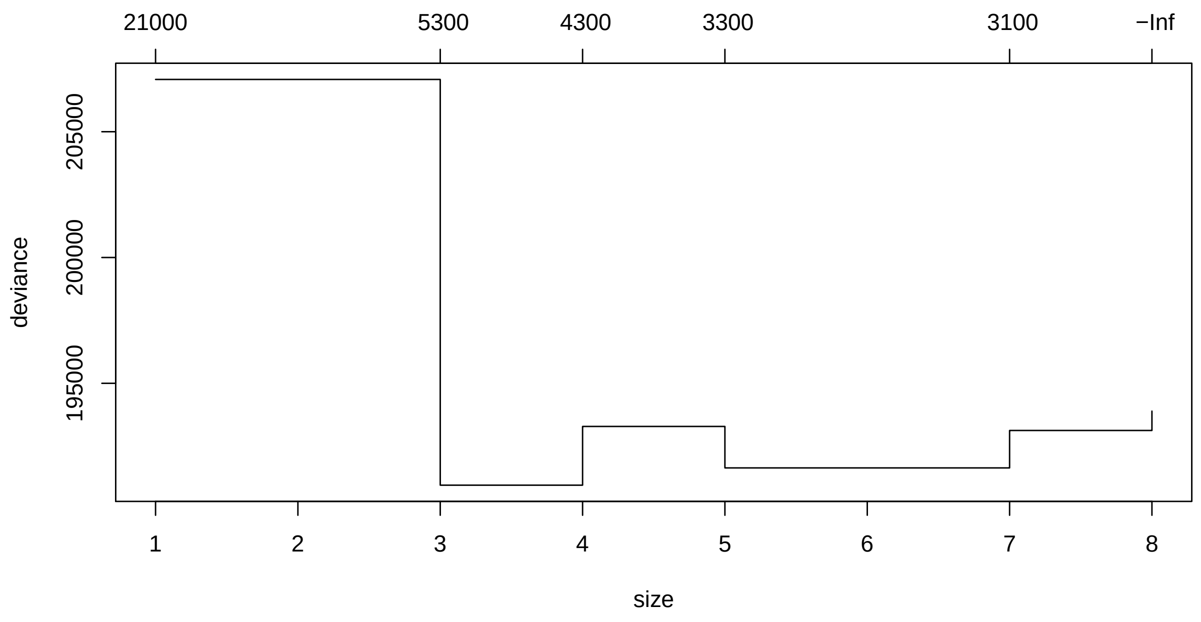
Assignment 3. Uncertainty estimation

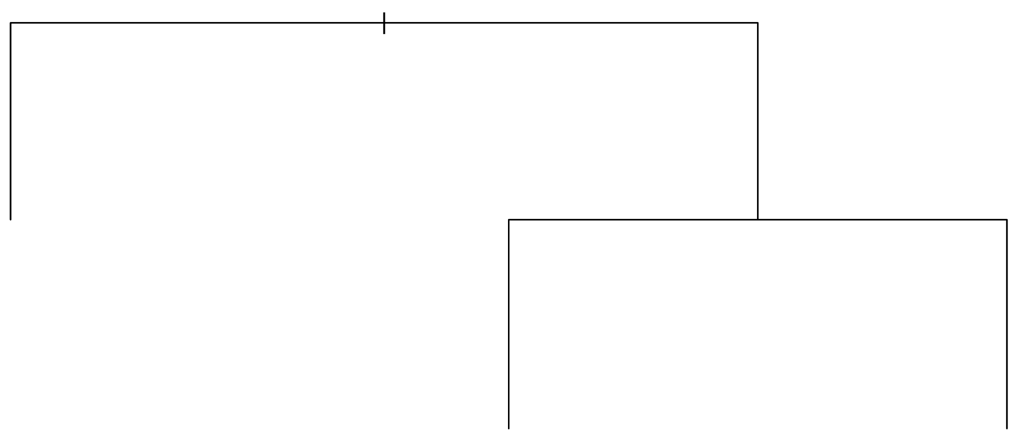
Reorder and plot the data

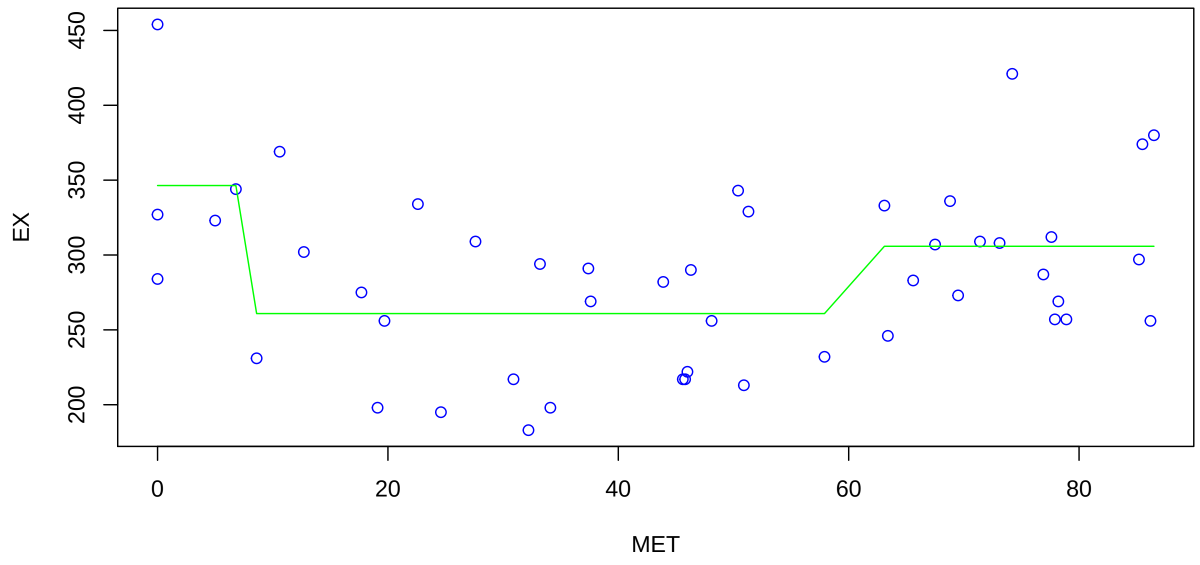


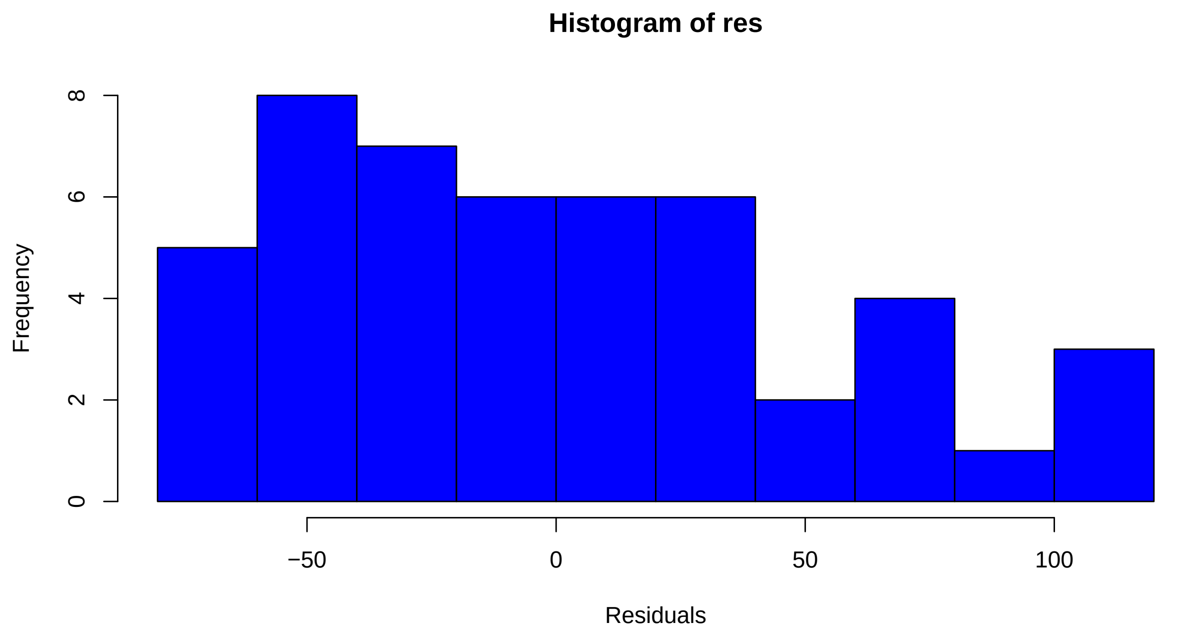
Fit a regression tree

A regression tree was fitted to the data and a cross validation was run. The plot above shows the Deviance variations with the number of the leaves. According to this figure “3” was selected the best value for the number of leaves in order to pruning the tree.

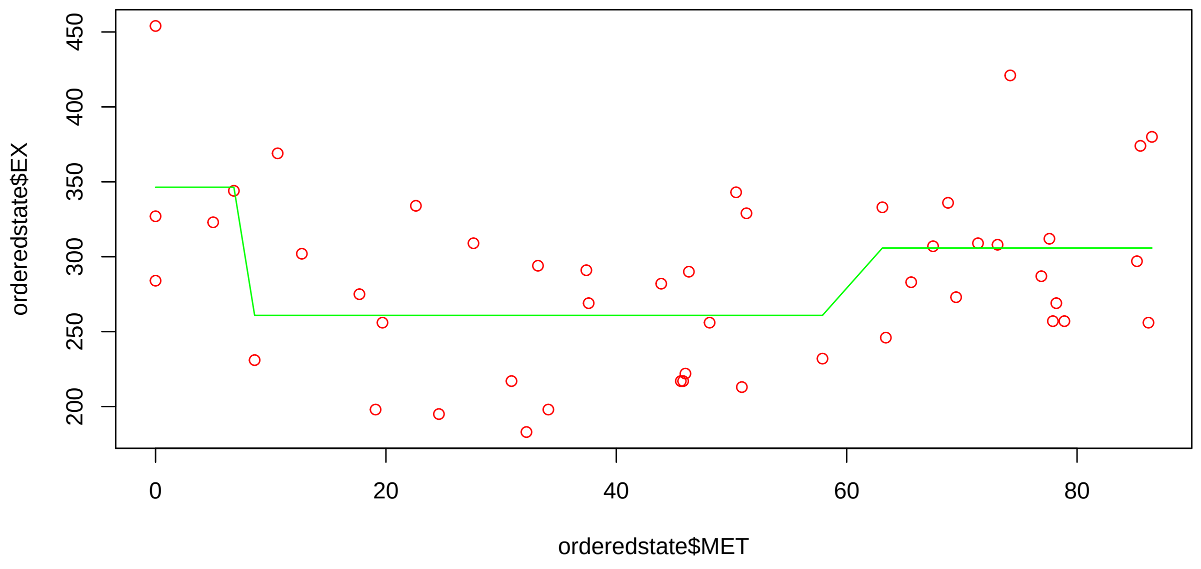

21000
5300
4300
3300
3100
−Inf
205000
200000
195000
deviance
1
2
3
4
5
6
7
8
size





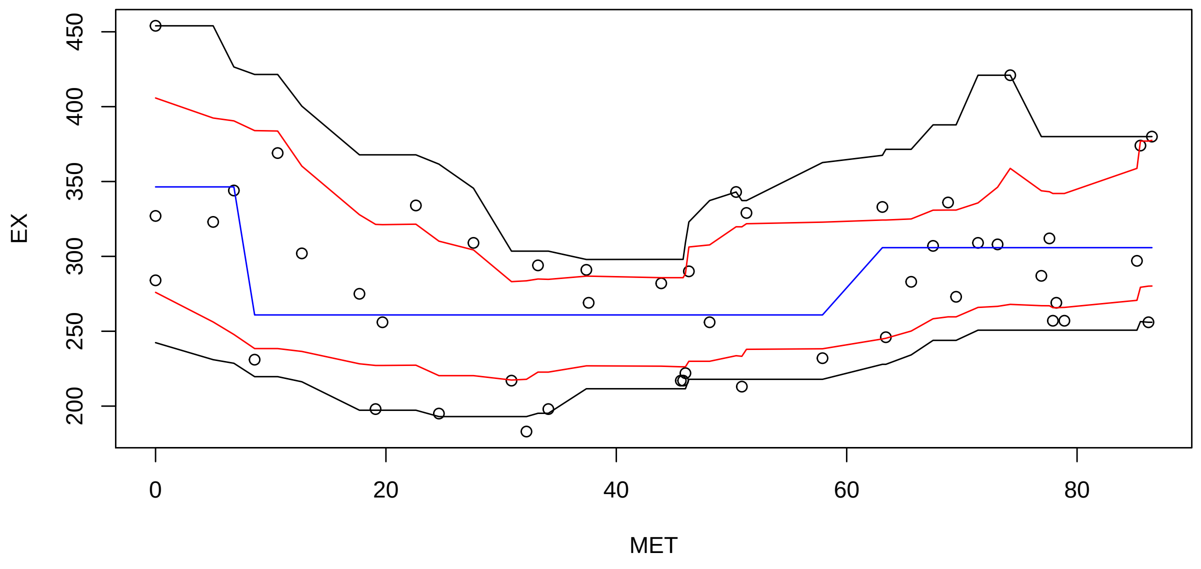


The residuals of the model are not normally distributed around the mean.



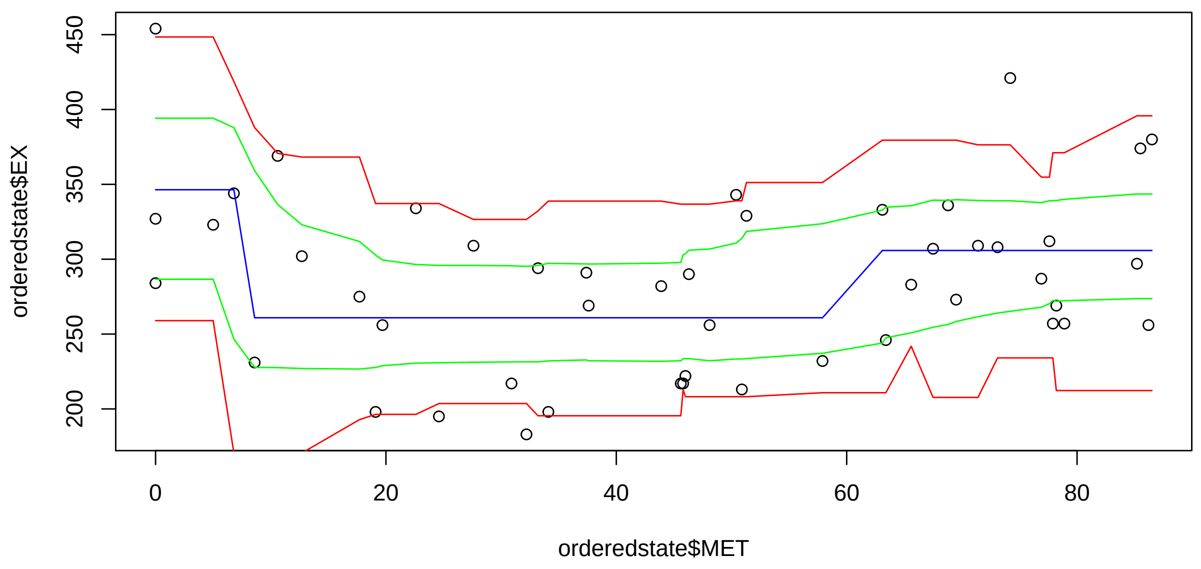
Compute and plot the 95% confidence bands by a non-parametric bootstrap

```
## Warning in prune.tree(reg.fun, best = 3): best is bigger than tree size
```



The confidence band appears to be bumpy. It almost changes with the levels of the fitted model. The classifier does not provide a very good fit to the model, so the confidence band is bumpy.

Compute and plot the 95% confidence bands by a parametric bootstrap



Considering the confience band of the models, we can see that the parametric model provides a better and smoother fit compared to the non-parametric model.

Assignment 4: Principal Components

4.1 PCA

PCA was conducted on a dataset containing information on viscosity levels and infrared-spectra of diesel fuels. The data was scaled prior to performing the PCA. A plot was made showing the proportion of variance explained by each principal component. Most of the variation is explained by the first handful principal components.

```
# 4.1 PCA

setwd("E:/Master Documents/Machine Learning/Lab2block1")
spectra <- read.csv2("NIRspectra.csv")
spectra <- spectra %>% select(-Viscosity)
pca_spectra <- prcomp(x = spectra , scale. = TRUE)
#ggbiplot(pca_spectra, var.axes=FALSE)
pca_prop <- summary(pca_spectra)
```

```
df_pca <- data.frame(prop_solo = pca_prop_solo, prop_cum = pca_prop_cum,
                     pca_index = 1:length(pca_prop_solo))
```

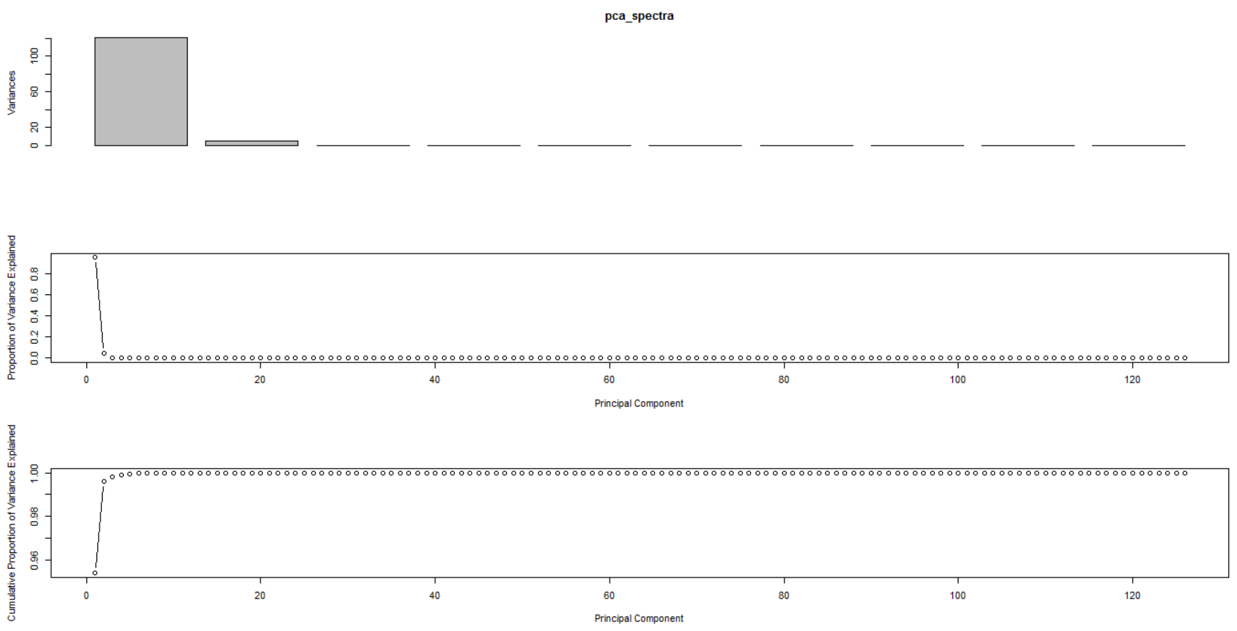
According to prop_cum we only require 2 first principal components in order to explain at least 99 percent of the variation.

```
# Minimal number of components explaining at least 99% of total variance
df_pca[1:(sum(pca_prop_cum < 0.99) + 1), ]
```

```
prop_solo prop_cum pca_index
```

PC1 0.95385 0.95385 1 PC2 0.04229 0.99614 2

```
df_pcs <- data.frame(PC1 = pca_spectra$x[,1], PC2 = pca_spectra$x[,2])
std_dev <- pca_spectra$sdev
pr_var <- std_dev^2
#pr_var
#proportion of variance explained
prop_var <- pr_var/sum(pr_var)
```



There are two clear outliers in on the PC1 axis (`PC1 > 100`) which may be labelled as unusual diesel fuels.

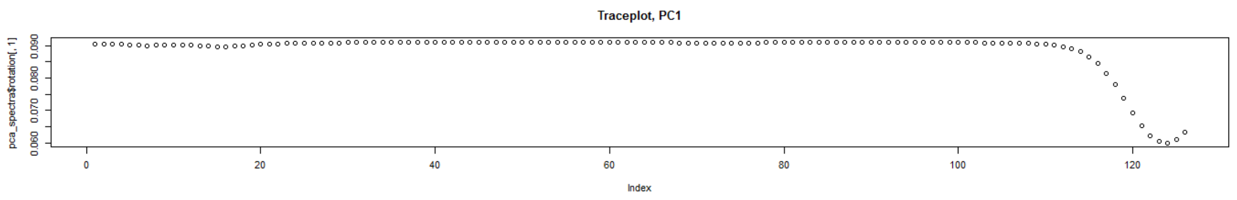
4.2 Trace plot of loadings

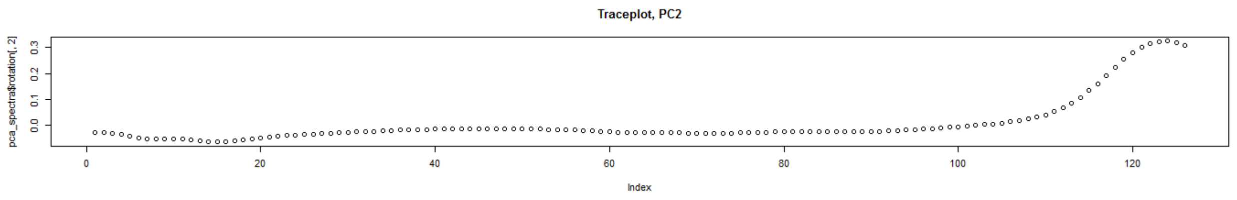
Here, principal component loadings for all the variables were extracted and plotted for the two components. We can see that `PC2` is mainly explained by a few variables.

```
# 4.2 Loadings
df_loadings <- data.frame(pc1_loading = pca_spectra$rotation[,1],
                          pc2_loading = pca_spectra$rotation[,2],
                          var = rownames(pca_spectra$rotation),
                          var_index = 1:nrow(pca_spectra$rotation))


#plot(pca_spectra$rotation[,1], main="Traceplot, PC1")
#plot(pca_spectra$rotation[,2],main="Traceplot, PC2")

#df_loadings %>%
#  tidyr::gather(key = "pc", value = "loading", -var, -var_index) %>%
#  ggplot(aes(x = var_index, y = loading, color = pc)) +
#  geom_point() +
#  labs(title = "PC loadings for variables") +
#  theme_minimal()
```





The table below displays the variables with high loadings for `PC2`. Since we scaled the data prior to performing PCA: The diesel fuels with higher than average near-infrared spectra on these variables can be interpreted to have higher `PC2` values. The opposite applies to diesel fuels with below average near-infrared spectra.

```
#df_loadings %>%
 # mutate(id = row_number()) %>%
  #tidyr::gather(key = "pc", value = "loading", -var, -var_index, -id) %>%
  #dplyr::filter(id > 118 & pc == "pc2_loading") %>%
  #xtable() %>%
  #print(comment=FALSE)
```

4.3 Independent component analysis

An independent component analysis was performed. Afterwards the independent component loadings (eigenvectors) were computed through the formula $\boldsymbol{W'} = \boldsymbol{KW}$. They are plotted below.

a)

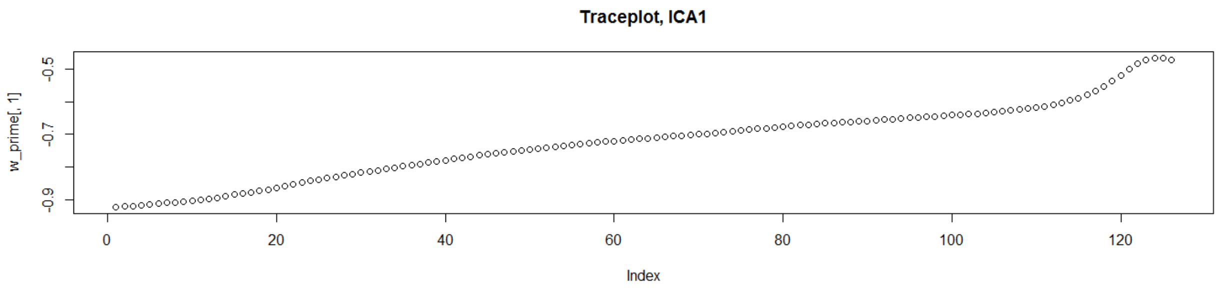
$\boldsymbol{W'}$ represents loadings (or in other words: eigenvectors). It describes which variables contribute the most to the respective component scores.

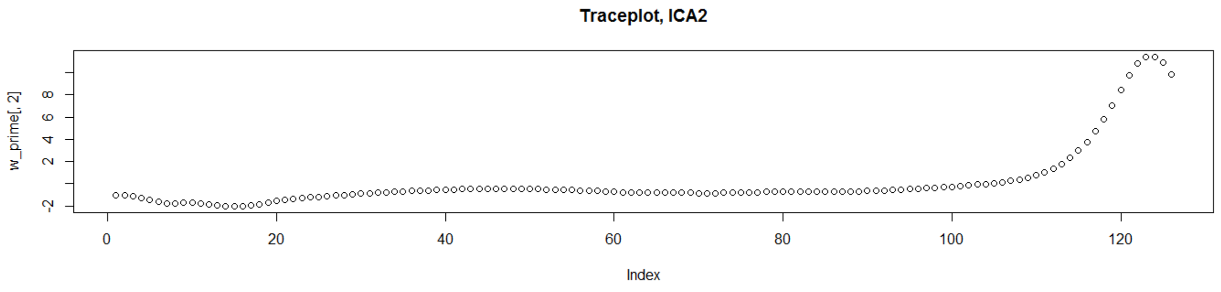
```
# 4.3 Score plot
set.seed(12345)
spectra <- as.matrix(spectra)
ica_mod <- fastICA(X = spectra ,
                   n.comp = 2)

w_prime <- (ica_mod$K %*% ica_mod$W)

#par(mfrow = c(2,1))
```

```r
#plot(w_prime[,1], main="Traceplot, ICA1")
#plot(w_prime[,2],main="Traceplot, ICA2")
```

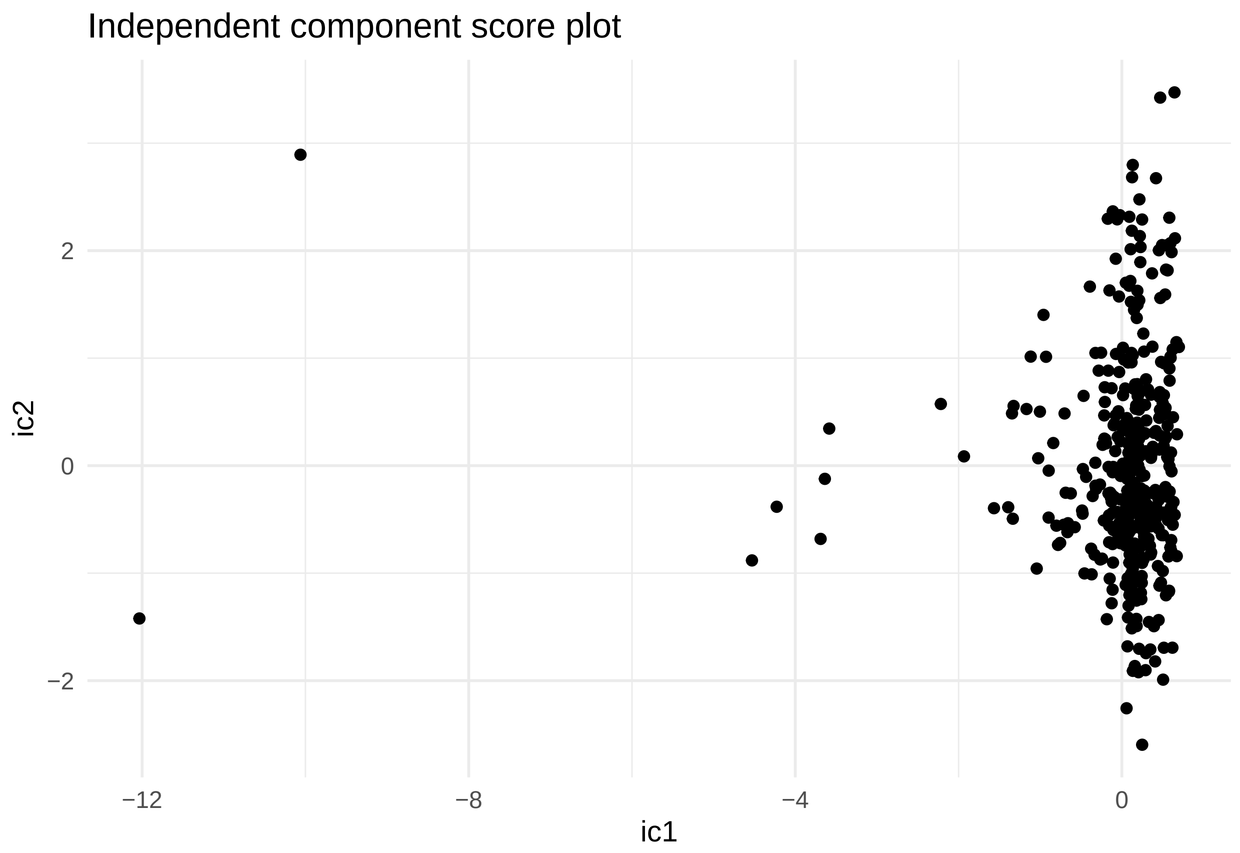




b)

The scores for the ICA can be found in the S-matrix. The scores look like a mirrored version of the PCA score plot. IC1 has mostly negative scores whereas PC1 had positive. The sign usually doesn't matter as it depends on the implementation of the algorithm. We can conclude the two methods produce fairly similar score plots (although mirrored).

```r
ica_mod$S %>%
  as.data.frame() %>%
  rename(ic1 = V1, ic2 = V2) %>%
  ggplot(aes(x = ic1, y = ic2)) +
  geom_point() +
  labs(title = "Independent component score plot") +
  theme_minimal()
```


Independent component score plot
ic2
2
0
−2
−12
−8
−4
0
ic1

Appendix

```r
rm(list = ls())
RNGversion('3.5.1')
library(ggplot2)
library(readxl)
library(dplyr)
library(xtable)
library(data.table)
library(boot)
#library(ggbiplot)
library(fastICA)
knitr::opts_chunk$set(echo = TRUE)

credit_data <- read_excel("E:/Master Documents/Machine Learning/Lab2block1/creditscoring.xls")
credit_data <- as.data.frame(credit_data)
credit_data$good_bad = as.factor(credit_data$good_bad)

#Spliting data int train, valid, and test
n=dim(credit_data)[1]


set.seed(12345)

id <- sample(1:n, floor(n * 0.5))
train <- credit_data[id,]

id1 <- setdiff(1:n, id)
set.seed(12345)
id2 <- sample(id1, floor(n * 0.25))
valid <- credit_data[id2,]

id3 <- setdiff(id1, id2)
test <- credit_data[id3,]

library(tree)
#RNGversion("4.5.1")
#set.seed(12345)
tree_dev <- tree(good_bad~., data = train, split = "deviance")
plot(tree_dev)
text(tree_dev)
summary(tree_dev)
#RNGversion("4.5.1")
#set.seed(12345)
tree_dev_pred <- predict(tree_dev, newdata = test, type = "class")
table(test$good_bad,tree_dev_pred )
cat("Misclassification Rate:", 1-mean(test$good_bad == tree_dev_pred))
tree_gini <- tree(good_bad~., data = train, split = "gini")
#plot(tree_gini)
#text(tree_gini)
summary(tree_gini)
tree_dev_gini <- predict(tree_gini, newdata = test, type = "class")
table(test$good_bad, tree_dev_gini)
```

```r
cat("Misclassification Rate:", 1-mean(test$good_bad == tree_dev_gini))
#RNGversion("4.5.1")
#set.seed(12345)
trainscore = rep(0,9)
validscore = rep(0,9)

for (i in 2:9) {
  PrunedTree <- prune.tree(tree = tree_dev, best = i)
  pred <- predict(PrunedTree, newdata = valid, type = "tree")
  trainscore[i] <- deviance(PrunedTree)
  validscore[i] <- deviance(pred)
}
par(mfrow=c(3,1))
plot(2:9, trainscore[2:9], type = "b", col = "red",
      ylab = "Deviance", main = "Deviance vs. Number of leaves")
plot(2:9, validscore[2:9], type = "b", col = "blue")
plot(2:9, scale(trainscore[2:9]), type = "b", col = "red",
      ylab = "Normalized Deviance", main = "Normalized Deviance vs. Number of leaves")
points(2:9, scale(validscore[2:9]), type = "b", col = "blue")
legend("topright", legend = c("Train", "Validation"), text.col = c("red", "blue"))

#RNGversion("4.5.1")
#set.seed(12345)
Tree <- tree(good_bad~., data = train, split = "deviance")
finalTree = prune.tree(tree = Tree, best = 4)
plot(finalTree)
text(finalTree, pretty = 0)
summary(finalTree)
set.seed(12345)
testFit <- predict(finalTree, newdata = test, type = "class")
with(test, table(good_bad, testFit))
cat("Misclassification Rate:", 1-mean(test$good_bad == testFit))

library(e1071)
Naive_fit <- naiveBayes(good_bad~., train)


fit_train <- predict(Naive_fit, newdata = train)
with(train, table(good_bad, fit_train))
cat("Misclassification Rate:", 1-mean(train$good_bad == fit_train))
fit_test <- predict(Naive_fit, newdata = test)
with(test, table(good_bad, fit_test))
cat("Misclassification Rate:", 1-mean(test$good_bad == fit_test))


set.seed(12345)
pi = seq(0.05, .95, by=0.05)
n = length(pi)
TPR_Tree <- vector(length = n)
FPR_Tree <- vector(length = n)

for (i in 1:n) {
  newtest_T = predict(finalTree, newdata = test)
  newtest_T
  newtest_pred = ifelse(newtest_T[,2] >pi[i], "1", "0")
  CMT = table(test$good_bad, newtest_pred)
  TPR_Tree[i] <- CMT[1,1]/sum(CMT[1,])
  FPR_Tree[i] <- CMT[2,1]/sum(CMT[2,])

}

set.seed(12345)
TPR_Naive <- vector(length = n)
FPR_Naive <- vector(length = n)

for (i in 1:n) {
  newtest_N = predict(Naive_fit, newdata = test, type = "raw")

  newtest_pred_N = ifelse(newtest_N[,2] >pi[i], "1", "0")
  CMN = table(test$good_bad, newtest_pred_N)
  TPR_Naive[i] <- CMN[1,1]/sum(CMN[1,])
  FPR_Naive[i] <- CMN[2,1]/sum(CMN[2,])

}
OptTree = data.frame(FPR_Tree = FPR_Tree, TPR_Tree = TPR_Tree)
Naive = data.frame(FPR_Naive = FPR_Naive, TPR_Naive = TPR_Naive)
df = cbind(OptTree, Naive)



ggplot(df)+geom_line(mapping=aes(FPR_Tree,TPR_Tree, color = "red"),size = 1, alpha = 0.7)+
  geom_line(mapping=aes(FPR_Naive, TPR_Naive, color = "blue"), size = 1, alpha = 0.7)+
  labs(title= "ROC curve",
       x = "False Positive Rate",
       y = "True Positive Rate")+
  scale_color_manual(labels = c("Tree", "Naive"), values = c("red", "blue")) +
  theme_bw()

set.seed(12345)
Naive_fit <- naiveBayes(good_bad~., train)

fit_train_prob <- predict(Naive_fit, newdata = train, type = "raw")
weighted_pred <- ifelse(fit_train_prob[,2]>10*fit_train_prob[,1], "good", "bad")
with(train, table(good_bad, weighted_pred))
cat("Misclassification Rate:", 1-mean(train$good_bad == weighted_pred))
fit_test_prob <- predict(Naive_fit, newdata = test, type = "raw")
weighted_pred <- ifelse(fit_test_prob[,2]>10*fit_test_prob[,1], "good", "bad")
with(test, table(good_bad, weighted_pred))
cat("Misclassification Rate:", 1-mean(test$good_bad == weighted_pred))
library(boot)
setwd("E:/Master Documents/Machine Learning/Lab2block1")
state <- read.csv2("state.csv")
orderedstate <- state[order(state$MET,decreasing = FALSE),]
plot(orderedstate$MET,orderedstate$EX,type = "p", col= "red", xlab="MET", ylab = "EX")
set.seed(12345)
reg.fun <-tree(EX~MET, orderedstate, control = tree.control(dim(orderedstate)[1],minsize = 8))
```

```r
cv.reg<-cv.tree(reg.fun)
plot(cv.reg)

opTree<-prune.tree(reg.fun, best = 3 )
plot(opTree)

reg.fit<-predict(opTree,orderedstate)
plot(orderedstate$MET,orderedstate$EX,type = "p",
     col = "blue", xlab="MET",ylab="EX")
lines(orderedstate$MET,reg.fit, col="green")

res<-(orderedstate$EX-reg.fit);

hist(res, col="blue",xlab = "Residuals")
plot(orderedstate$MET , orderedstate$EX, col = "red", type = "p")
lines(orderedstate$MET, reg.fit, col = "green")
set.seed(12345)
bootstraping <- function(dat, id){
  selected <- dat[id, ]
  reg.fun <-tree(EX~MET, selected,
               control = tree.control(dim(selected)[1],minsize = 8))
  opTree <- prune.tree(reg.fun, best=3)
  reg.fit <- predict(opTree, dat)
  return(reg.fit)
}

bstr <- boot(orderedstate, bootstraping, R = 1000)
ci<- envelope(bstr,level = 0.95)
plot(orderedstate$MET,orderedstate$EX,type = "p", xlab="MET",ylab="EX")
lines(orderedstate$MET,reg.fit, col="blue")
par(new= TRUE)
points(orderedstate$MET,ci$point[2,], type="l", col="red")
points(orderedstate$MET,ci$point[1,], type="l", col="red")
points(orderedstate$MET,ci$overall[2,], type="l", col="black")
points(orderedstate$MET,ci$overall[1,], type="l", col="black")

set.seed(12345)

predfun <- function(data, mle) {
dat<-data.frame(EX=data$EX, MET=data$MET)
n=length(data$EX)
dat$EX<-rnorm(n,predict(mle,dat),sd(resid(mle)))
return(dat)
}

reg.pred <- function(dat){
iniTree<-tree(EX~MET, dat, control = tree.control(dim(dat)[1],minsize = 8))
opTree1<-prune.tree(iniTree, best=3)
optPred<-predict(opTree1,dat)
return(optPred)
}

TreeInit<-tree(EX~MET, orderedstate,control = tree.control(dim(state)[1],minsize = 8))
```

```
opTree2<-prune.tree(TreeInit, best=3)

solution<-boot(orderedstate, statistic=reg.pred, R=1000, mle=opTree2,ran.gen=predfun, sim="parametric")

CI<-envelope(solution,level = 0.95)
plot(x = orderedstate$MET , y = orderedstate$EX, type = "p")
lines( x = orderedstate$MET, CI$overall[1,], col = "red", type="l")
lines(x = orderedstate$MET, y = CI$overall[2,], col = "red",  type="l")
lines( x = orderedstate$MET, CI$point[1,], col = "green")
lines(x = orderedstate$MET, y = CI$point[2,], col = "green",  type="l")
points(x = orderedstate$MET, y = reg.fit, col = "blue", type="l")

# 4.1 PCA

setwd("E:/Master Documents/Machine Learning/Lab2block1")
spectra <- read.csv2("NIRspectra.csv")
spectra <- spectra %>% select(-Viscosity)
pca_spectra <- prcomp(x = spectra , scale. = TRUE)
#ggbiplot(pca_spectra, var.axes=FALSE)
pca_prop <- summary(pca_spectra)
pca_prop_solo <- pca_prop$importance[2,] # Proportion explained
pca_prop_cum <- pca_prop$importance[3,] # Cumulative prop explained

df_pca <- data.frame(prop_solo = pca_prop_solo, prop_cum = pca_prop_cum,
                     pca_index = 1:length(pca_prop_solo))
# Minimal number of components explaining at least 99% of total variance
df_pca[1:(sum(pca_prop_cum < 0.99) + 1), ]
df_pcs <- data.frame(PC1 = pca_spectra$x[,1], PC2 = pca_spectra$x[,2])
std_dev <- pca_spectra$sdev
pr_var <- std_dev^2
#pr_var
#proportion of variance explained
prop_var <- pr_var/sum(pr_var)
# 4.2 Loadings
df_loadings <- data.frame(pc1_loading = pca_spectra$rotation[,1],
                          pc2_loading = pca_spectra$rotation[,2],
                          var = rownames(pca_spectra$rotation),
                          var_index = 1:nrow(pca_spectra$rotation))


#plot(pca_spectra$rotation[,1], main="Traceplot, PC1")
#plot(pca_spectra$rotation[,2],main="Traceplot, PC2")

#df_loadings %>%
#  tidyr::gather(key = "pc", value = "loading", -var, -var_index) %>%
#  ggplot(aes(x = var_index, y = loading, color = pc)) +
#  geom_point() +
#  labs(title = "PC loadings for variables") +
#  theme_minimal()

#df_loadings %>%
 # mutate(id = row_number()) %>%
  #tidyr::gather(key = "pc", value = "loading", -var, -var_index, -id) %>%
```

```r
  #dplyr::filter(id > 118 & pc == "pc2_loading") %>%
  #xtable() %>%
  #print(comment=FALSE)
# 4.3 Score plot
set.seed(12345)
spectra <- as.matrix(spectra)
ica_mod <- fastICA(X = spectra ,
                   n.comp = 2)

w_prime <- (ica_mod$K %*% ica_mod$W)

#par(mfrow = c(2,1))
#plot(w_prime[,1], main="Traceplot, ICA1")
#plot(w_prime[,2],main="Traceplot, ICA2")

ica_mod$S %>%
  as.data.frame() %>%
  rename(ic1 = V1, ic2 = V2) %>%
  ggplot(aes(x = ic1, y = ic2)) +
  geom_point() +
  labs(title = "Independent component score plot") +
  theme_minimal()
```