lab2-block1_group5_report

Bjorn Hansen, Erik Anders, Ahmed Alhasan

12/8/2019

```
setwd("/home/erik/Documents/SML/Semester 1/Machine Learning/Lab 2")
library(readxl)
library(tree)
library(e1071)
library(SDMTools)
library(ggrepel)
```

```
## Loading required package: ggplot2
```

```
library(ggplot2)
library(boot)
library(fastICA)
```

Assignment 2. Analysis of credit scoring

```
##       Y hat
## Y      bad good
##   bad   22   43
##   good  44  141
```

```
## [1] "Deviance Tree Error Rate with Test Data: 0.348"
```

```
##       Y hat
## Y      bad good
##   bad   71   77
##   good  49  303
```

```
## [1] "Deviance Tree Error Rate with Train Data: 0.252"
```

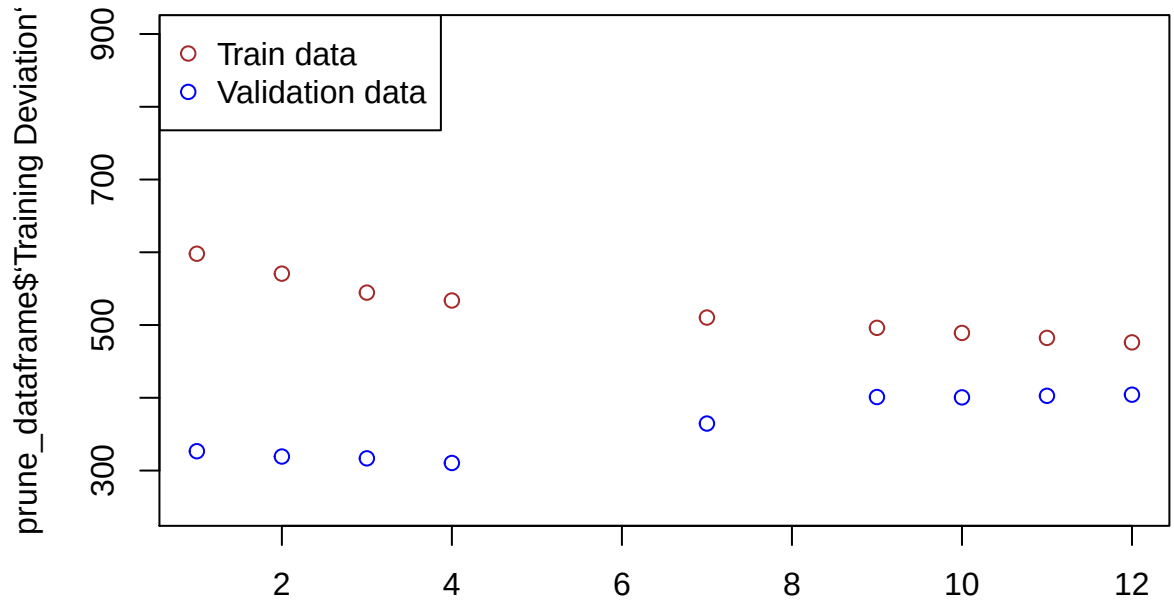
```
##       Y hat
## Y      bad good
##   bad   19   46
##   good  42  143
```

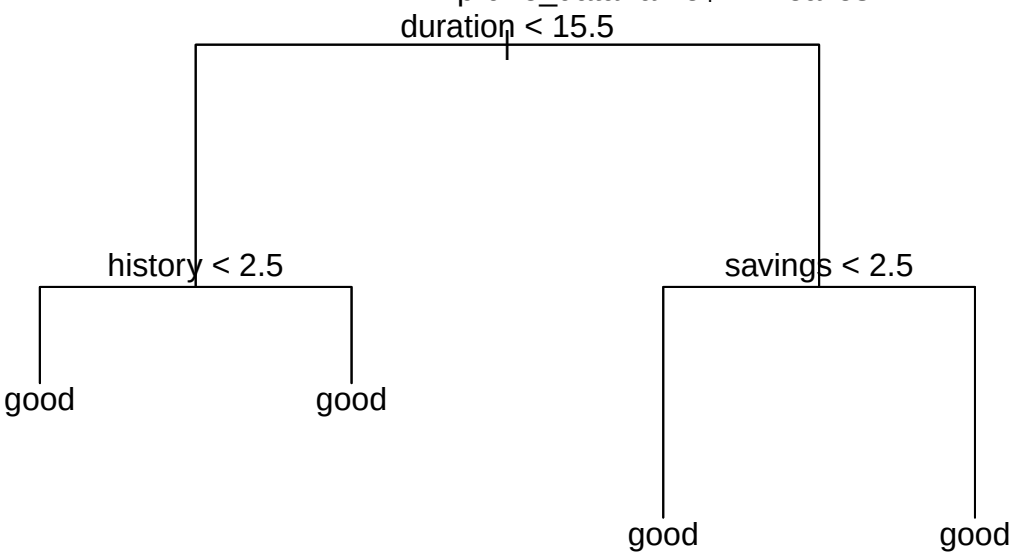
```
## [1] "Gini Tree Error Rate with Test Data: 0.352"
```

```
##       Y hat
## Y      bad good
##   bad   56   92
##   good  35  317
```

```
## [1] "Gini Tree Error Rate with Train Data: 0.254"
```

Above are the results from the deviance tree and gini index models. After running the deviance and gini models a few times it seems that both have similar results, but the deviance index may be a slightly better predicter for the data.





The lowest deviance seems to be when nr leaves = 4 for the validation data accoding to the plot above. As seen from the tree graph above the deviance model with four leaves is using the following variables: duration, history, savings.

```
##       Y hat
## Y      bad good
##   bad   42   23
##   good  71  114
```

```
## [1] "naivebayes Error Rate with Test Data: 0.376"
```

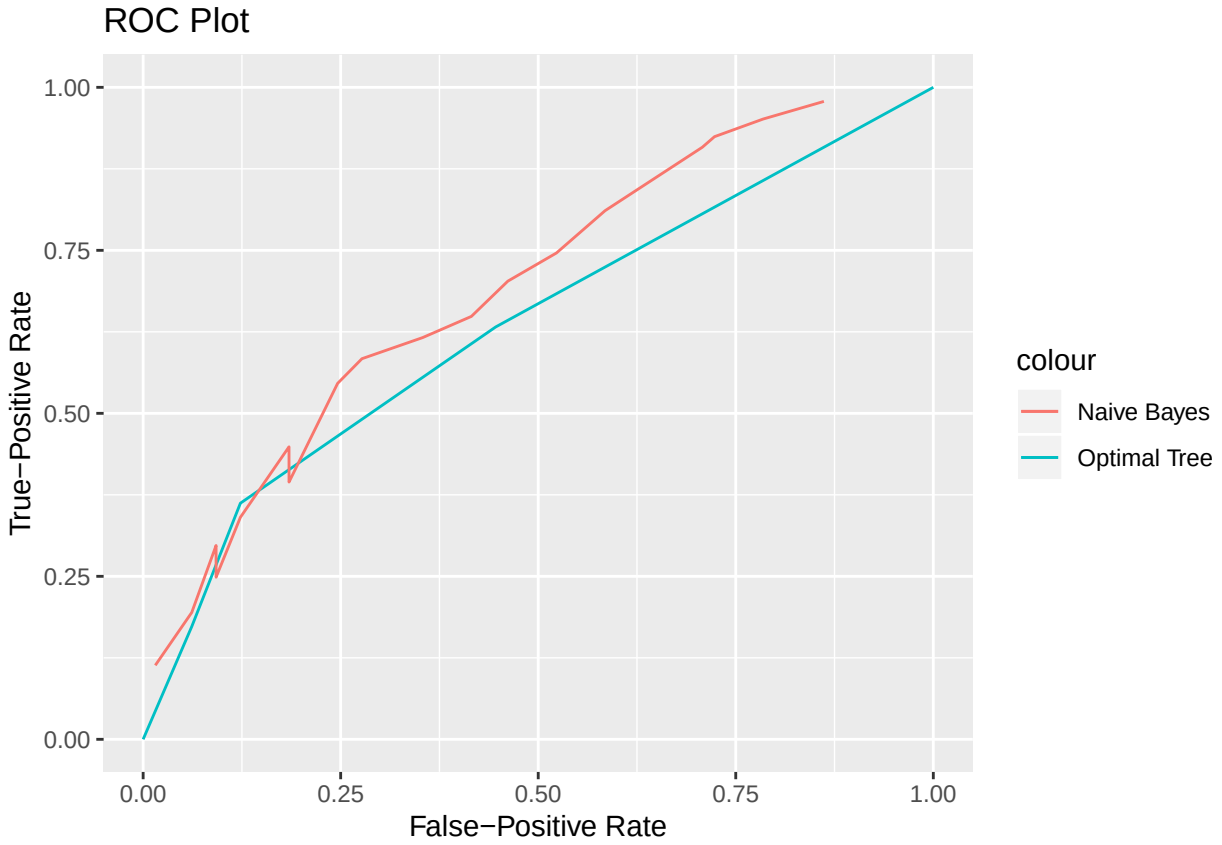
```
##       Y hat
## Y      bad good
##   bad   42   23
##   good  71  114
```

```
## [1] "naivebayes Error Rate with Train Data: 0.312"
```

The results above indicat that the Naive Bayes model performs similarly well compared to the deviance tree

and gini index models, but has a bit more error than both.

Your tree just bends two times, but you have 4 leaves. That's odd!



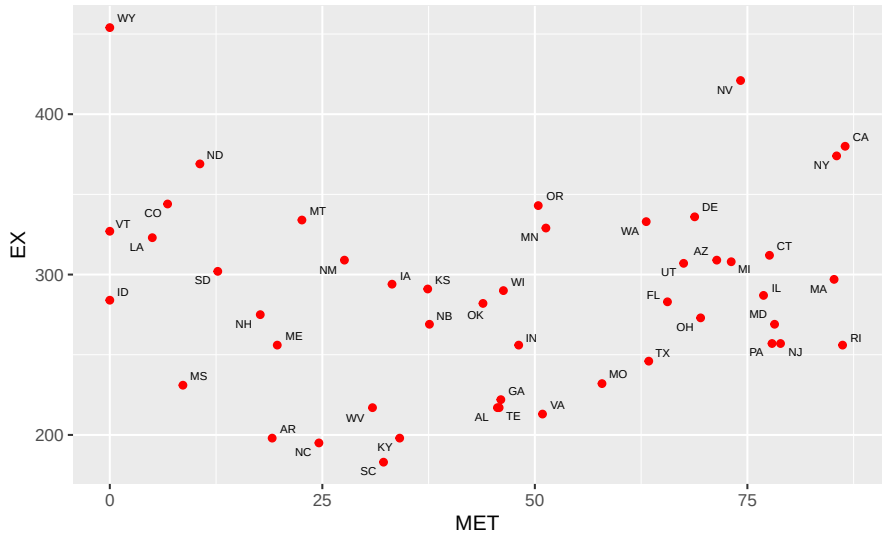
According to the ROC plot, the Naive Bayes performs better due to its larger AUC, this could be attributed to all thresholds being used. The percentage of FPR to TPR in Naive Bayes is smaller than the percentage of the optimal tree.

```
## [1] "Results using Training data:"

## $`Confusion Matrix`
##    Y hat
## Y     0   1
##   0 145   3
##   1 286  66
##
## $`Error Rate`
## [1] 0.578

## [1] "Results using Testing data:"

## $`Confusion Matrix`
##    Y hat
## Y     0   1
##   0  62   3
##   1 150  35
##
## $`Error Rate`
## [1] 0.612
```

As can be seen from the results using training and test data above, When the loss matrix was applied the FPR decreased, but the error rate has gotten worse.

Assignment 3. Uncertainty estimation

3.1 Appropriate Model

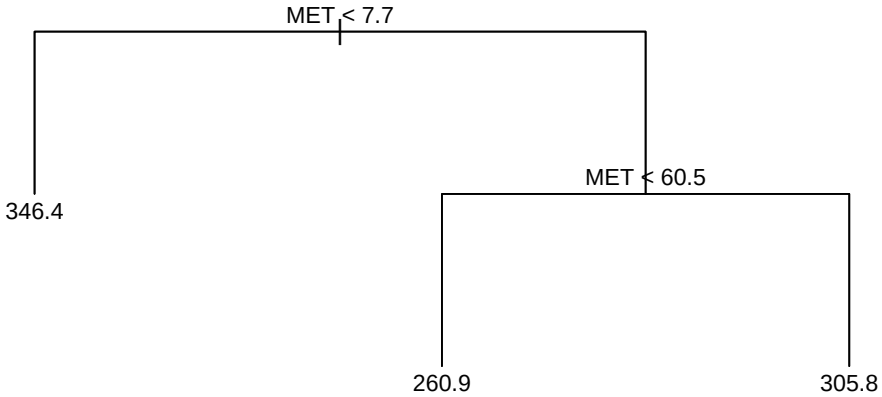


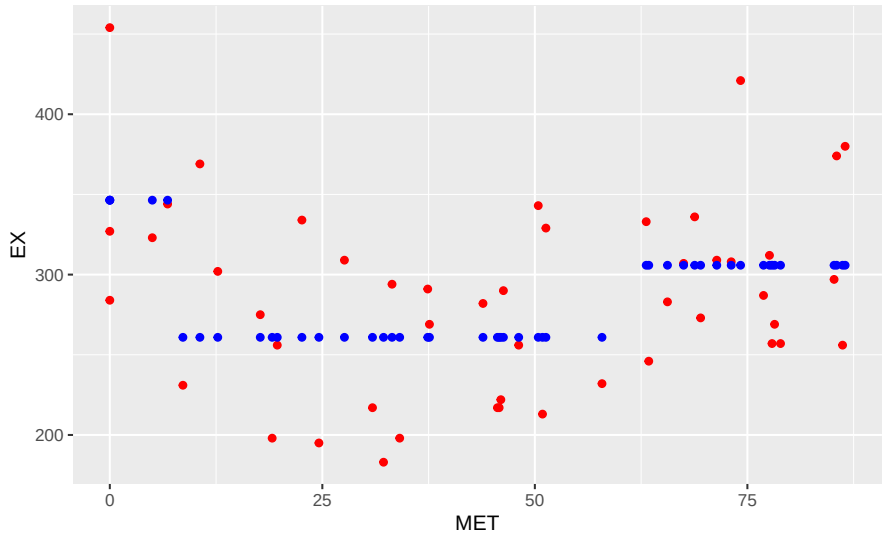
- Because it's hard to select any model without overfitting and we don't know the distribution of the data, a combination of ploynomial model or splines with bootstrap would be good option.

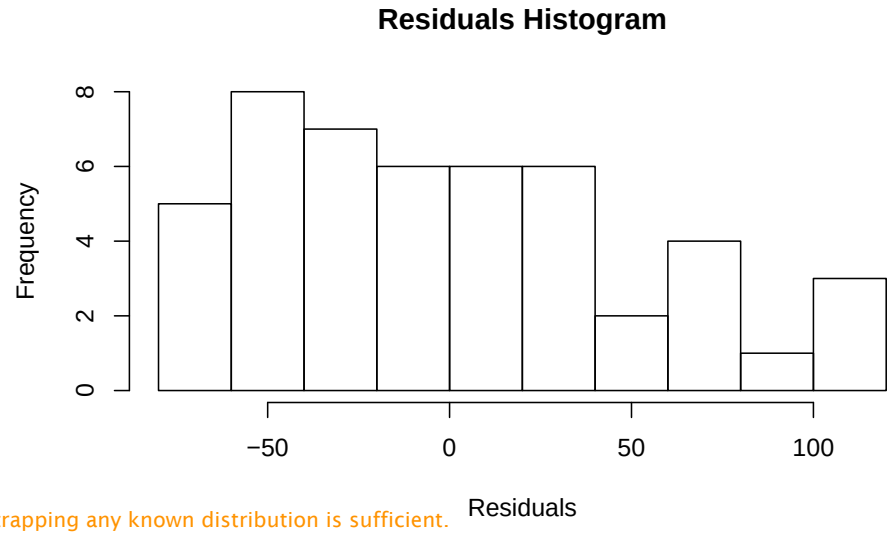
Why splines?

3.2 Tree Model

```
##
## Regression tree:
## snip.tree(tree = tree_model, nodes = 7:6)
## Number of terminal nodes:  3
## Residual mean deviance:  2698 = 121400 / 45
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -77.88  -43.88   -4.88    0.00   30.13  115.20
```



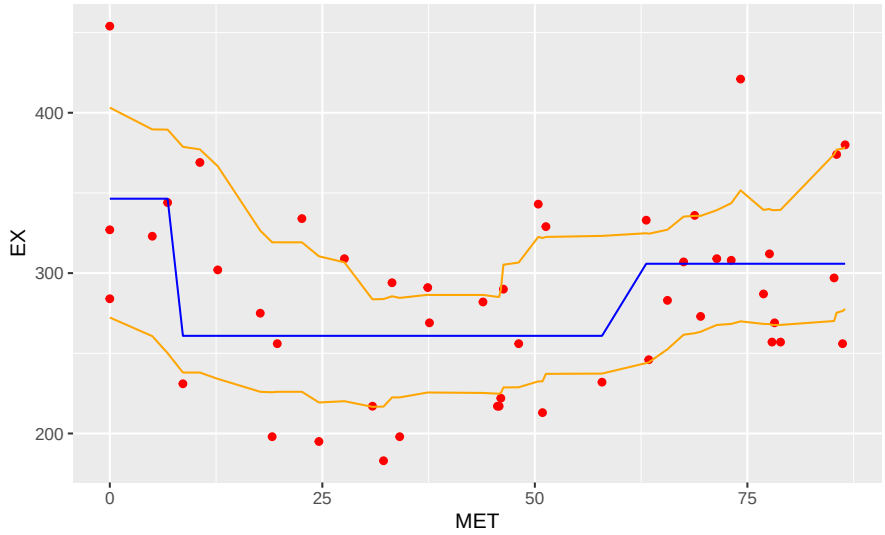




For parametric bootstrapping any known distribution is sufficient.

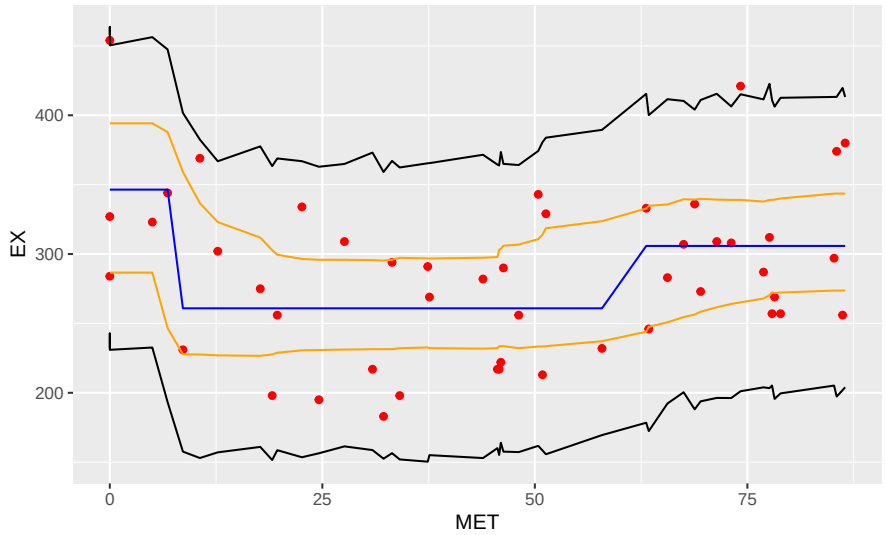
- An ideal representation of the residuals in the histogram should take a Normal distribution where most of the ressiduals are close to zero and few in the positive and the negative side, however in this case it is not and there are many predictions with high residuals on the sides, also on the positive side there are more outliers (have high residuals)
- The tree model fit very poorly to the data, there are many outliers on both sides of the three terminal nodes.

3.3 Nonparametric Bootstrap



- The confidence interval is bumby because of the variance of point estimates as we are getting large and varied residuals every time take a bootstrap sample.
- Because the confidence interval is rather large it confirms the belief that the tree model is a poor fit for this data set and not reliable.

3.4 Parametric Bootstrab



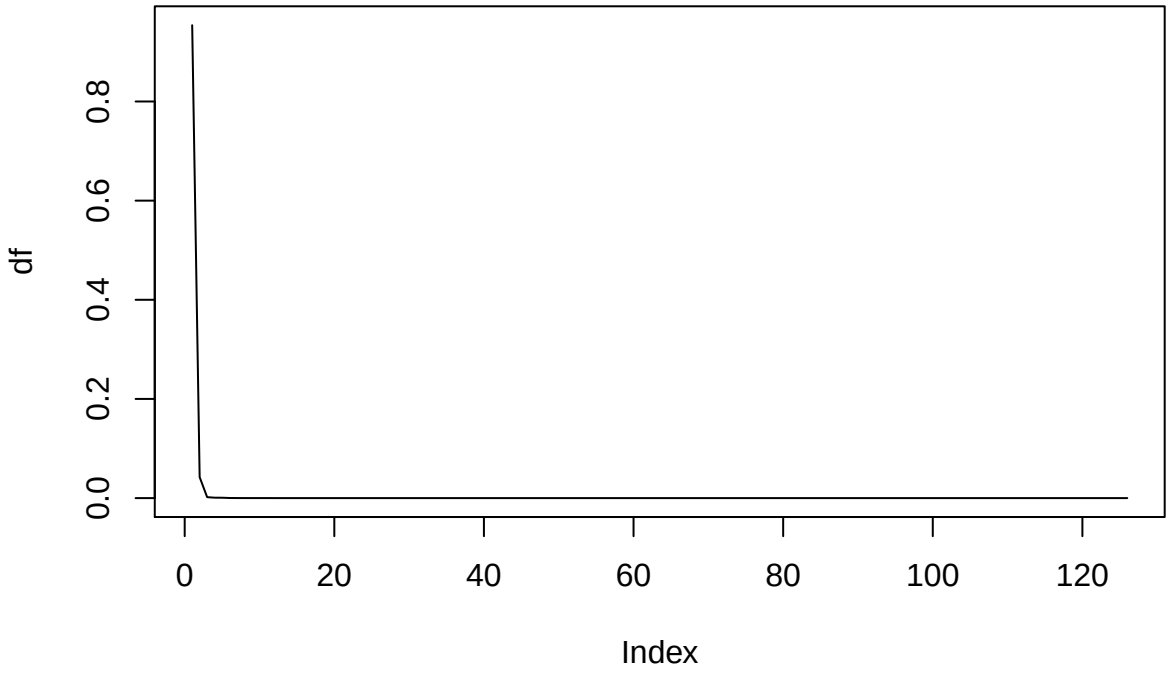
- The confidence interval is smoother and narrawer with parametric bootsrap which make the tree model a slightly more reliable, however the confidence interval still wide enough allowing wide range of the prediction means to fit in.
- Only 2 points out of 48 are outside the prediction interval which is around 5%, and that's how it should be because we acounted for the error rate(the residuals) in the prediction interval, meaning 95% the future sampled EX values should be withen this interval.

3.5 Optimal Bootstrap

- The Parametric Bootstrap is more optimal because it gives less variance in the confidence interval than Nonparametric Bootstrap even with high residuals.

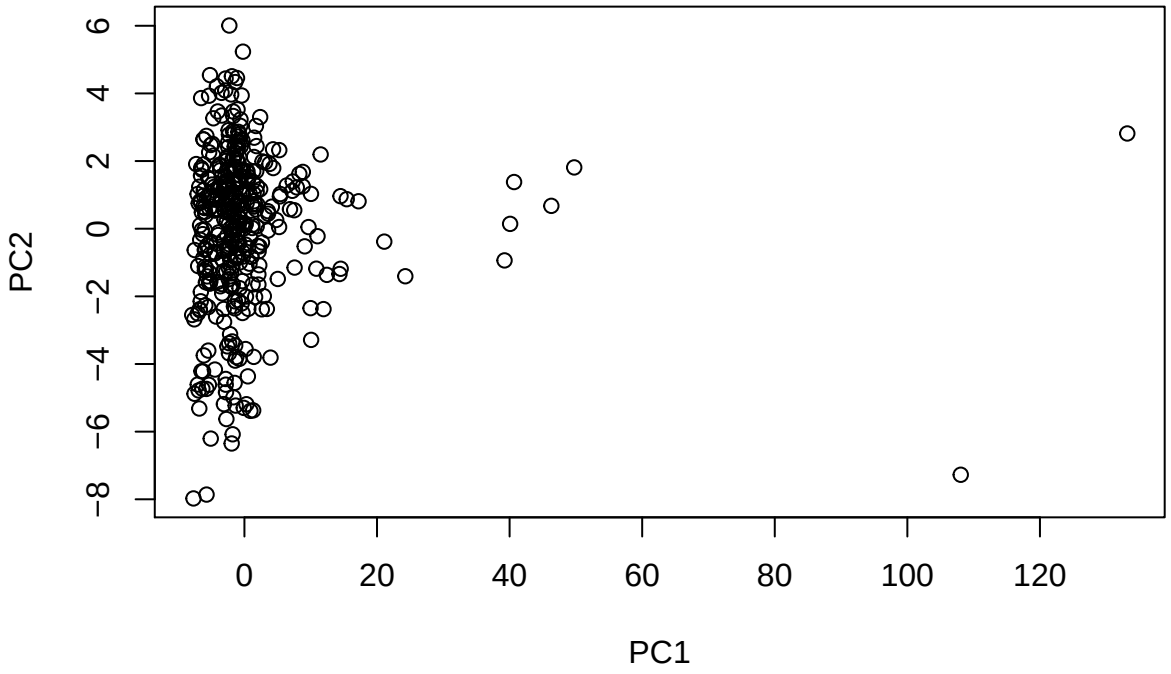
  That is not correct. Assuming a bad model that has a really low variance for the confident bands is not necessarily better, as it highly overestimates itself. You have to look at the residuals and conclude if it is a know distribution and if your sample size is large enough to generate this distribution in a robust way.

Assignment 4. Principal components Seems fine!

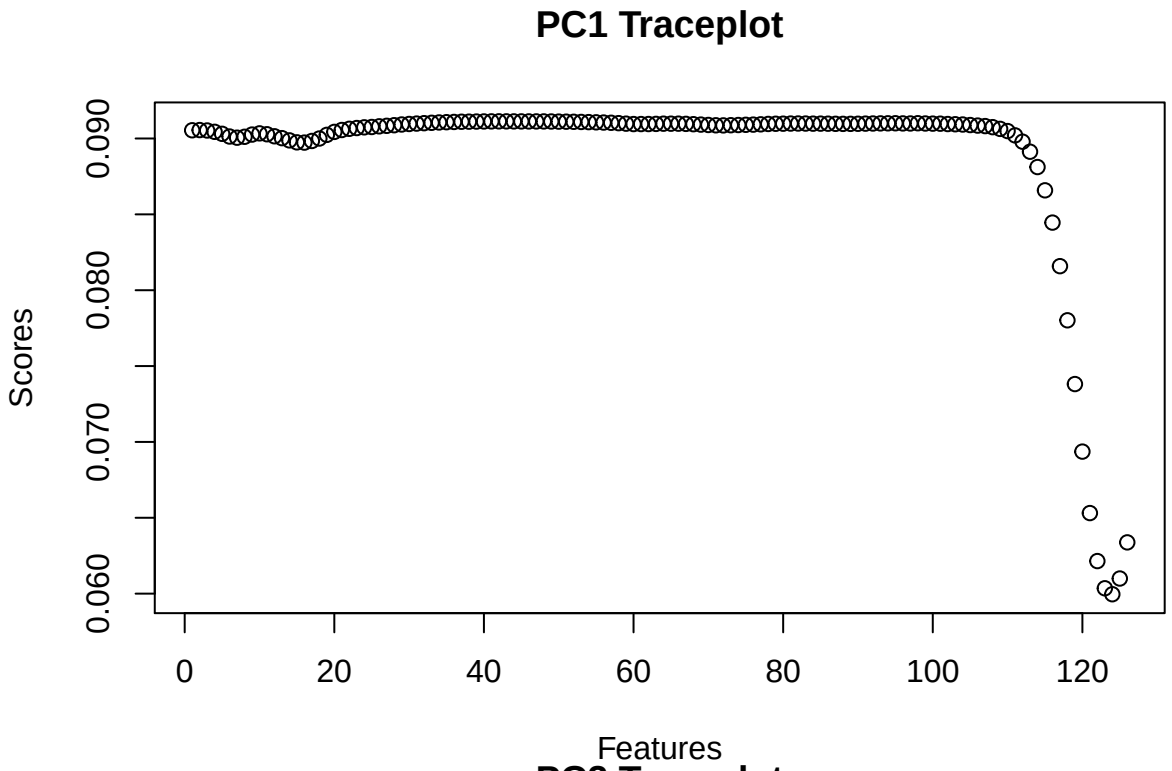


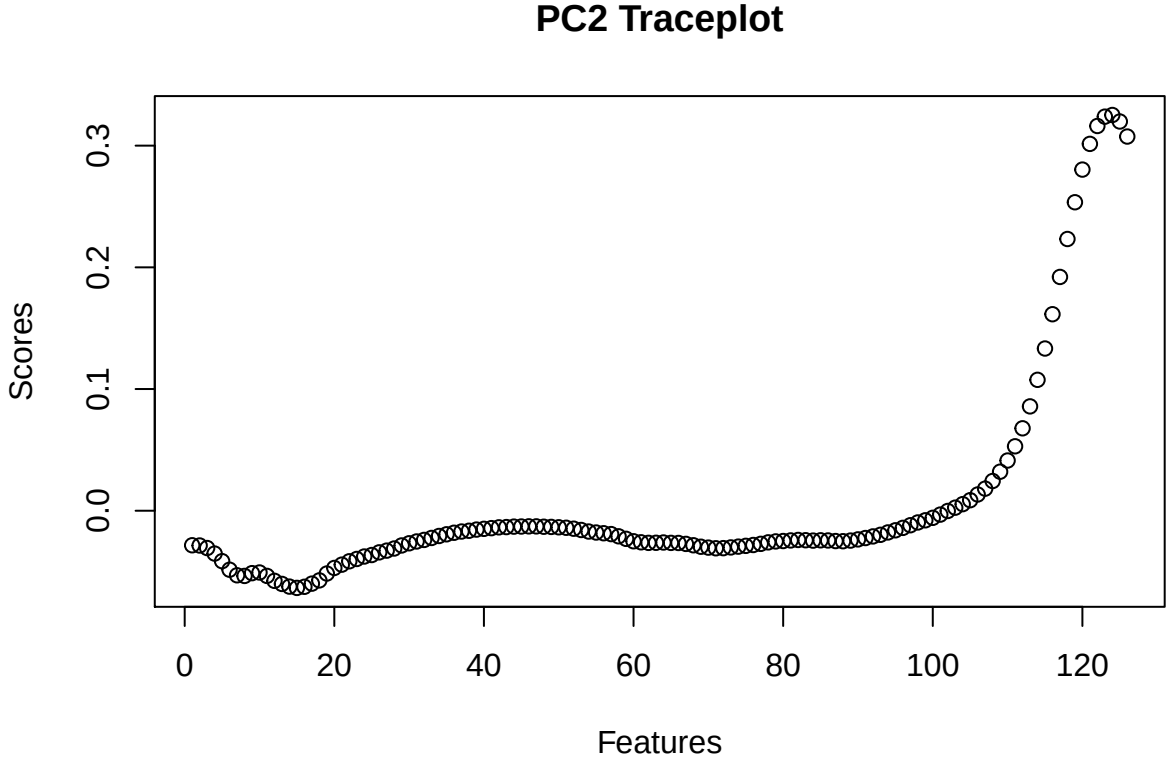

```
## [1] 0.9961398
## [1] 2
```

As is shown in the plot and the console output. Using the first two PCs we can already explain 99.6% of the variation.



Yes, there are two variables which have very extreme values for PC1. One of them is also extrem for PC2(~-7). There is also a group of 5 variables with values for PC1 from 40-60. All the other values for PC1 seem to be smaller than 30. PC2 seems almost normally distributed, with small outliers of 2 variables in both directions(5-6 and ~-8).



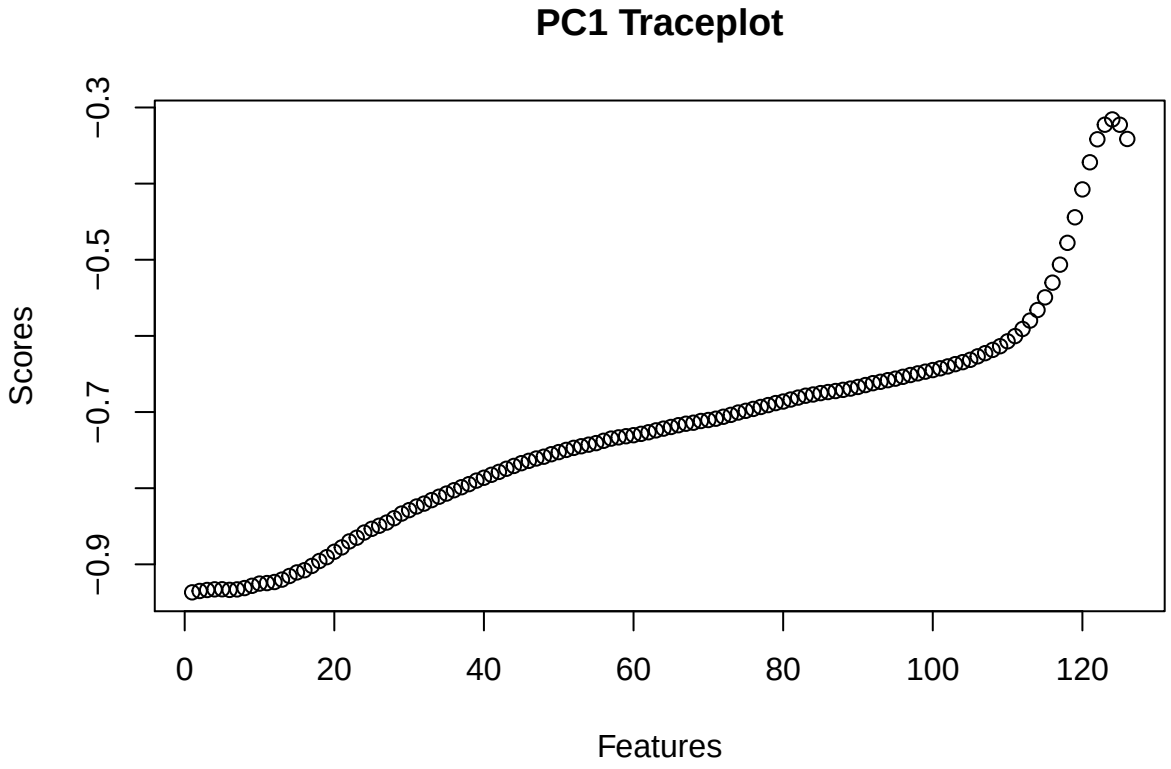


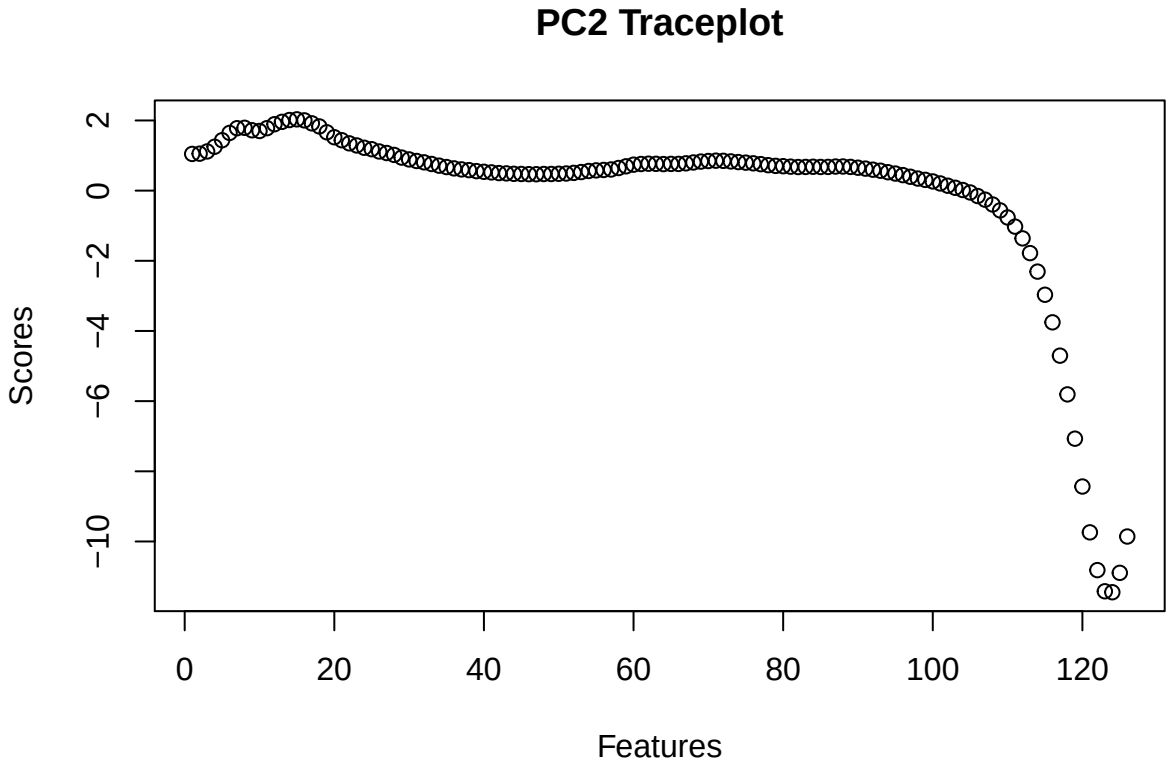
Yes, PC2 only shows high scores for a bunch of original features (nr. 110+). This means that PC2 is only explained by few features.

```
## Centering
```

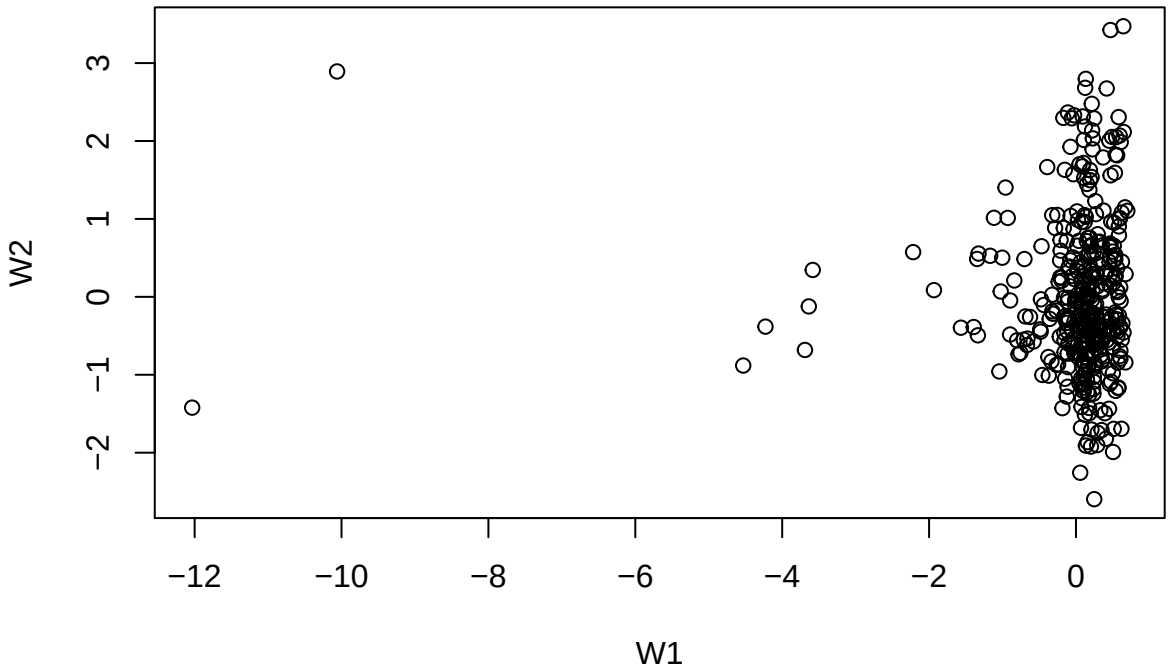
```
## Whitening
```

```
## 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
```





It appears as if the plots for PC1 and PC2 are switched for W' compared to 2.



W is the un-mixing matrix that maximizes the non-gaussianity of the components so we can extract the independent components.

The ICA plot basically takes the standarized PCA factors after scaling and whitening "projecting the data onto its principal component directions" and rotate them to maximmize the non-gaussianity of the two components, thats why we see the ICA plot as a rotated PCA plot.

Apendix

```
library(readxl)
library(tree)
library(e1071)
library(SDMTools)
library(ggrepel)
library(ggplot2)
library(boot)
library(fastICA)

### Assignment 2. Analysis of credit scoring ###

credit_data = read_excel("creditscoring.xls")
credit_data$good_bad = as.factor(credit_data$good_bad)

## 1
# Split data into training/validation/test as 50/25/25.

n=dim(credit_data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
id2=sample((1:1000)[-id],floor(n*0.25))
train=credit_data[id,]
validation=credit_data[id2,]
test=credit_data[-c(id,id2),]

## 2
# Produce the models

deviance_model = tree(good_bad ~., data=train, split="deviance")
gini_model = tree(good_bad ~., data=train, split="gini")

# Predictions/ errors for deviance tree
# Test Error
deviance_fitted.results_test = predict(deviance_model, newdata=test, type="class")
deviance_misClasificError_test = mean(deviance_fitted.results_test != test$good_bad)
deviance_confmat_test = table("Y"=test$good_bad,"Y hat"=deviance_fitted.results_test)
deviance_confmat_test
print(paste("Deviance Tree Error Rate with Test Data:", 1-sum(diag(deviance_confmat_test)) / sum(devian
# Train Error
deviance_fitted.results_train = predict(deviance_model, newdata=train, type="class")
deviance_misClasificError_train = mean(deviance_fitted.results_train != train$good_bad)
deviance_confmat_train = table("Y"=train$good_bad,"Y hat"=deviance_fitted.results_train)
deviance_confmat_train
print(paste("Deviance Tree Error Rate with Train Data:", 1-sum(diag(deviance_confmat_train)) / sum(devi

# Predictions/ errors for gini tree
# Test Error
gini_fitted.results_test = predict(gini_model, newdata=test, type="class")
gini_misClasificError_test = mean(gini_fitted.results_test != test$good_bad)
gini_confmat_test = table("Y"=test$good_bad,"Y hat"=gini_fitted.results_test)
gini_confmat_test
```

```r
print(paste("Gini Tree Error Rate with Test Data:", 1-sum(diag(gini_confmat_test)) / sum(gini_confmat_t

# Train Error
gini_fitted.results_train = predict(gini_model, newdata=train, type="class")
gini_misClasificError_train = mean(gini_fitted.results_train != train$good_bad)
gini_confmat_train = table("Y"=train$good_bad,"Y hat"=gini_fitted.results_train)
gini_confmat_train
print(paste("Gini Tree Error Rate with Train Data:", 1-sum(diag(gini_confmat_train)) / sum(gini_confmat

# After running the deviance and gini models a few times it seems that both have similar
# results, but the deviance index may be a slightly better predicter for the data.

## 3

prune_train = prune.tree(deviance_model, method = "deviance")
prune_validation = prune.tree(deviance_model, newdata = validation, method = "deviance")

prune_dataframe = data.frame(Length = prune_train$size, Train = prune_train$dev, Validation = prune_vali
colnames(prune_dataframe) = c("Nr Leaves", "Training Deviation", "Validation deviation")

plot(prune_dataframe$`Nr Leaves`, prune_dataframe$`Training Deviation`, col="brown", ylim = c(250,900))
points(prune_dataframe$`Nr Leaves`, prune_dataframe$`Validation deviation`, col="blue")
legend("topleft", c("Train data","Validation data"),pch=c(1,1),col = c("brown","blue"))
# Lowest deviance seems to be when nr leaves = 4 for the validation data and about 12
# for the training data.

# Plot the variables used in model
best_tree = prune.tree(deviance_model, best = 4, method="deviance")
plot(best_tree)
text(best_tree, pretty = 0)
# The deviance model with four leaves is using variables: duration, history, savings.


## 4
naivebayes_model = naiveBayes(good_bad ~., data=train)

# Predictions/ errors for Naive Bayes model
# Test Error
naivebayes_fitted.results_test = predict(naivebayes_model, newdata=test, type="class")
naivebayes_confmat_test = table("Y"=test$good_bad,"Y hat"=naivebayes_fitted.results_test)
naivebayes_confmat_test
print(paste("naivebayes Error Rate with Test Data:", 1-sum(diag(naivebayes_confmat_test)) / sum(naiveba

# Train Error
naivebayes_fitted.results_train = predict(naivebayes_model, newdata=train, type="class")
naivebayes_confmat_train = table("Y"=train$good_bad,"Y hat"=naivebayes_fitted.results_train)
naivebayes_confmat_test
print(paste("naivebayes Error Rate with Train Data:", 1-sum(diag(naivebayes_confmat_train)) / sum(naivel
# It looks like the Naive Bayes model performs similarly well compared to the deviance
# tree and gini index models, but has a bit more error than both.

## 5
```

```r
# TPR and FPR for optimal tree
Pi = seq(0.05, 0.95, 0.05)

best_tree_fit = predict(best_tree, newdata = test)
tree_good = best_tree_fit[,2]
true_assign = ifelse(test$good_bad == "good", 1, 0)

tree_TPR_FPR = matrix(nrow = 2, ncol = length(Pi))
rownames(tree_TPR_FPR) = c("TPR", "FPR")



for (i in 1:length(Pi)){
  tree_assign = ifelse(tree_good > Pi[i], 1, 0)
  tree_confmat = confusion.matrix(tree_assign, true_assign)

  tpr1 = tree_confmat[2,2]/(tree_confmat[2,1] + tree_confmat[2,2])
  fpr1 = tree_confmat[1,2]/(tree_confmat[1,1] + tree_confmat[1,2])

  tree_TPR_FPR[,i] <- c(tpr1,fpr1)
}

# TPR and FPR for naive bayes
bayes_model = naiveBayes(good_bad ~ ., data = train)
bayes_fit = predict(bayes_model, newdata = test, type = "raw")
bayes_good = bayes_fit[,2]

bayes_TPR_FPR = matrix(nrow = 2, ncol = length(Pi))
rownames(bayes_TPR_FPR) = c("TPR", "FPR")


for (i in 1:length(Pi)) {
  bayes_assign = ifelse(bayes_good > Pi[i], 1, 0)
  bayes_confmat = confusion.matrix(bayes_assign, true_assign)

  tpr2 = bayes_confmat[2,2]/(bayes_confmat[2,1] + bayes_confmat[2,2])
  fpr2 = bayes_confmat[1,2]/(bayes_confmat[1,1] + bayes_confmat[1,2])

  bayes_TPR_FPR[,i] = c(tpr2,fpr2)
}

# ROC optimal Tree Naive Bayes plot

ggplot() +
  geom_line(aes(x = tree_TPR_FPR[2,], y = tree_TPR_FPR[1,], col = "Optimal Tree")) +
  geom_line(aes(x = bayes_TPR_FPR[2,], y = bayes_TPR_FPR[1,], col = "Naive Bayes")) +
  xlab("False-Positive Rate") +
  ylab("True-Positive Rate") +
  ggtitle("ROC")
# According to the ROC plot the Naive Bayes performs better due to its larger AUC,
# this could be attributed to all thresholds being used.
# The percentage of FPR to TPR in Naive Bayes is smaller than its percentage in the
# optimal tree.
```

```r
## 6

loss_mat = matrix(c(0,10,1,0), nrow = 2)

loss_function = function(data,loss_mat){
  prob = ifelse(data$good_bad == "good",1,0)

  naivebayes_model = naiveBayes(good_bad ~.,data=train)
  naivebayes_fit = predict(naivebayes_model, newdata = data, type = "raw")

  naivebayes_fit = ifelse(loss_mat[1,2] * naivebayes_fit[,2] > loss_mat[2,1] * naivebayes_fit[,1],1,0)

  conf_mat = table("Y" = prob, "Y hat" = naivebayes_fit)
  miss_rate = 1-sum(diag(conf_mat))/sum(conf_mat)

  result = list("Confusion Matrix" = conf_mat, "Error Rate" = miss_rate)
  return(result)
}
print("Training data:")
loss_function(train,loss_mat)
print("Testing data:")
loss_function(test,loss_mat)
# When the loss matrix was applied the FPR decreased, but the error rate has gotten worse.

### Assignment 3. Uncertainty estimation ###

state <- read.csv2("Data/State.csv", header = TRUE)
state <- state[order(state$MET),]

ggplot(data = as.data.frame(state), aes(y = state[,1], x = state[,3]))+
  xlab("MET") + ylab("EX")+
  geom_text_repel(label = state[,8], size = 2)+
  geom_point(color = 'red')

tree_model <- tree(EX ~ MET,
                   data = state,
                   control = tree.control(nobs = nrow(state),
                                          minsize = 8))
set.seed(12345)
best_tree1 <- cv.tree(tree_model)
best_tree2 <- prune.tree(tree_model, best = 3)
summary(best_tree2)

plot(best_tree2)
text(best_tree2, pretty=1,
     cex = 0.8,
     xpd = TRUE)

tree_pred <- predict(best_tree2, newdata = state)

ggplot(data = as.data.frame(state),
       aes(y = state[,1], x = state[,3])) +
  xlab("MET") +
```

```
  ylab("EX") +
  geom_point(col = "red") +
  geom_point(x = state$MET, y = tree_pred, col = "blue")

hist(residuals(best_tree2),
     main = "Residuals Histogram",
     xlab = "Residuals")

f <- function(data, ind){
  set.seed(12345)
  sample  <- state[ind,]
  my_tree <- tree(EX ~ MET,
                  data = sample,
                  control = tree.control(nobs = nrow(sample), minsize = 8))

  pruned_tree <- prune.tree(my_tree, best = 3)

  pred    <- predict(pruned_tree, newdata = state)
  return(pred)
}

res  <- boot(state, f, R=1000)

conf <- envelope(res, level=0.95)

ggplot(data = as.data.frame(state),
       aes(y = state[,1], x = state[,3])) +
  xlab("MET") +
  ylab("EX") +
  geom_point(col = "red") +
  geom_line(aes(x = state$MET, y = tree_pred), col = "blue") +
  geom_line(aes(x = state$MET, y = conf$point[1,]), col = "orange") +
  geom_line(aes(x = state$MET, y = conf$point[2,]), col = "orange")

mle <- best_tree2

rng <- function(data, mle){
  data1    <- data.frame(EX = data$EX, MET = data$MET)
  n        <- length(data1$EX)
  pred     <- predict(mle, newdata = state)
  residual <- data1$EX - pred
  data1$EX <- rnorm(n, pred, sd(residual))
  return(data1)
}

f1 <- function(data){
  res      <- tree(EX ~ MET,
                   data = data,
                   control = tree.control(nobs=nrow(state),minsize = 8))
  opt_res  <- prune.tree(res, best = 3)
  return(predict(opt_res, newdata = data))
}
```

```r
f2 <- function(data){
  res      <- tree(EX ~ MET,
                   data = data,
                   control = tree.control(nobs=nrow(state),minsize = 8))
  opt_res  <- prune.tree(res, best = 3)
  n        <- length(state$EX)
  opt_pred <- predict(opt_res, newdata = state)
  pred     <- rnorm(n,opt_pred, sd(residuals(mle)))
  return(pred)
}
set.seed(12345)
par_boot_conf <- boot(state, statistic = f1, R = 1000, mle = mle, ran.gen = rng, sim = "parametric")
conf_interval <- envelope(par_boot_conf, level=0.95)

set.seed(12345)
par_boot_pred <- boot(state, statistic = f2, R = 1000, mle = mle, ran.gen = rng, sim = "parametric")
pred_interval <- envelope(par_boot_pred, level=0.95)


ggplot(data = as.data.frame(state),
       aes(y = state[,1], x = state[,3])) +
  xlab("MET") +
  ylab("EX") +
  geom_point(col = "red") +
  geom_line(aes(x = state$MET, y = tree_pred), col = "blue") +
  geom_line(aes(x = state$MET, y = conf_interval$point[1,]), col = "orange") +
  geom_line(aes(x = state$MET, y = conf_interval$point[2,]), col = "orange") +
  geom_line(aes(x = state$MET, y = pred_interval$point[1,]), col = "black") +
  geom_line(aes(x = state$MET, y = pred_interval$point[2,]), col = "black")



### Assignment 4. Principal components ###

#import data
setwd("/home/erik/Documents/SML/Semester 1/Machine Learning/Lab 2")
r_data<-read.csv2("NIRSpectra.csv")
data<-r_data[,-which(names(r_data)=="Viscosity")]

#1.
pca<-prcomp(data, center=TRUE, scale. = TRUE)

df<-c()
for (i in 1:length(pca$sdev)) {
  df[i]<-pca$sdev[i]^2/sum(pca$sdev^2)
}
plot(df, type = "l")

vsum<-0
for (i in 1:length(pca$sdev)) {
  vsum<-vsum+pca$sdev[i]^2/sum(pca$sdev^2)
  if (vsum>0.99) {
    print(vsum)
```

```
    print(i)
    break()
  }
}

plot(pca$x[,"PC1"],pca$x[,"PC2"], xlab = "PC1", ylab = "PC2")

#2.
plot(pca$rotation[,1], main="PC1 Traceplot", xlab = "Features", ylab = "Scores")
plot(pca$rotation[,2], main="PC2 Traceplot", xlab = "Features", ylab = "Scores")


#3.
library(fastICA)
set.seed(12345)
mdata<-as.matrix(data)

ica<-fastICA(mdata, 2, fun = "logcosh", verbose = TRUE)

W_<-ica$K %*% ica$W

plot(W_[,1], main="PC1 Traceplot", xlab = "Features", ylab = "Scores")
plot(W_[,2], main="PC2 Traceplot", xlab = "Features", ylab = "Scores")

plot(ica$S[,1],ica$S[,2], xlab = "W1", ylab = "W2")
```