Lab2ML

Omkar Bhutra

7 December 2018

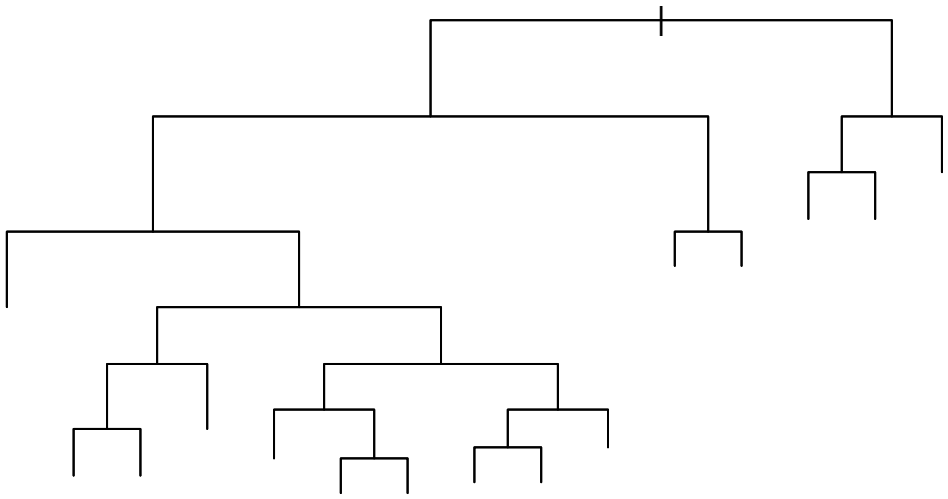
```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

## Loading required package: ggplot2

##
## Attaching package: 'plotly'

## The following object is masked from 'package:ggplot2':
##
##     last_plot

## The following object is masked from 'package:stats':
##
##     filter

## The following object is masked from 'package:graphics':
##
##     layout
```

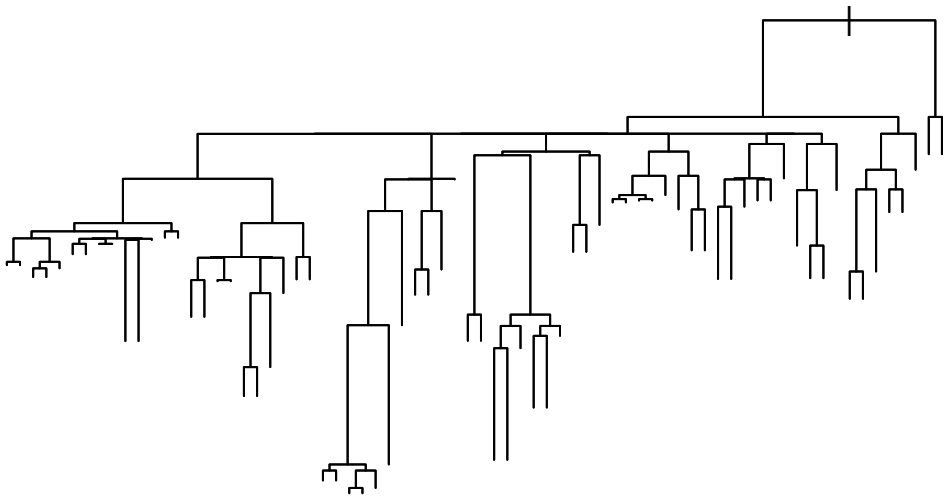
Assignment 2

Analysis of credit scoring

```
##
## Classification tree:
## tree(formula = as.factor(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
```

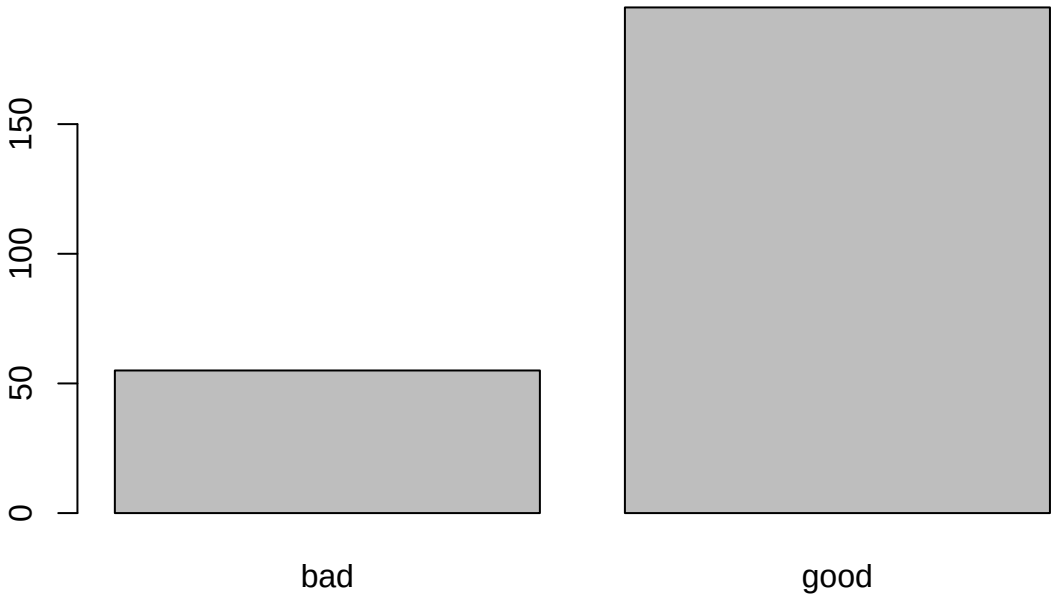
```
##
## Classification tree:
## tree(formula = as.factor(good_bad) ~ ., data = train, split = "gini")
## Variables actually used in tree construction:
##  [1] "foreign"  "coapp"    "depends"  "telephon" "existcr"  "savings"
##  [7] "history"  "property" "marital"  "duration" "employed" "age"
## [13] "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
```

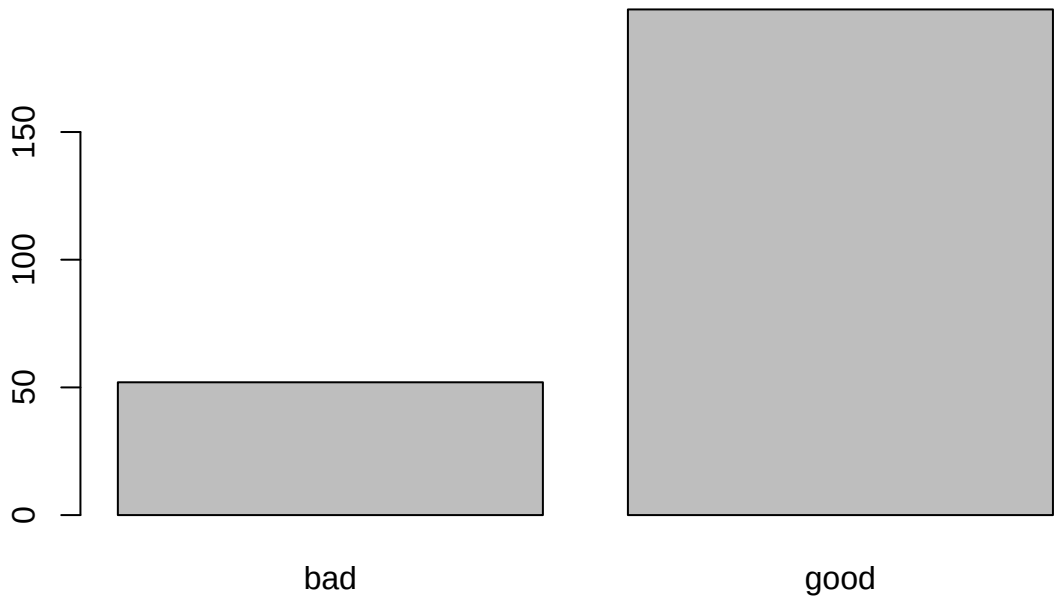




```
## [1] 0.212
```

```
## [1] 0.23
```

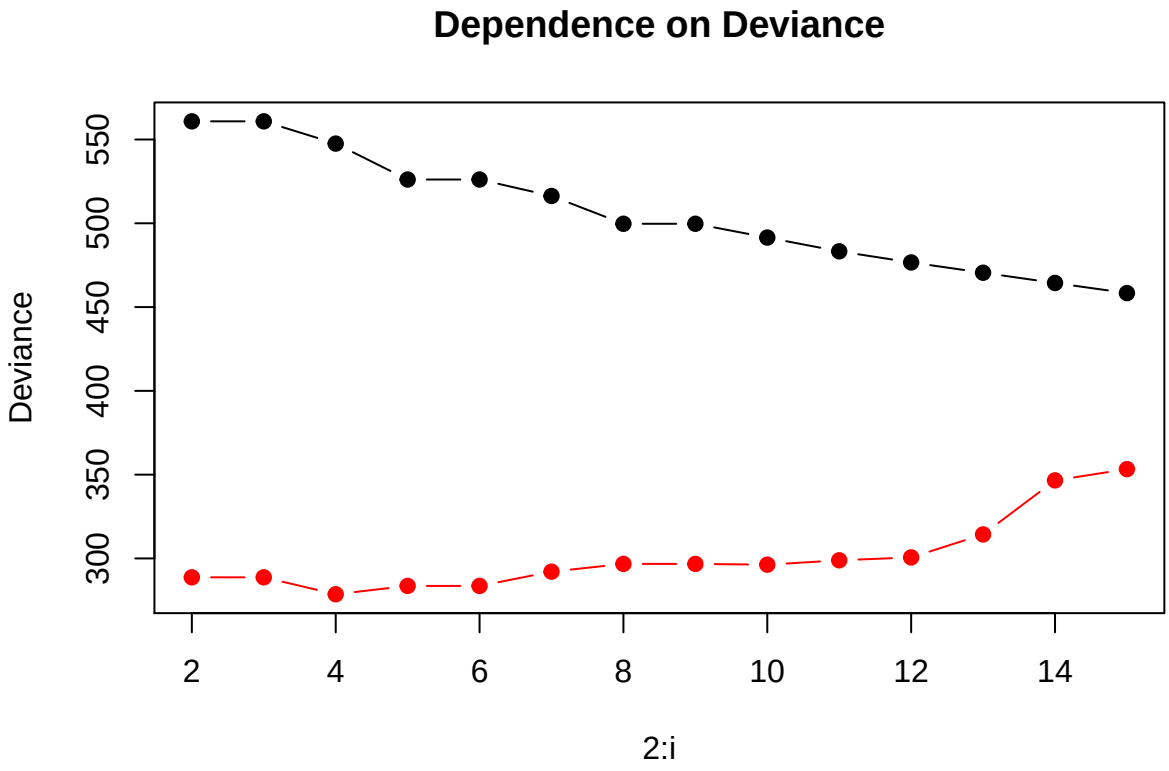

0
50
100
150
bad
good




```
## [1] 0.248
```

```
## [1] 0.356
```

The Misclassification rate for training dataset is 21.2% and 23% for deviance and gini respectively.
The Misclassification rate for testing dataset is 24.8% and 35.6% for deviance and gini respectively.
Better results are obtained with deviance as the misclassification rate is lower than that for gini.



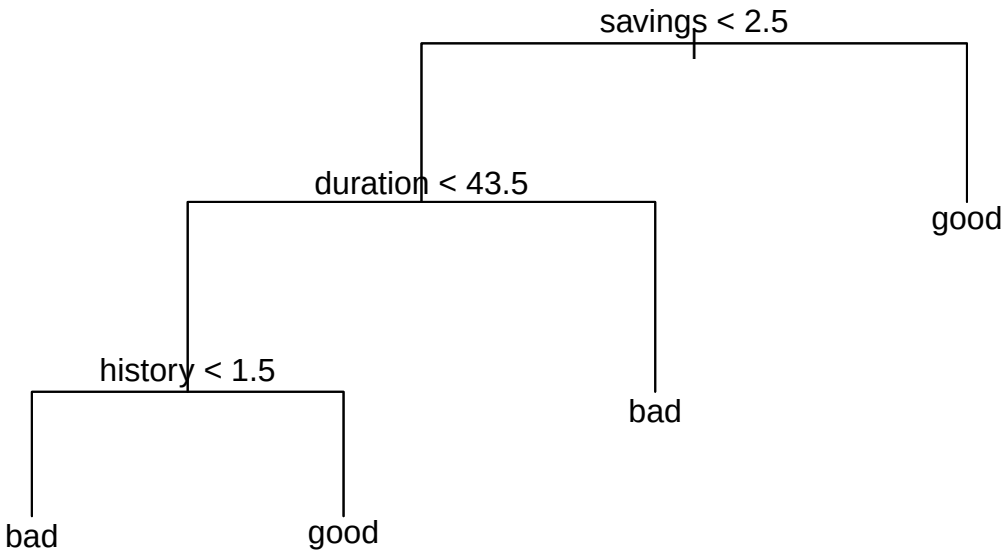

```
##
## Classification tree:
## snip.tree(tree = treestep1, nodes = c(5L, 3L, 9L))
## Variables actually used in tree construction:
## [1] "savings"  "duration" "history"
## Number of terminal nodes:  4
## Residual mean deviance:  1.117 = 547.5 / 490
## Misclassification error rate: 0.251 = 124 / 494
```

```
## [1] "Confusion Matrix"
```

```
##       fit
##        bad good
##   bad   22   53
##   good  12  163
```

```
## [1] "Misclassification rate"
```

```
## [1] 0.26
```



The misclassification rate is reported to be 26% with 22 true negatives and 163 true positives. The classification is done to find good customers that may pay back loans on time. The deviance vs the tree depth is plotted in the given figure. The line for training is shown in red vs the line for validation is shown in black. The optimal tree depth i.e the lowest deviance is present at the tree depth of 4. The optimal tree is shown in the figure. The savings lesser than 2.5 is considered bad, duration lesser than 43.5 is considered good and history lesser than 1.5 is considered bad.

```
##       nbtest
##        bad good
##   bad   50   25
##   good  61  114
```

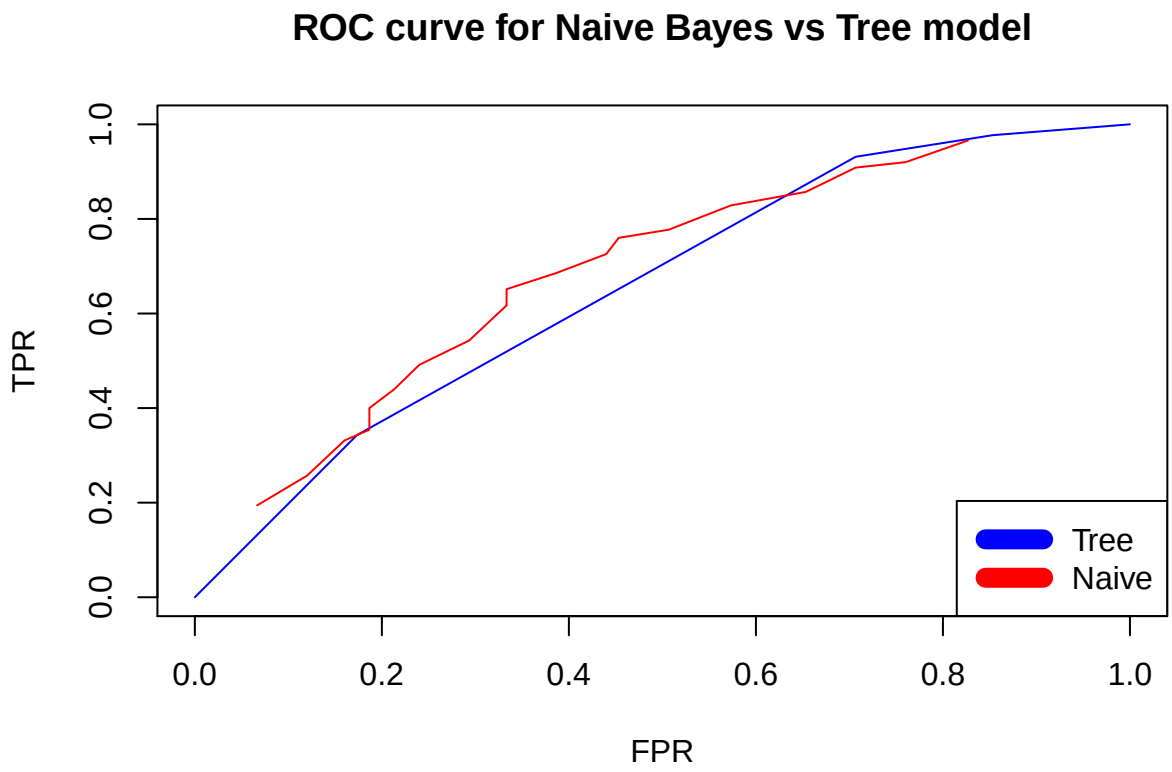
```
##       nbtrain
##        bad good
##   bad   95   52
##   good  98  255
```

```
## Misclassification rate on train data with Naive Bayes classification is:  0.3
```

```
## Misclassification rate on test data using Naive Bayes classification is:  0.344
```

Misclassification rate on train data with Naive Bayes classification is: 30%. Misclassification rate on test data using Naive Bayes classification is: 34.4%. The error rate has increased from step 3 from 26% to 34.4% for the test data which implies that naive baye's is not good predictor.

```
## Warning in data.matrix(newdata): NAs introduced by coercion
## Warning in data.matrix(newdata): NAs introduced by coercion
## Warning in data.matrix(newdata): NAs introduced by coercion
## Warning in data.matrix(newdata): NAs introduced by coercion
## Warning in data.matrix(newdata): NAs introduced by coercion
## Warning in data.matrix(newdata): NAs introduced by coercion
## Warning in data.matrix(newdata): NAs introduced by coercion
## Warning in data.matrix(newdata): NAs introduced by coercion
## Warning in data.matrix(newdata): NAs introduced by coercion
## Warning in data.matrix(newdata): NAs introduced by coercion
## Warning in data.matrix(newdata): NAs introduced by coercion
## Warning in data.matrix(newdata): NAs introduced by coercion
## Warning in data.matrix(newdata): NAs introduced by coercion
## Warning in data.matrix(newdata): NAs introduced by coercion
## Warning in data.matrix(newdata): NAs introduced by coercion
## Warning in data.matrix(newdata): NAs introduced by coercion
## Warning in data.matrix(newdata): NAs introduced by coercion
## Warning in data.matrix(newdata): NAs introduced by coercion
## Warning in data.matrix(newdata): NAs introduced by coercion
```



It is seen that Naive Baye's performs better according to the ROC plot,with a higher true positive rate and lesser false positive rate.

This type of graph is called a Receiver Operating Characteristic curve (or ROC curve.) It is a plot of the true positive rate against the false positive rate for the different possible cutpoints of a diagnostic test.

An ROC curve demonstrates several things:

It shows the tradeoff between sensitivity and specificity (any increase in sensitivity will be accompanied by a decrease in specificity). The closer the curve follows the left-hand border and then the top border of the ROC space, the more accurate the test. The closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the test.

```
##       nbtest1
##        FALSE TRUE
##   bad     66    9
##   good   130   45
```

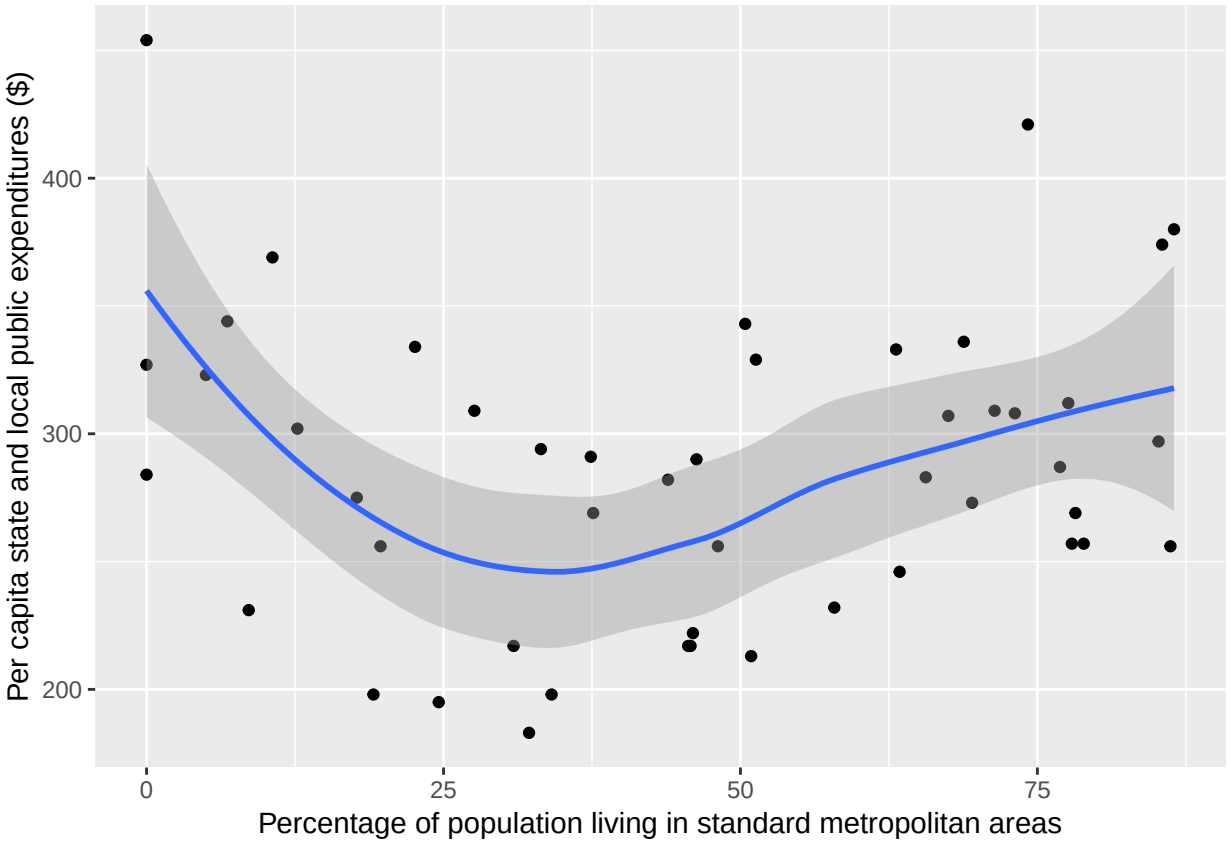
```
##       nbtrain1
##        FALSE TRUE
##   bad    137   10
##   good   263   90
```

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

```
## [1] 0.556
```

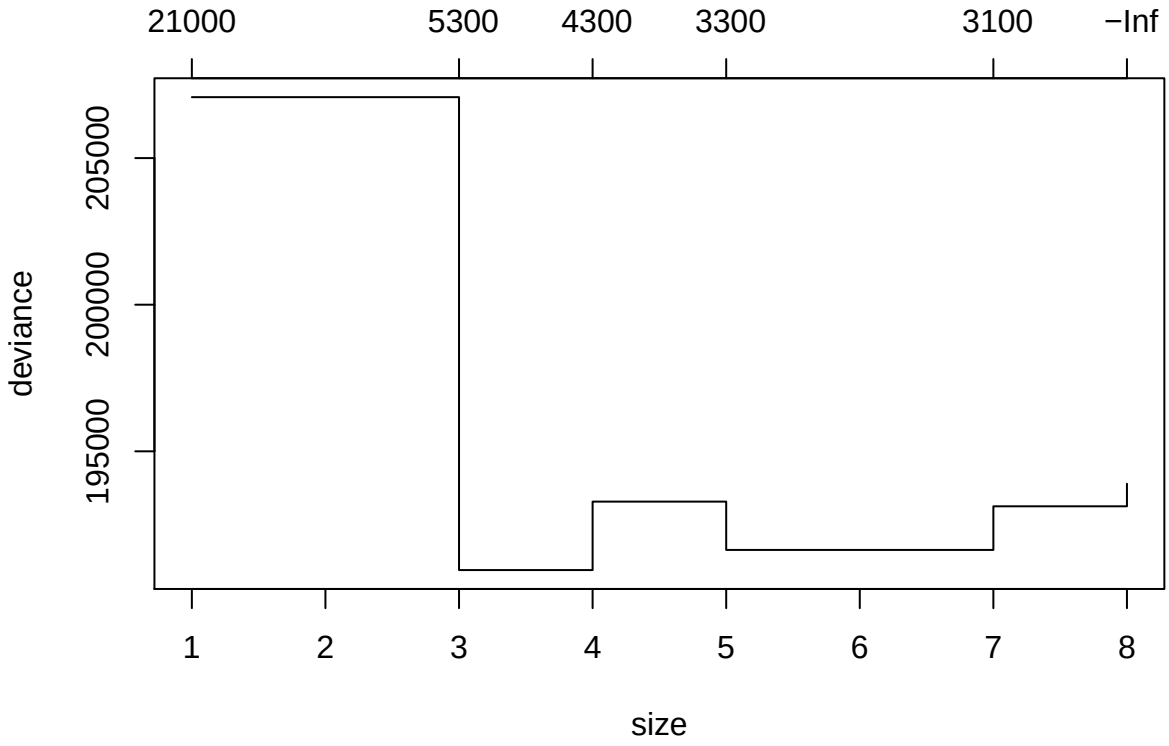
The misclassification rate for the training dataset is reported as 54.6% and slightly higher for the test at 55.6%. This error rate is much higher from Naive Baye's also which stood at 34.4%.

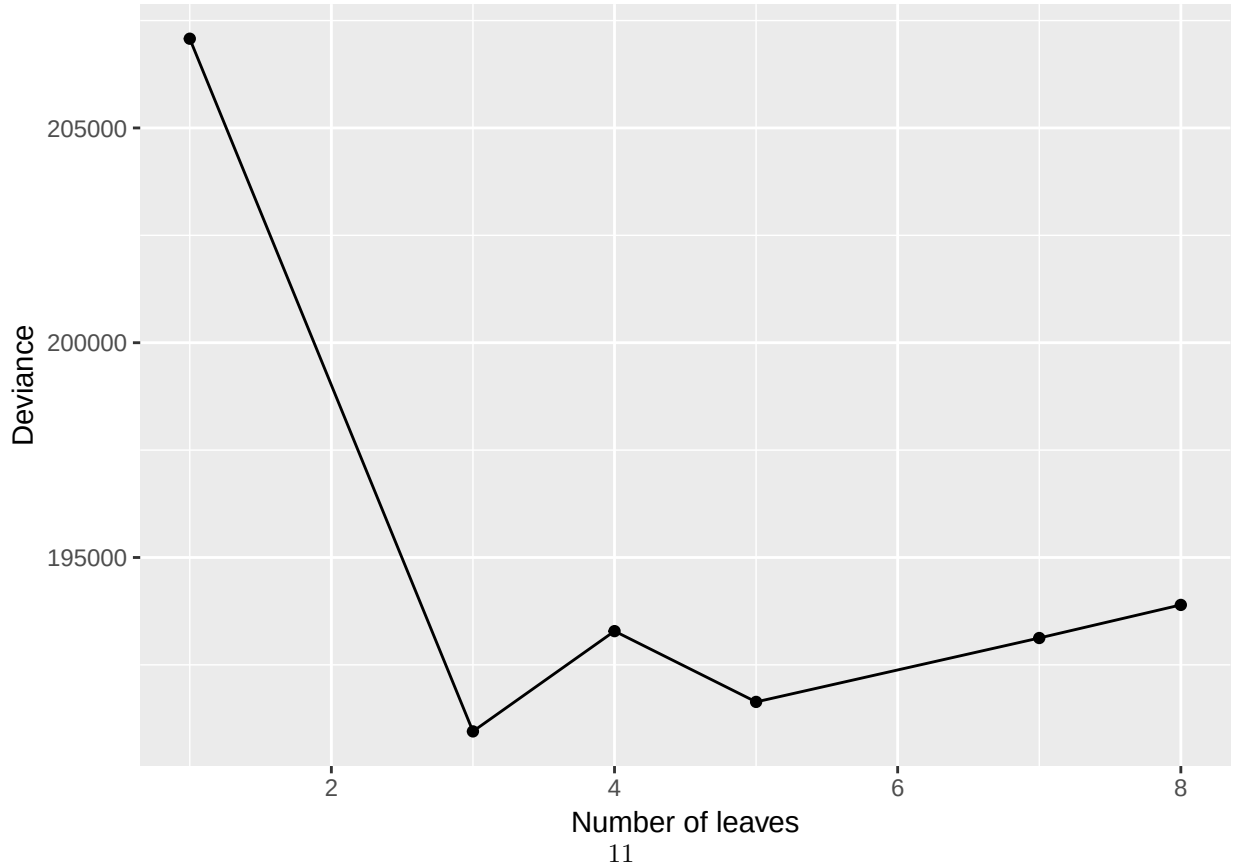
Assignment3 Uncertainty Estimation

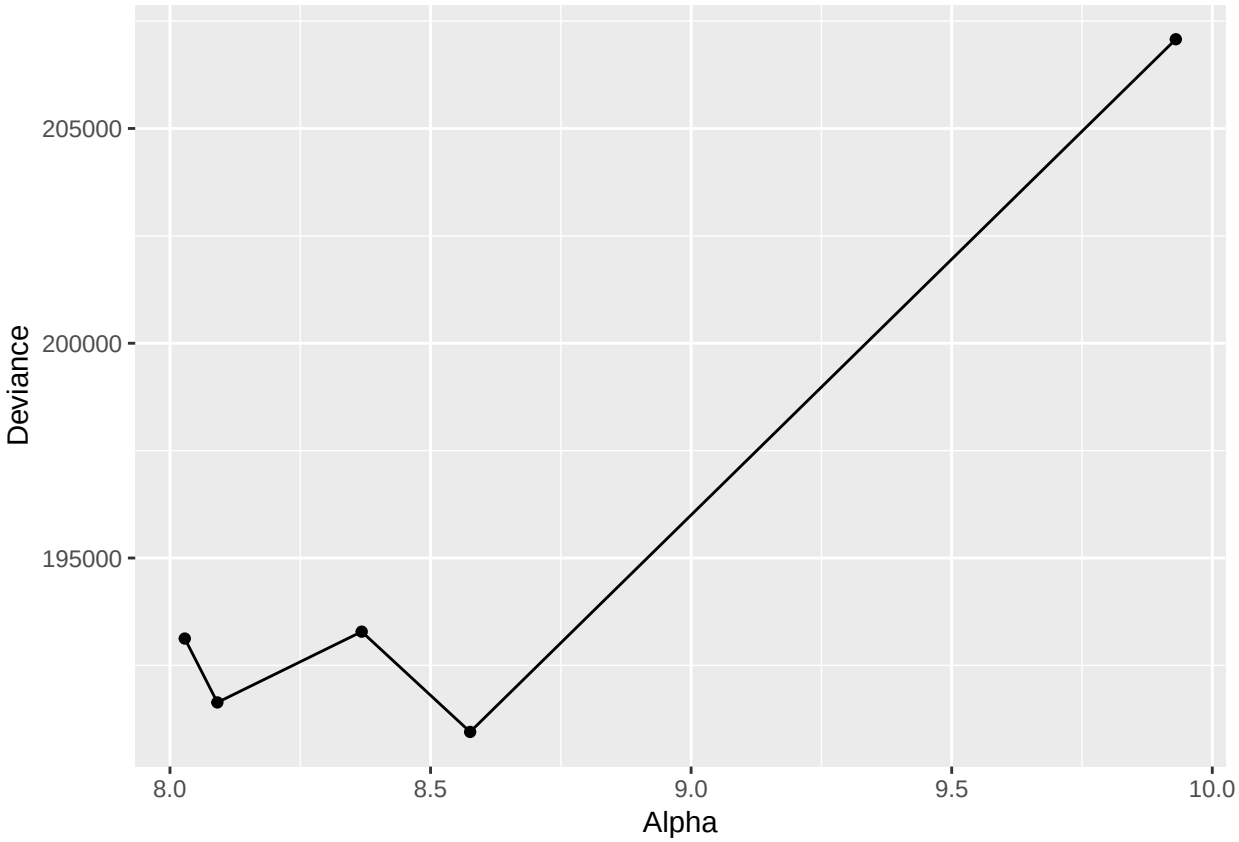


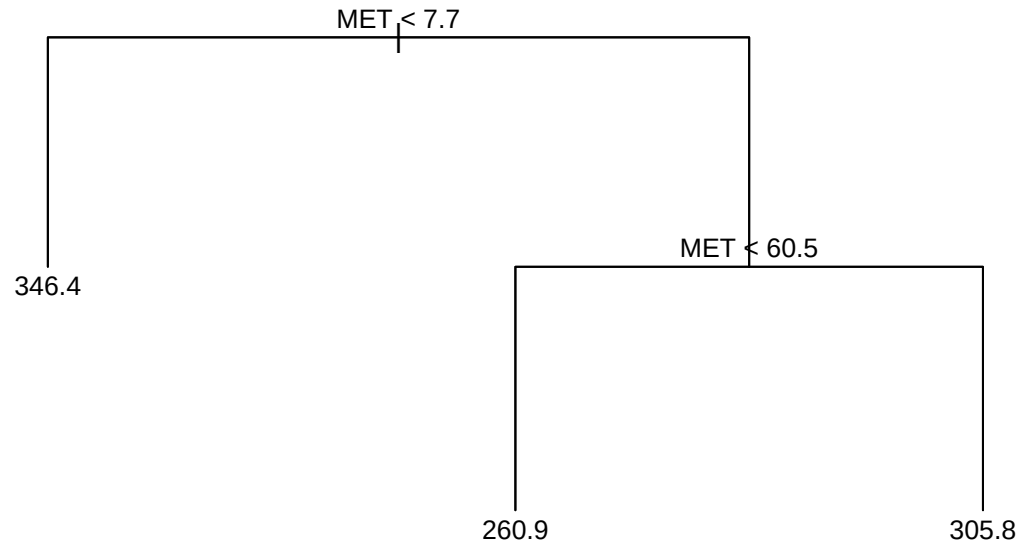
As seen from the plot data seems to be scatter all around, the variance is high. A decision tree would be better to be used here.

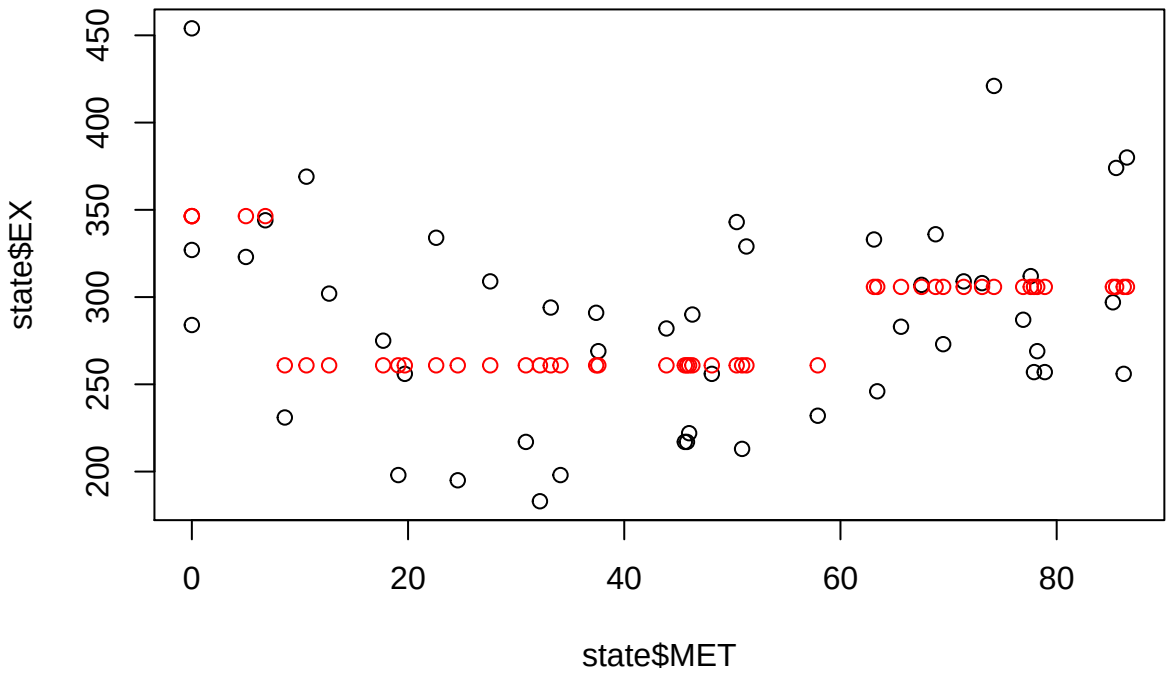
3.2

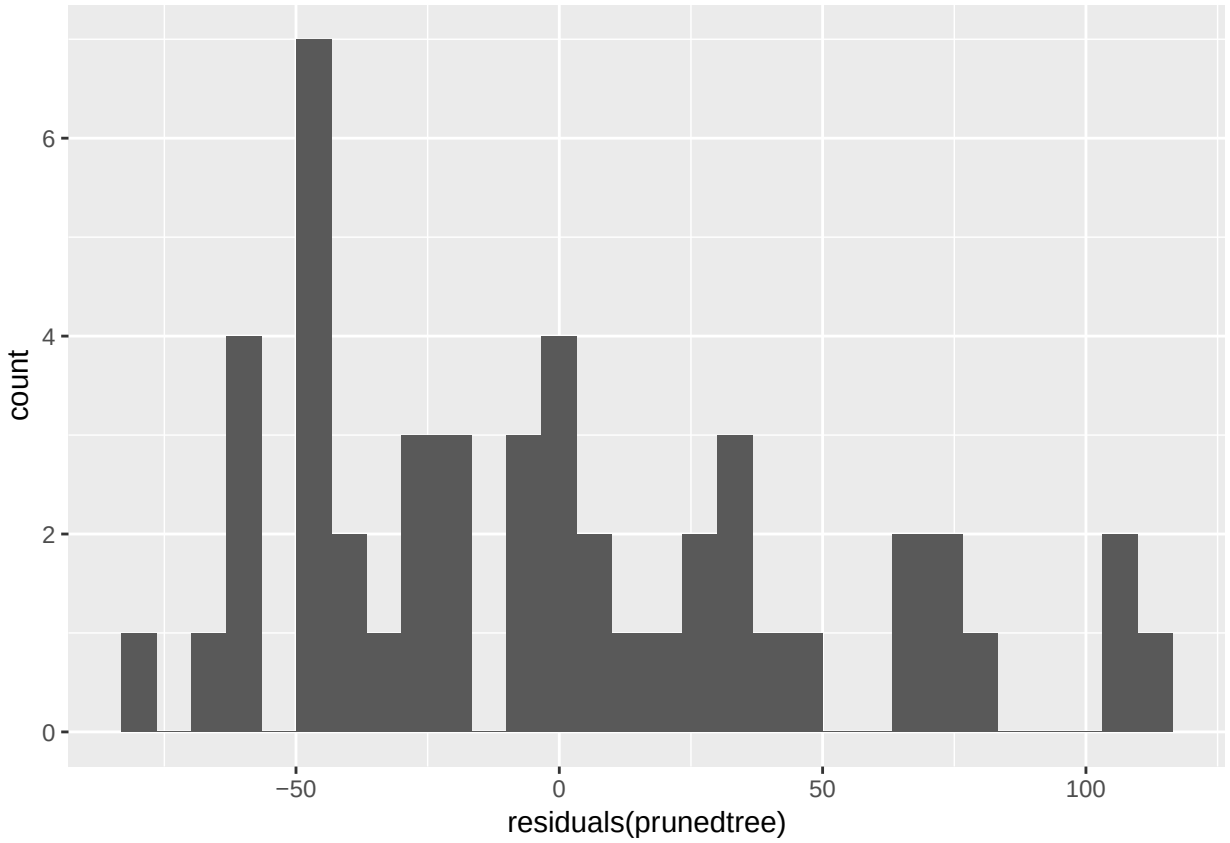

21000
5300
4300
3300
3100
−Inf
deviance
205000
200000
195000
1
2
3
4
5
6
7
8
size

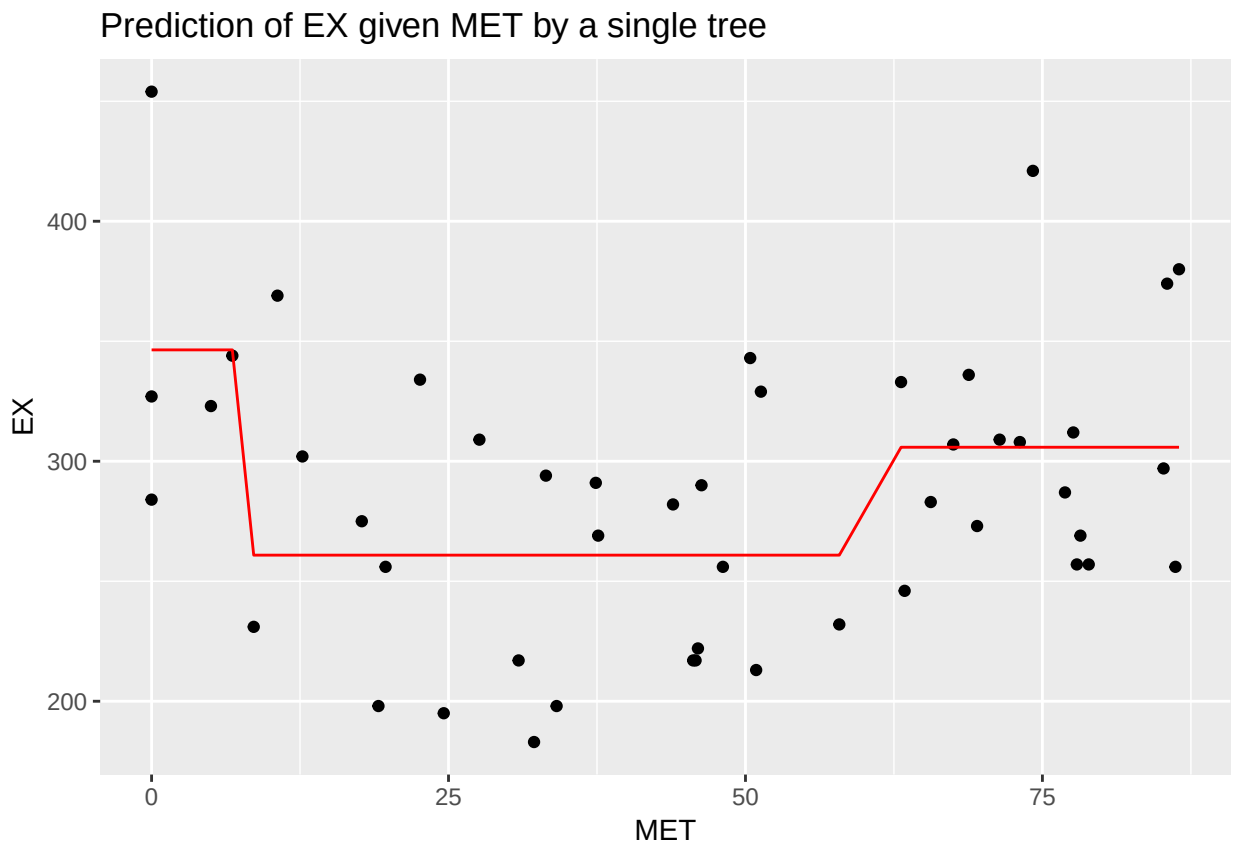

Deviance
205000
200000
195000
2
4
6
8
Number of leaves


205000
200000
195000
Deviance
8.0
8.5
9.0
9.5
10.0
Alpha


MET < 7.7
346.4
MET < 60.5
260.9
305.8


state$EX
200
250
300
350
400
450
0
20
40
60
80
state$MET

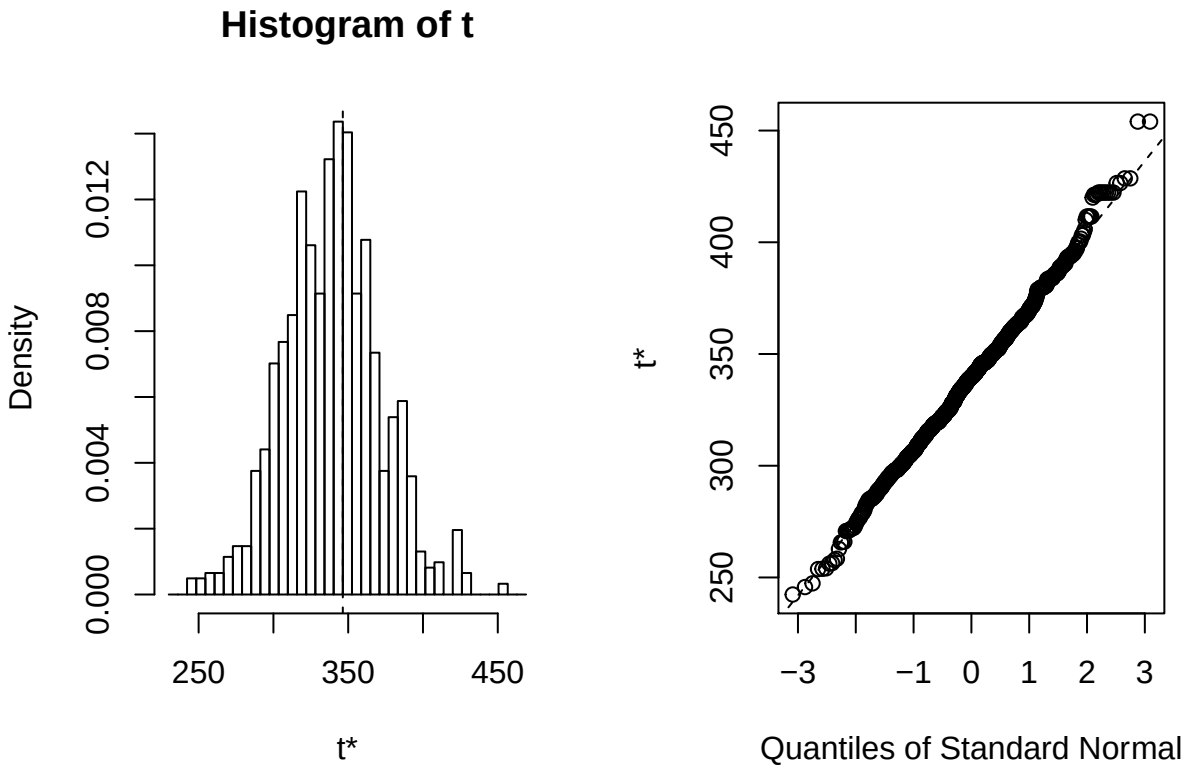

count
6
4
2
0
−50
0
50
100
residuals(prunedtree)

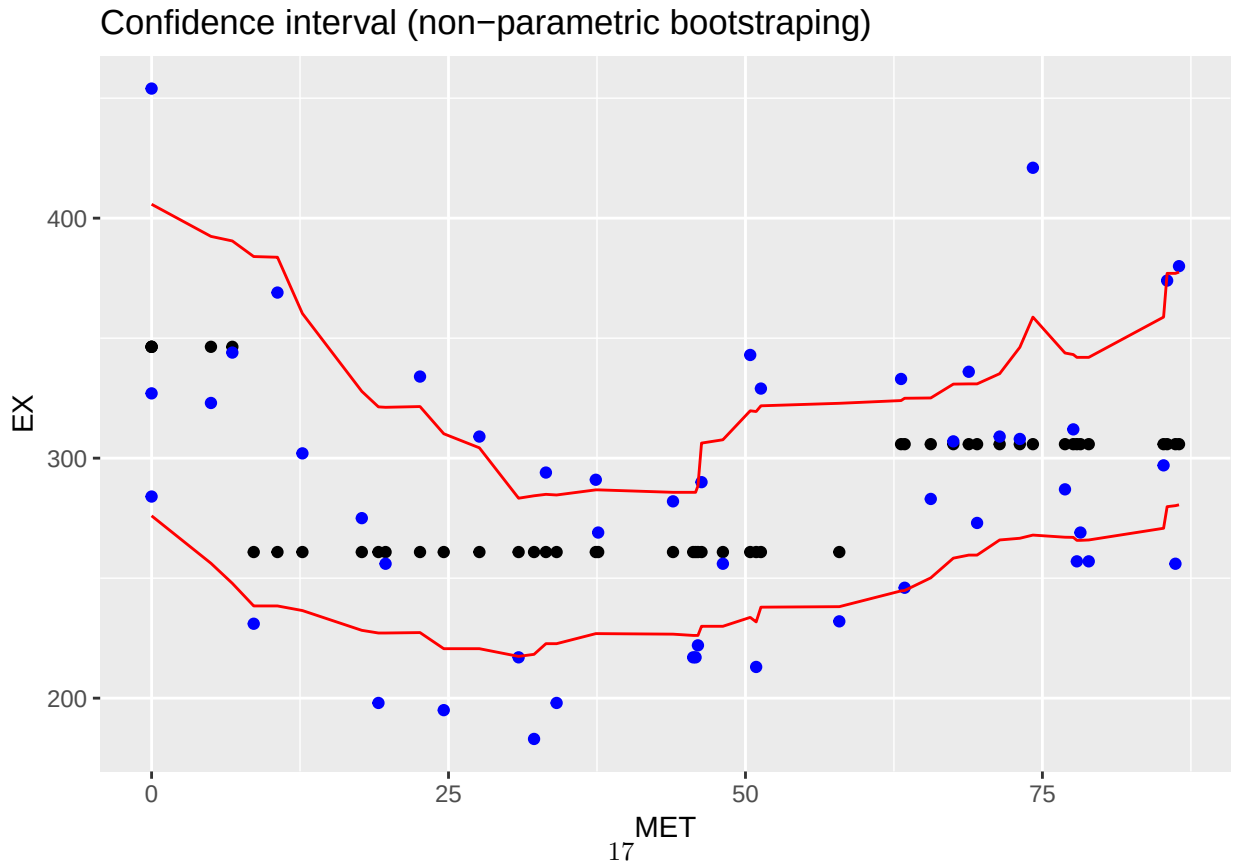


The optimal tree depth is of size 3, as it has the lowest variance when compared to others. From the plot we can say that residuals can be improved or reduced by applying better fitting.

From the plots, we can see the 3 values for each leaves on the scatter plot. The residuals seem to be normally distributed but skewed to the left, like a Chi-squared distribution that could manage negative values. This tells us that the fit is not as good as it should be, since we should expect the distribution of the residuals to be symmetric between the positive axis to the negative or vice versa.

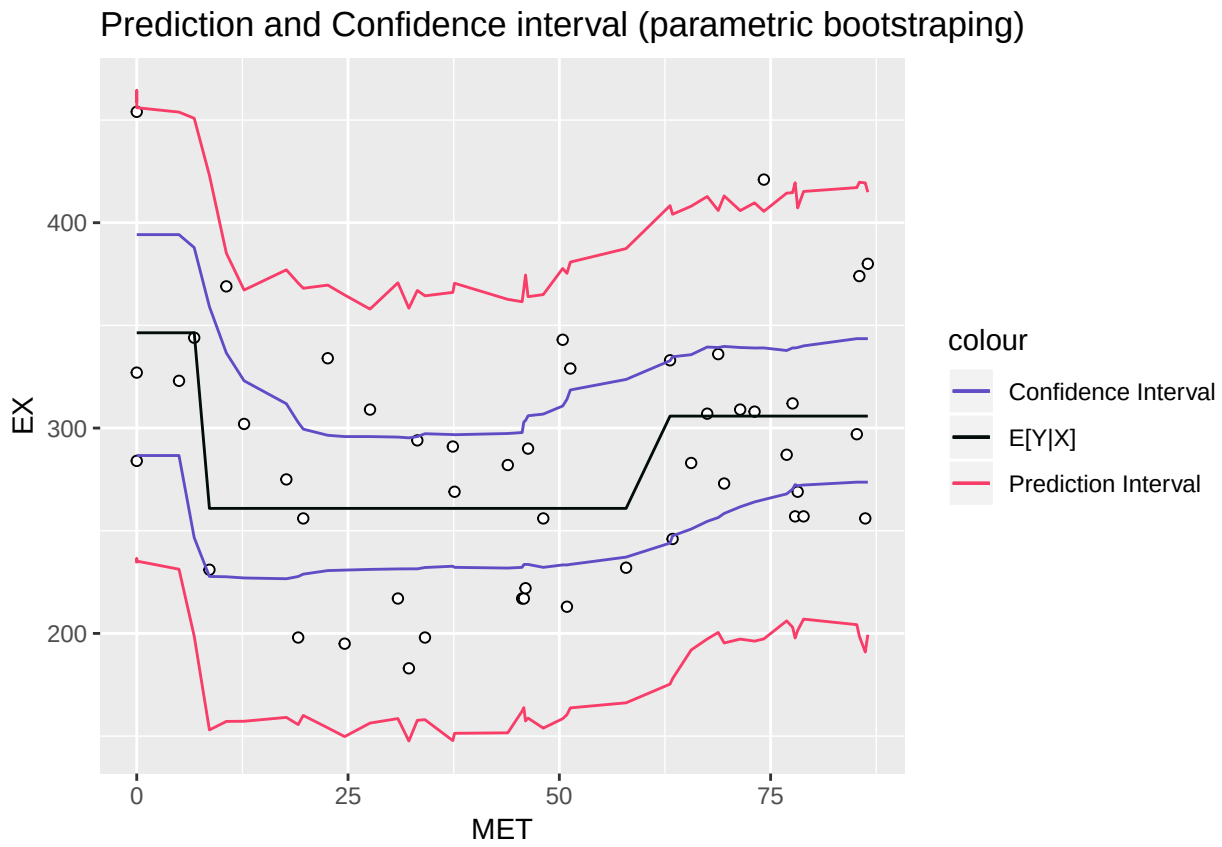
3.3 Non-Paramatric Bootstrap

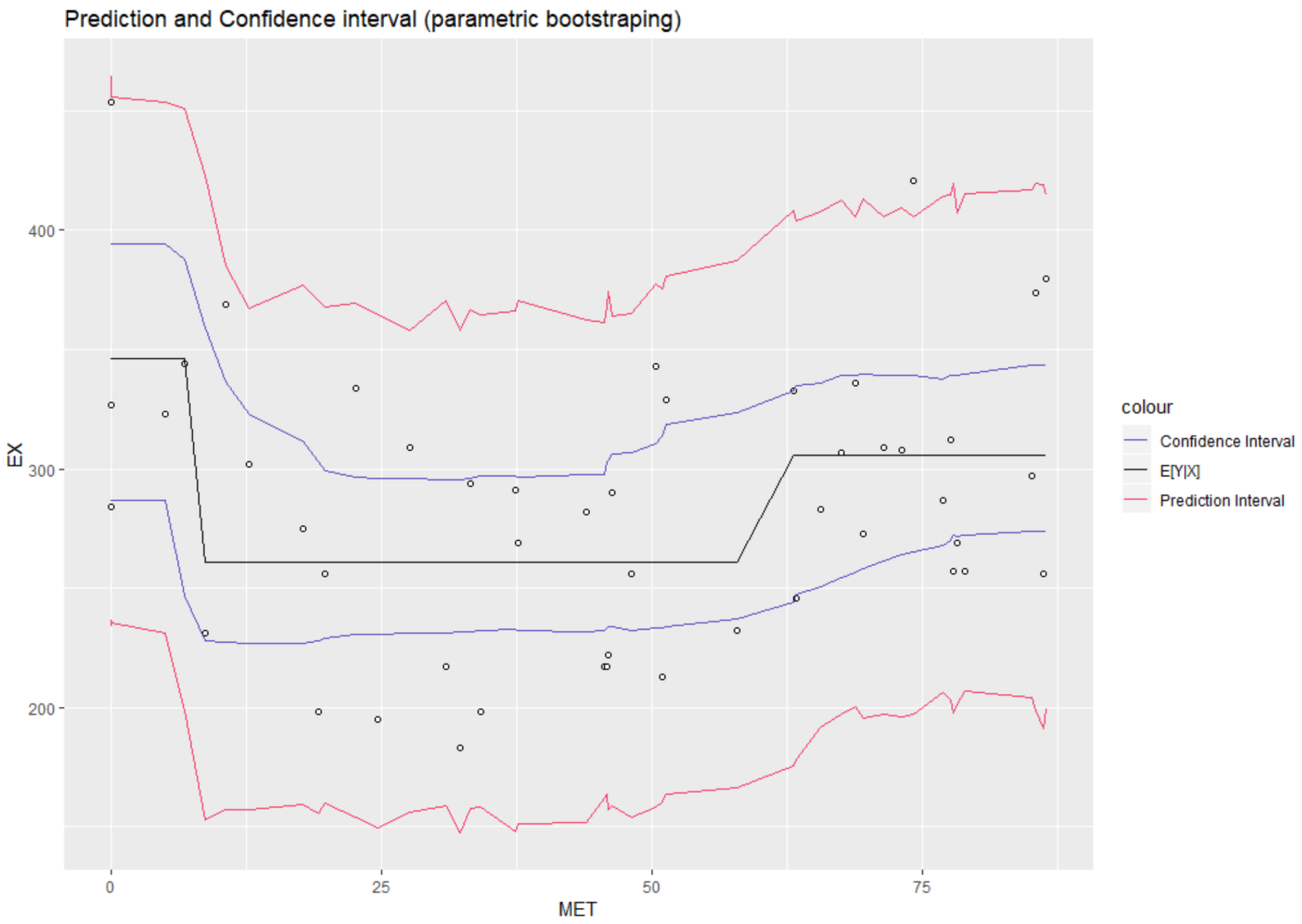




Confidence band is a combination of different confidence intervals computed for different replicates of bootstrap. The confidence band for the parametric bootstrap is volatile due to the impact of bias on bootstrap. Considering the width of confidence interval, the result of regression model computed in part 2 is reliable as it lies within the confidence band

Calculating the confidence interval using non-parametric bootstrap for the following statistic which is given by the tree: $\mu = E[Y|X]$ In this case, since no assumptions are made about the distribution, we want to generate a sample with replacement Xs and then calculate our statistic $\mu s = E[Y|Xs]$ several times (in this case 1000). After that, for each x belongs to X we get the 2.5 and 97.5 percentile of our μs,percentile and build the confidence interval. The confidence bands for this statistic seems bumpy. This is because we are calculating our statistic μ given 3 intervals (3 leaves) for each bootstrap sample which is not a smooth function and makes it bumpy when averaging over all of the samples. The predictions from our model from step 2 seems reliable since it captures the general trend of the data and it's not affected by the outliers.





In this case we assume the following distribution for our data:

$$Y \sim N(\mu, \sigma^2)$$

Where

$$\mu$$

is given by our tree:

$$\mu = \hat{f}(X|\hat{\Theta}(X,Y)) = E_{Y\sim N}[Y|X]$$

and

$$\sigma^2$$

is given by the variance of the residuals:

$$\sigma^2 = Var(Y - \mu) = Var(Y - \hat{f}(X|\hat{\Theta}(X,Y))$$

The first step is to create a sample

$$Y_s$$

from Y|X. Given this sample we want to create intervals for μ and for Y . The interval for μ is going to be constructed by calculating multiple

$$\mu_s$$

from each bootstrap sample generated from

$$Y \sim N(\mu, \sigma^2)$$

.It ss worth noting that each µs is being generated by a different

$$\hat{f}_s$$

, which means, that we are going to train a different tree for each tuple

$$(X, Y_s), s \in (1, 2, ..., S)$$

where S is the number of samplings we are going to perform (in this case 1000).

$$\text{t}_s = \hat{f}_s(X|\hat{\Theta}(X, Y_s))$$

As for the interval of Y|X, we get

$$\mu_s$$

from a bootstrap sample tuple

$$(X, Y_s)$$

and then we get our bootstrap sample

$$Y_{boot}$$

from

$$N(\mu_s, \sigma^2)$$

. We repeat this procedure 1000 times and again, we select the 2.5 and 97.5 percentile to create the 95% interval. The width of the confidence band seems to resemble the one from the previous task. So, as stated above, the predictions from the model in step 2 seems to be reliable. As for the prediction band, to the naked eye it doesn't seem that 5% of the data is outside of it and it's totally fine to happen since we assume the following distribution for the data

$$Y \sim N(\mu, \sigma^2)$$

. This means, that if the plot contained all of the samples Yboot we would be able to see that 5% of the sampled data is outside of the prediction interval. Another way to confirm that this interval seems right is to remember that

$$2\sigma$$

from the mean

$$\mu$$

amounts for 95% of the observed data. In this case, the standard deviation of the residuals if roughly 50%, which means that the prediction interval of 95% should be ±100 around the mean.

Assignment4

4.1

```
##   [1] 1.489914e-02 9.998545e-04 2.954195e-05 1.608532e-05 1.091077e-05
##   [6] 3.939315e-06 1.414911e-06 4.981545e-07 4.262849e-07 2.577774e-07
##  [11] 2.080001e-07 1.587511e-07 1.425823e-07 1.126727e-07 7.232246e-08
##  [16] 6.878939e-08 5.307373e-08 4.373598e-08 3.975200e-08 3.627181e-08
##  [21] 3.473207e-08 2.838554e-08 2.750156e-08 2.356802e-08 2.057859e-08
##  [26] 1.921151e-08 1.772579e-08 1.719151e-08 1.546958e-08 1.450458e-08
##  [31] 1.349010e-08 1.229577e-08 1.210005e-08 1.144210e-08 1.068630e-08
```

```
##  [36] 1.046807e-08 9.148433e-09 8.884085e-09 8.567593e-09 8.126130e-09
##  [41] 7.768325e-09 7.271742e-09 7.005011e-09 6.462762e-09 6.415715e-09
##  [46] 6.123419e-09 5.705293e-09 5.634860e-09 5.489343e-09 5.237779e-09
##  [51] 5.146764e-09 4.927885e-09 4.683481e-09 4.541157e-09 4.483382e-09
##  [56] 4.334378e-09 4.102898e-09 3.859924e-09 3.754631e-09 3.735784e-09
##  [61] 3.569046e-09 3.458093e-09 3.357414e-09 3.280258e-09 3.117910e-09
##  [66] 3.077594e-09 2.965870e-09 2.877830e-09 2.821708e-09 2.689767e-09
##  [71] 2.553543e-09 2.451608e-09 2.443009e-09 2.351475e-09 2.323987e-09
##  [76] 2.261444e-09 2.210244e-09 2.146417e-09 2.044353e-09 1.957622e-09
##  [81] 1.932558e-09 1.871133e-09 1.864625e-09 1.752181e-09 1.698602e-09
##  [86] 1.676920e-09 1.656262e-09 1.581932e-09 1.557666e-09 1.467634e-09
##  [91] 1.410370e-09 1.380420e-09 1.347822e-09 1.336314e-09 1.236098e-09
##  [96] 1.209134e-09 1.179186e-09 1.123574e-09 1.068961e-09 9.985917e-10
## [101] 9.858015e-10 9.526850e-10 8.980246e-10 8.728980e-10 8.288593e-10
## [106] 8.144341e-10 7.545504e-10 7.293165e-10 7.199373e-10 6.733103e-10
## [111] 6.580809e-10 6.168986e-10 6.099031e-10 5.763810e-10 5.525311e-10
## [116] 5.421167e-10 5.266189e-10 5.093443e-10 4.827224e-10 4.463711e-10
## [121] 4.127067e-10 3.895925e-10 3.786334e-10 3.495125e-10 3.033627e-10
## [126] 2.675822e-10
```

```
##   [1] "93.332" "6.263"  "0.185"  "0.101"  "0.068"  "0.025"  "0.009"
##   [8] "0.003"  "0.003"  "0.002"  "0.001"  "0.001"  "0.001"  "0.001"
##  [15] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
##  [22] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
##  [29] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
##  [36] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
##  [43] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
##  [50] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
##  [57] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
##  [64] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
##  [71] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
##  [78] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
##  [85] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
##  [92] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
##  [99] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
## [106] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
## [113] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
## [120] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
```

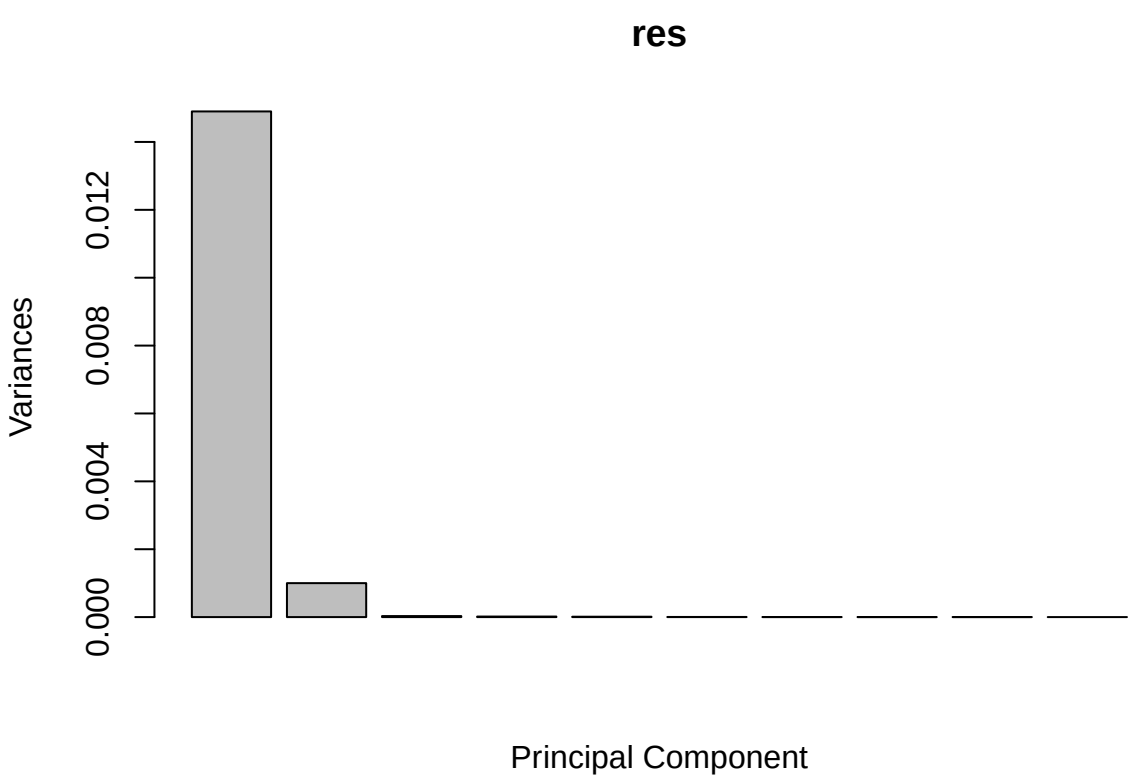
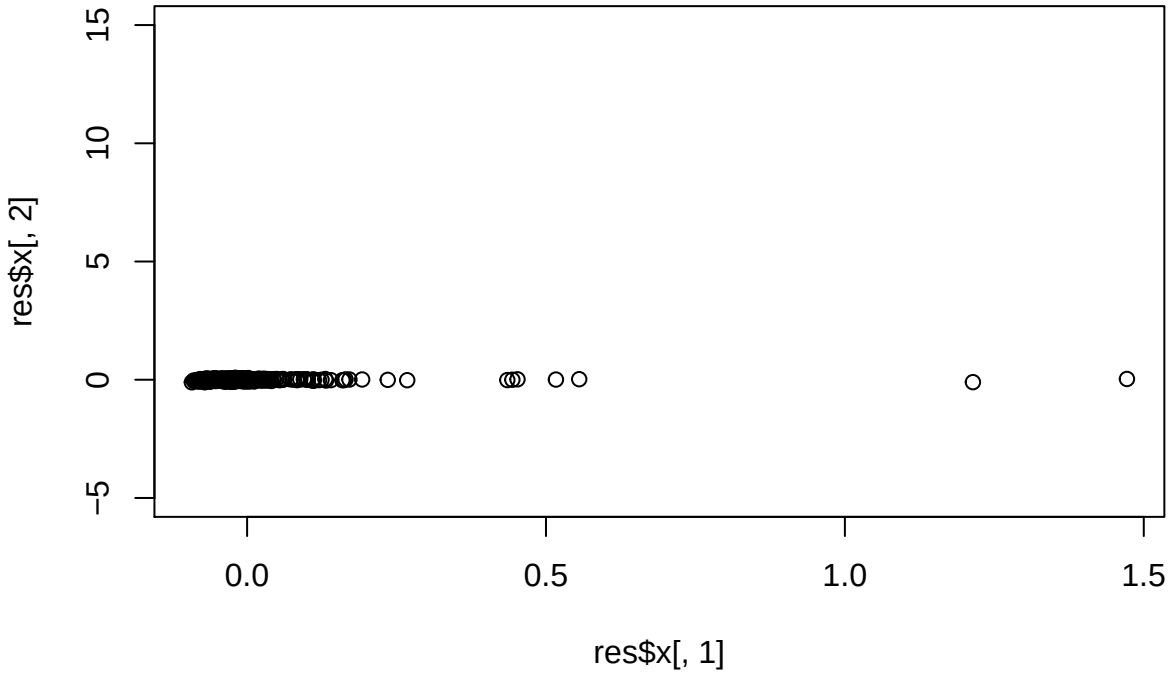
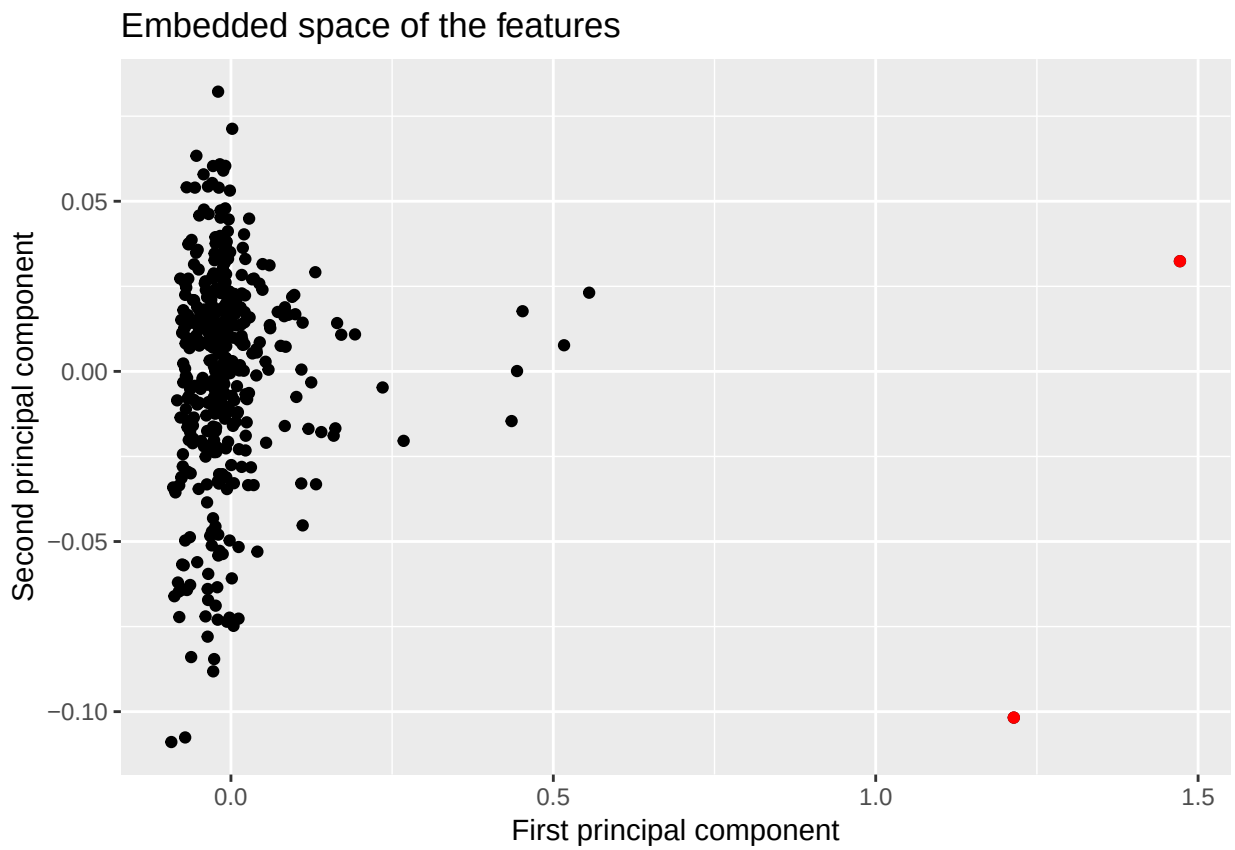

res
Variances
0.000
0.004
0.008
0.012
Principal Component

Table 1: Eigenvalues for the top axis of the new basis

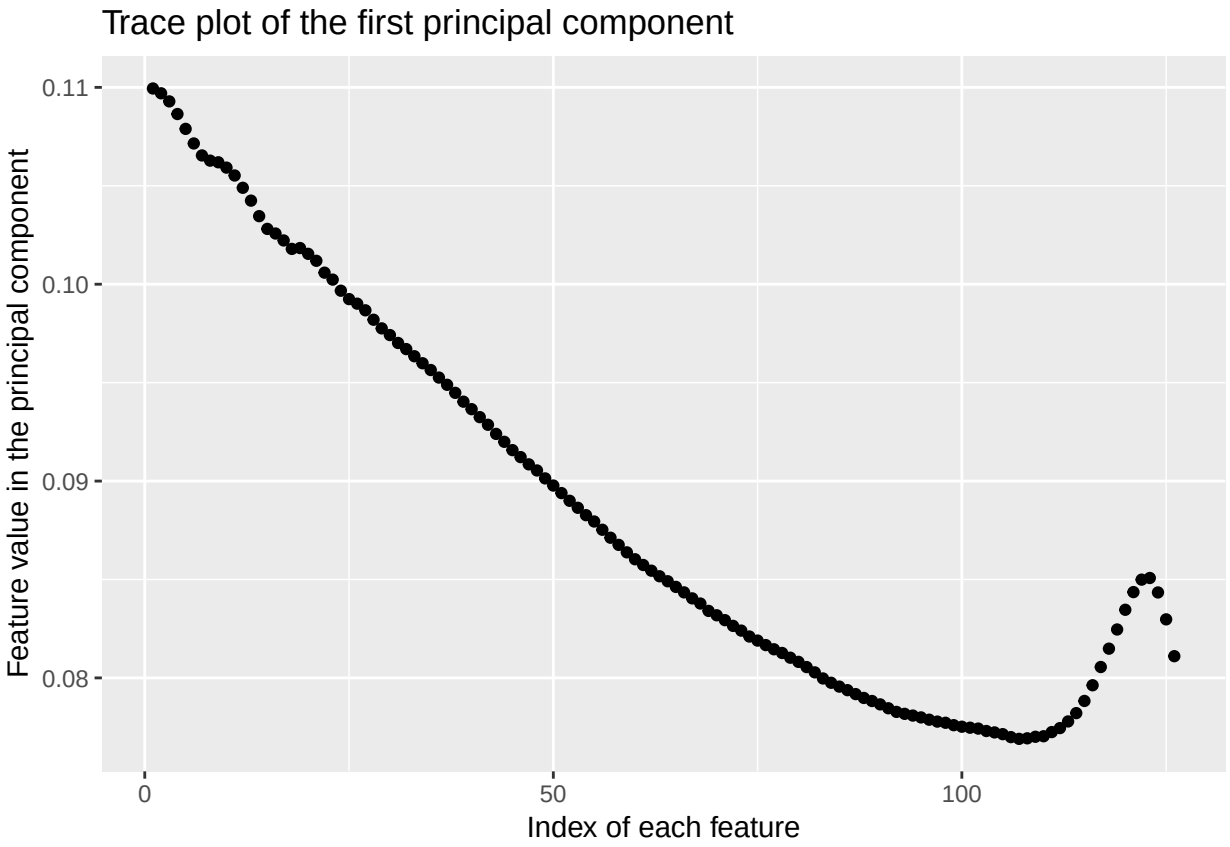
x
0.0148991
0.0009999

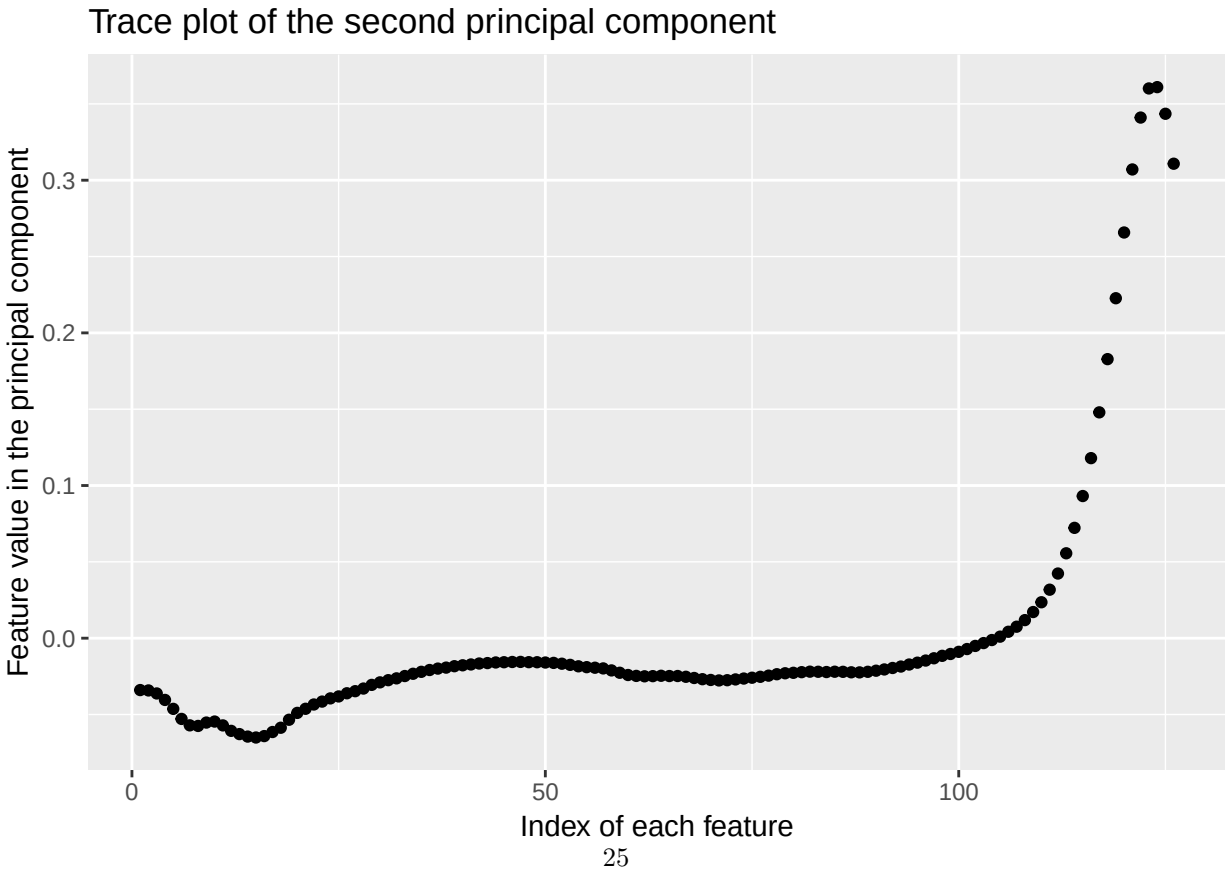




The first plot shows how many Principal componenets should be extracted. According to the plot the first 2 principal components should be extracted. Yes ,Unusual diesel fuels are seen as outliers in the 2nd plot.

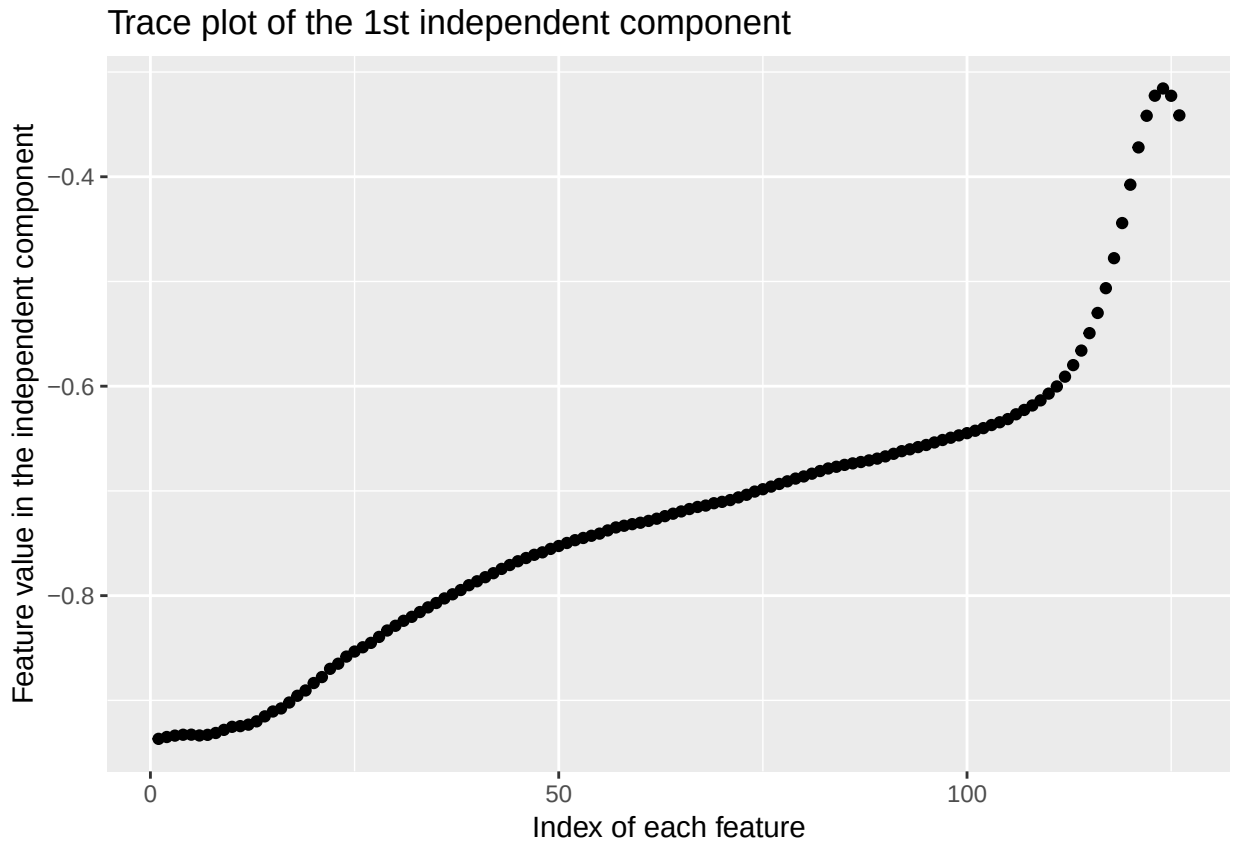
4.2

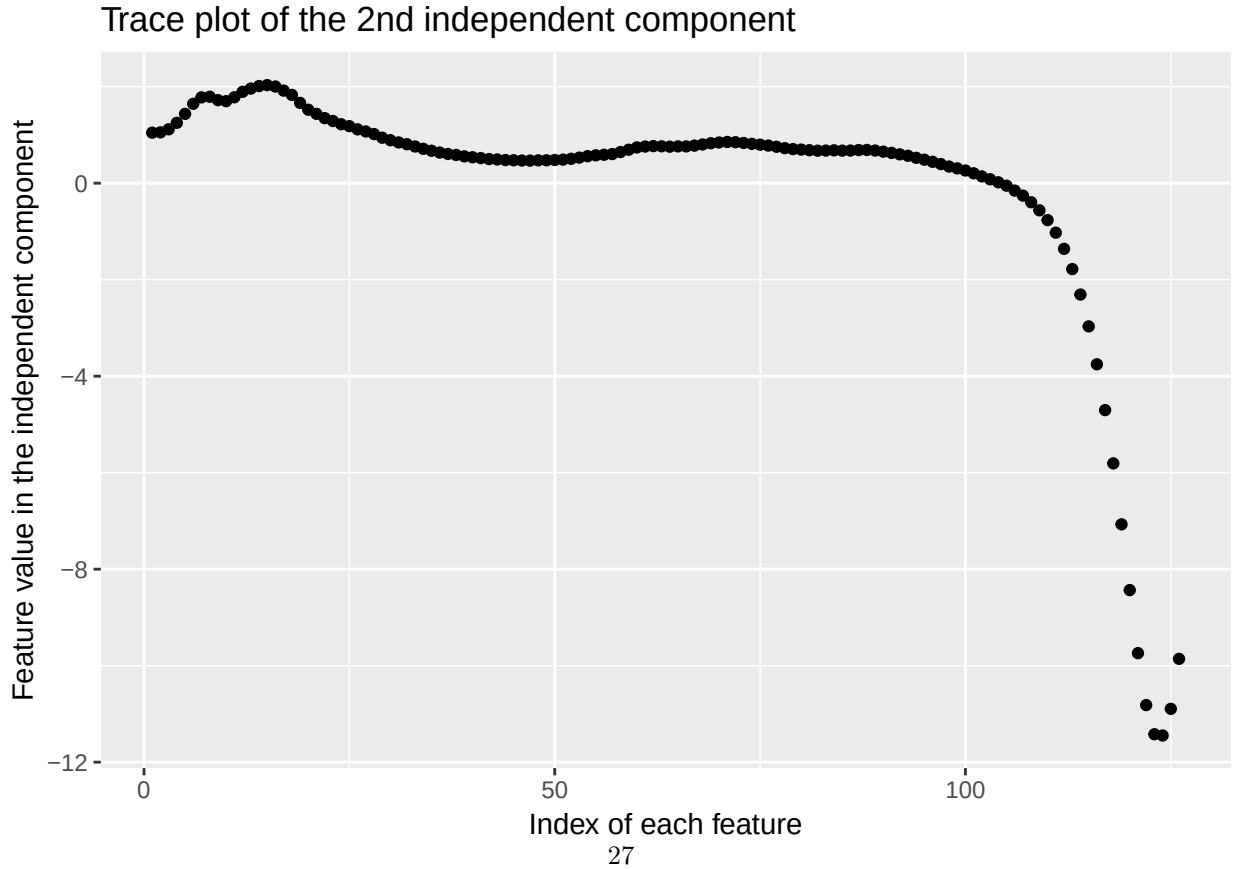

Trace plot of the first principal component
Feature value in the principal component
0.11
0.10
0.09
0.08
0
50
100
Index of each feature

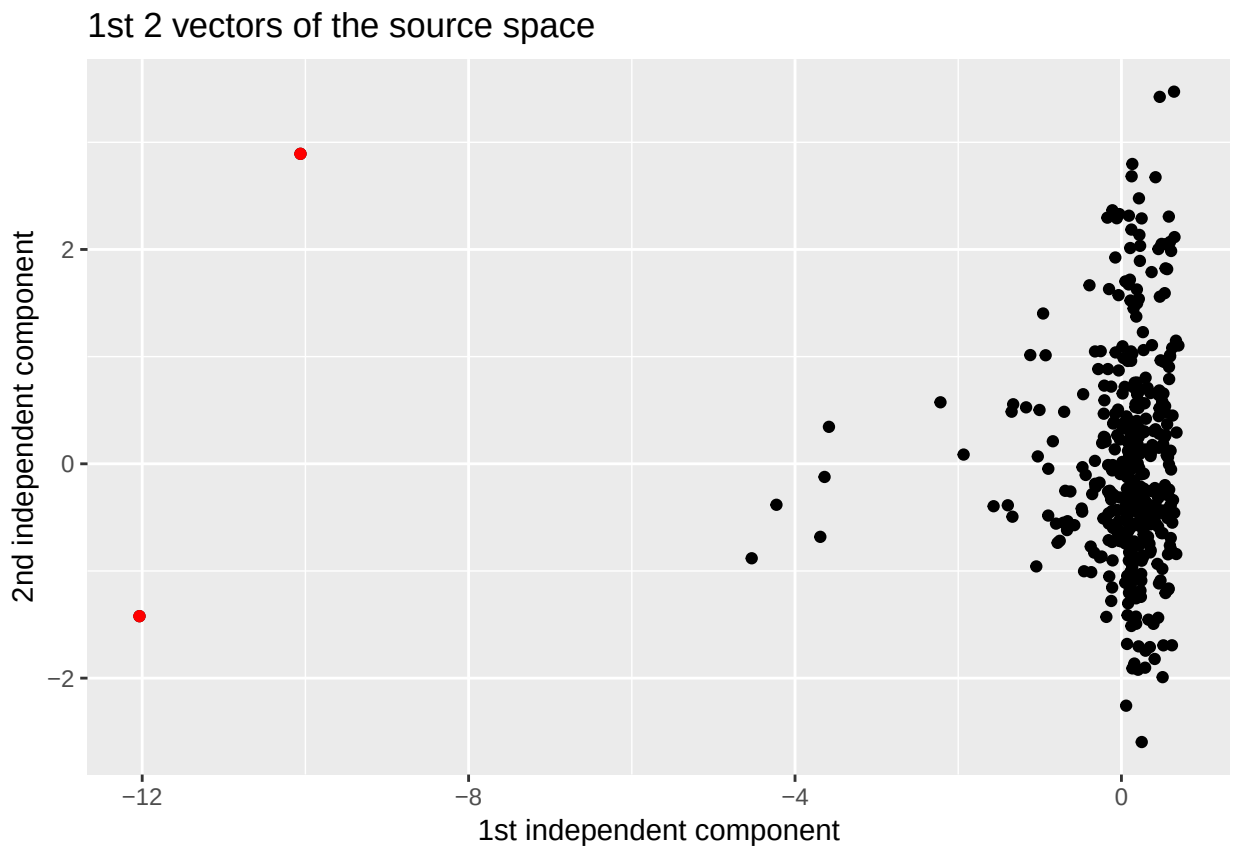

Trace plot of the second principal component
Feature value in the principal component
0.3
0.2
0.1
0.0
0
50
100
Index of each feature

By looking at the second plot it can be seen that Pincipal Component 2 can be explained by a few features which fall in the columns after 120 in the actual dataset i.e last 7-8 features since most of its features are around 0.

4.3


Trace plot of the 1st independent component
Feature value in the independent component
−0.4
−0.6
−0.8
0
50
100
Index of each feature


Trace plot of the 2nd independent component
Feature value in the independent component
0
−4
−8
−12
0
50
100
Index of each feature



The trace plots of W' show that ICA found a similar basis as the one that PCA found. The only difference is that the axis is inverted. This can be seen on the plot with the scores. This means that the latent variables found by PCA and ICA have the properties of being statistically independent, non-gaussian and identifiable. In this case W' represents a direct projection from the feature space to the source space found by ICA. It first projects the data to the PCA space S = X W' = (XK)W and then it projects it again to the source space

$$S = XW' = X_{white}W$$

. As stated above, we get the same latent space but with the axis inverted and elongated. In addition we also get the same two unusual diesel fuels and once again, they are coloured in red.

Apendix

```
knitr::opts_chunk$set(echo = TRUE)
library(dplyr)
library(plotly)
library(ggplot2)
library(xlsx)
library(readxl)
library(tree)
library(e1071)
library(boot)
library(kableExtra)
library(fastICA)
library(knitr)
```

```
creditscoring = read_excel("creditscoring.xls")
n=dim(creditscoring)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=creditscoring[id,]
tester=creditscoring[-id,]
m=dim(tester)[1]
id1<-sample(1:m,floor(m*0.5))
test<-tester[id1,]
validation<-tester[-id1,]
treestep1<-tree(as.factor(good_bad) ~ .,data=train,split = "deviance")
summary(treestep1)

treestep2<-tree(as.factor(good_bad) ~ .,data=train,split = "gini")
summary(treestep2)

devfit1<-predict(treestep1, newdata = train, type = "class")
ginifit1<-predict(treestep2, newdata = train, type = "class")

plot(treestep1)

plot(treestep2)
tabdev1<-table(devfit1,train$good_bad)
tabgini1<-table(ginifit1,train$good_bad)
msrdev1 <- 1-sum(diag(tabdev1))/sum(tabdev1)
msrdev1
msrgini1 <- 1-sum(diag(tabgini1))/sum(tabgini1)
msrgini1

devfit2<-predict(treestep1,newdata = test,type = "class")
ginifit2<-predict(treestep2, newdata = test, type = "class")
plot(devfit2)
plot(ginifit2)
tabdev2<-table(devfit2,test$good_bad)
tabgini2<-table(ginifit2,test$good_bad)
msrdev2 <- 1-sum(diag(tabdev2))/sum(tabdev2)
msrdev2
msrgini2 <- 1-sum(diag(tabgini2))/sum(tabgini2)
msrgini2
i<-summary(treestep1)[4]$size
trainscore<-rep(0,i)
testscore<-rep(0,i)
for (o in 2:i) {
  sniptree<-tree::prune.tree(treestep1,best=o)
  pred=predict(sniptree, newdata=validation,type="tree")
  trainscore[o]=deviance(sniptree)
  testscore[o]=deviance(pred)
}
plot(2:i,trainscore[2:i],col="Black",type = "b", main = "Dependence on Deviance",
     ylim=c(min(testscore[2:i]),max(trainscore)), pch=19, cex=1, ylab="Deviance")
points(2:i,testscore[2:i],col="Red",type="b", pch=19, cex=1)

msrtest = prune.tree(treestep1, best = 4)
```

```
summary(msrtest)
fit = predict(msrtest, newdata = test, type="class")
testconf= table(test$good_bad,fit)
print("Confusion Matrix")
testconf
mcr <- 1-sum(diag(testconf))/sum(testconf)
print("Misclassification rate")
mcr
plot(msrtest)
text(msrtest)
nb<-naiveBayes(as.factor(good_bad) ~ . , data=train)
nbtest = predict(nb, newdata = test[,-20], type = "class") #removing the last column with 'good_bad'
nbtrain = predict(nb,newdata = train[,-20])
# Confusion Matrix Using Naive Bayes
nbtesttab<- table(test$good_bad,nbtest)
print(nbtesttab)
nbtraintab<-table(train$good_bad,nbtrain)
print(nbtraintab)
mcrnbtrain <- 1-sum(diag(nbtraintab))/sum(nbtraintab)
cat("Misclassification rate on train data with Naive Bayes classification is: ", mcrnbtrain)
# Missclassification test data value Using Naive Bayes
mcrnbtest <- 1-sum(diag(nbtesttab))/sum(nbtesttab)
cat("Misclassification rate on test data using Naive Bayes classification is: ", mcrnbtest)
#Q5
df = data.frame(pi=double(), tree_tpr=double(), tree_fpr=double(),
naive_tpr=double(), naive_fpr=double())
for (pi in seq(0.05, 0.95, by=0.05)) {
pred_tree = as.data.frame(predict(msrtest, test))
pred_naive = as.data.frame(predict(nb, test, type="raw"))
pi_tree = ifelse(pred_tree$good > pi, "good", "bad")
pi_naive = ifelse(pred_naive$good > pi, "good", "bad")
tree_table = table(test$good_bad, factor(pi_tree, levels=c("bad", "good")))
naive_table = table(test$good_bad, factor(pi_naive, levels=c("bad", "good")))

df = rbind(df, c(pi,
tree_table[4]/(tree_table[4]+tree_table[2]),
tree_table[3]/(tree_table[3]+tree_table[1]),
naive_table[4]/(naive_table[4]+naive_table[2]),
naive_table[3]/(naive_table[3]+naive_table[1])
))
}
colnames(df) = c("pi", "tpr_tree", "fpr_tree", "tpr_naive", "fpr_naive")
plot(-1, -1, xlim=c(0, 1), ylim=c(0, 1), xlab="FPR", ylab="TPR",
main="ROC curve for Naive Bayes vs Tree model")
lines(df$fpr_tree, df$tpr_tree, lwd=1, col="blue")
lines(df$fpr_naive, df$tpr_naive, lwd=1, col="red")
legend("bottomright", c("Tree", "Naive"), col=c("blue", "red"), lwd=10)

#q6
nbtest1 = predict(nb, test[,-20] , type="raw")
nbtrain1 = predict(nb, train[,-20] , type="raw")
# loss matrix
nbtest1 = (nbtest1[, 2] / nbtest1[, 1]) > 10 # compare with loss matrix
```

```
nbtrain1 = (nbtrain1[, 2] / nbtrain1[, 1]) > 10
# confusion matrix for train & test
nbtabtest = table(test$good_bad,nbtest1)
nbtabtest
nbtabtrain = table(train$good_bad,nbtrain1)
nbtabtrain
# missclasification rates for train and test respectively
1-sum(diag(nbtabtrain))/sum(nbtabtrain)
1-sum(diag(nbtabtest))/sum(nbtabtest)
# 3.1 Data import, reorder and Plot
set.seed(12345)
state = read.csv2("State.csv", header = TRUE)
state = state[order(state$MET),]
ggplot(state)+geom_point(aes(x=MET,y=EX))+geom_smooth(aes(x=MET,y=EX),method = "loess")+labs(x = "Perce
set.seed(12345)
control_parameter = tree.control(nobs = nrow(state),minsize = 8)
fit_tree = tree(formula = EX ~ MET,data = state,control = control_parameter)
leaffit = cv.tree(fit_tree)
plot(leaffit)
#plotting deviance against number of leaves
p = ggplot() +
geom_line(aes(x=leaffit$size, y=leaffit$dev)) +
geom_point(aes(x=leaffit$size, y=leaffit$dev)) +
labs(x="Number of leaves", y="Deviance")
print(p)
# Plotting Deviance vs alpha.
p = ggplot() +
geom_line(aes(x=log(leaffit$k), y=leaffit$dev)) +
geom_point(aes(x=log(leaffit$k), y=leaffit$dev)) +
labs(x="Alpha", y="Deviance")
print(p)

prunedtree = prune.tree(fit_tree,best = leaffit$size[which.min(leaffit$dev)])

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

fitted_val = predict(prunedtree, newdata=state)

plot(state$MET, state$EX)
points(state$MET, fitted_val, col="red")

# Plotting the histogram of the residuals.
p = ggplot() +
geom_histogram(aes(residuals(prunedtree)), bins = 30)
print(p)

# Plotting the original data and the predictions.
p = ggplot() +
geom_point(aes(x=state$MET, y=state$EX)) +
geom_line(aes(x=state$MET, y=fitted_val), colour='red') +
labs(x="MET", y="EX", title="Prediction of EX given MET by a single tree")
print(p)
```

```
f_np = function(state,index){
  sample = state[index,]
  Ctrl = tree.control(nrow(sample), minsize = 8)
  fit = tree( EX ~ MET, data=sample, control = Ctrl)
  optimal_tree = prune.tree(fit, best= leaffit$size[which.min(leaffit$dev)])
  return(predict(optimal_tree, newdata=state))
}
np_bs = boot(state, statistic = f_np, R=1000)
conf_bound = envelope(np_bs,level=0.95) # For 95% Confidence interval

predictions = predict(prunedtree,state)
plot(np_bs)
fig_data = data.frame(orig = state$EX, x=state$MET, pred=predictions,
                      upper=conf_bound$point[1,], lower=conf_bound$point[2,])

p = ggplot(fig_data, aes(x,predictions,upper,lower)) + geom_point(aes(x, pred)) +
  geom_point(aes(x, orig),colour="blue") +
  geom_line(aes(x,upper),colour="red") +
  geom_line(aes(x,lower),colour="red")+
labs(x='MET',
y='EX',
title='Confidence interval (non-parametric bootstraping)',
color="Type")
p
# 3.4 Paramatric Bootstrap
set.seed(12345)
best_nleaves = leaffit$size[which.min(leaffit$dev)]
original_data = state
reg = tree(EX ~ MET,
data=state,
control=tree.control(nobs=nrow(state), minsize=8))
reg = prune.tree(reg, best=best_nleaves)
original_reg = reg
# Creating a function that is going to sample from our prior.
# Our prior is that the data is distributed as a normal distribution.
rng = function(state, model)
{
# Getting the parameters for the normal distribution.
nobs = nrow(state) # Number of observations.
y_hat = predict(model, newdata=state) # Predictions.
y = state$EX # Real values.
resid = y - y_hat # Residuals.
state$EX = rnorm(nobs, # Normal distribution.
y_hat,
sd(resid))
return(state)
}
# Function that is going to return the predictions, given new
# random generated data. This will be used for getting the
# confidence interval for the predictions.
citree_prediction = function(state)
{
# Training the model.
```

```
reg = tree(EX ~ MET,data=state,control=tree.control(nobs=nrow(state), minsize=8))
reg = prune.tree(reg, best=best_nleaves)
# Predicting over my original data.
y_hat = predict(reg, newdata=original_data)
return(y_hat)
}
# Function that is going to return the predictions, given our new
# random generated data. This will be used for getting the
# prediction interval for our predictions.
ptree_prediction = function(state)
{
# Fitting our tree given our bootstrap sample.
reg = tree(EX ~ MET,data=state,control=tree.control(nobs=nrow(state), minsize=8))
reg = prune.tree(reg, best=best_nleaves)
# Getting E[Y|X] given our bootstrap model
# on the original data.
y_hat = predict(reg, newdata=original_data)
# Getting the residuals from the original model.
y = original_data$EX
resid = y - predict(original_reg, newdata=original_data)
# Sampling from N(E[Y|X], Var(residuals)).
nobs = nrow(original_data)
y_hat = rnorm(nobs,y_hat,sd(resid))
return(y_hat)
}
#Running the bootstrap for the confidence interval.
results = boot(state,statistic=citree_prediction,R=1000,mle=reg,ran.gen=rng,sim="parametric")
ci_results = envelope(results)
# Running the bootstrap for
# the prediction interval.
results = boot(state,statistic=ptree_prediction,R=1000,mle=reg,ran.gen=rng,sim="parametric")
p_results = envelope(results)
# Getting the predictions.
reg = tree(EX ~ MET,data=state,control=tree.control(nobs=nrow(state), minsize=8))
reg = prune.tree(reg, best=best_nleaves)
y_hat = predict(reg, newdata=state)
# Plotting the results.
z = ggplot() +
geom_point(aes(x=state$MET, y=state$EX), fill='white', shape=21) +
geom_line(aes(x=state$MET, y=p_results$point[1, ], color="Prediction Interval")) +
geom_line(aes(x=state$MET, y=p_results$point[2, ], color="Prediction Interval")) +
geom_line(aes(x=state$MET, y=ci_results$point[1, ], color="Confidence Interval")) +
geom_line(aes(x=state$MET, y=ci_results$point[2, ], color="Confidence Interval")) +
geom_line(aes(x=state$MET, y=y_hat, color="E[Y|X]")) +
scale_colour_manual(values=c("#604dc5", "#020c0b", "#f83d69")) +
labs(x='MET', y='EX',
title='Prediction and Confidence interval (parametric bootstraping)')
print(z)
knitr::include_graphics("bootstraping.png")
NIRSpectra <- read.csv2("NIRSpectra.csv", header = TRUE)
data1 <- NIRSpectra
data1$Viscosity = c()
res=prcomp(data1)
```

```r
lambda=res$sdev^2
lambda
sprintf("%2.3f", lambda/sum(lambda)*100)
screeplot(res,xlab = "Principal Component")
U=res$rotation
#plot of scores
plot(res$x[,1],res$x[,2],ylim=c(-5,15))

# Getting the variance explained by each eigenvector.
eigen_values = res$sdev^2
pcvariance = eigen_values * 100 / sum(eigen_values)
# Components that explain at least 99% of the variation.
toppc = res$rotation[, c(1, 2)]
topev = eigen_values[c(1, 2)]
kable(topev, caption="Eigenvalues for the top axis of the new basis")
# Getting the outliers.
mask = res$x[, 1] > 1
outliers = res$x[mask, ]
# Embeded space of the coordinates
p = ggplot() +
geom_point(aes(x=res$x[, 1], y=res$x[, 2])) +
geom_point(aes(x=outliers[, 1], y=outliers[, 2]), color='red') +
labs(title="Embedded space of the features",
x="First principal component",
y="Second principal component")
print(p)
#U=res$rotation
#plot(U[,1], main="Traceplot, PC1")
#plot(U[,2],main="Traceplot, PC2")

# TASK 4.2
# Traceplots.
# First component.
p = ggplot() +
geom_point(aes(x=1:nrow(toppc), y=toppc[, 1])) +
labs(title="Trace plot of the first principal component",
x="Index of each feature",
y="Feature value in the principal component")
print(p)
# Second component.
p = ggplot() +
geom_point(aes(x=1:nrow(toppc), y=toppc[, 2])) +
labs(title="Trace plot of the second principal component",
x="Index of each feature",
y="Feature value in the principal component")
print(p)
set.seed(12345)
# Calculating W'
ica = fastICA(data1, 2)
W_prime = ica$K %*% ica$W # Latent variable, same as the eigenvalues for PCA.
# This operation maps the basis constructed from ICA
# and maps it into the eigenspace. This is mapping the PCA basis to the
# ica basis.
```

```
# Trace plots.
# First component.
p = ggplot() +
geom_point(aes(x=1:nrow(W_prime), y=W_prime[, 1])) +
labs(title="Trace plot of the 1st independent component",
x="Index of each feature",
y="Feature value in the independent component")
print(p)
# Second component.
p = ggplot() +
geom_point(aes(x=1:nrow(W_prime), y=W_prime[, 2])) +
labs(title="Trace plot of the 2nd independent component",
x="Index of each feature",
y="Feature value in the independent component")
print(p)
# Getting the outliers.
mask = ica$S[, 1] < -8
outliers = ica$S[mask, ]
# Embeded space of the coordinates
p = ggplot() +
geom_point(aes(x=ica$S[, 1], y=ica$S[, 2])) +
geom_point(aes(x=outliers[, 1], y=outliers[, 2]), color='red') +
labs(title="1st 2 vectors of the source space",
x="1st independent component",
y="2nd independent component")
print(p)
```