Untitled

vinbe289

December 10, 2018

Assignment 2

1

2

```
##
## deviance_yfit bad good
##          bad   61   20
##          good  86  333
```

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

```
##
## gini_yfit bad good
##      bad   66   38
##      good  81  315
```

```
## [1] 0.238
```

```
##
## deviance_yfit_test bad good
##               bad   28   19
##               good  48  155
```

```
## [1] 0.268
```

```
##
## gini_yfit_test bad good
##           bad   18   35
##           good  58  139
```

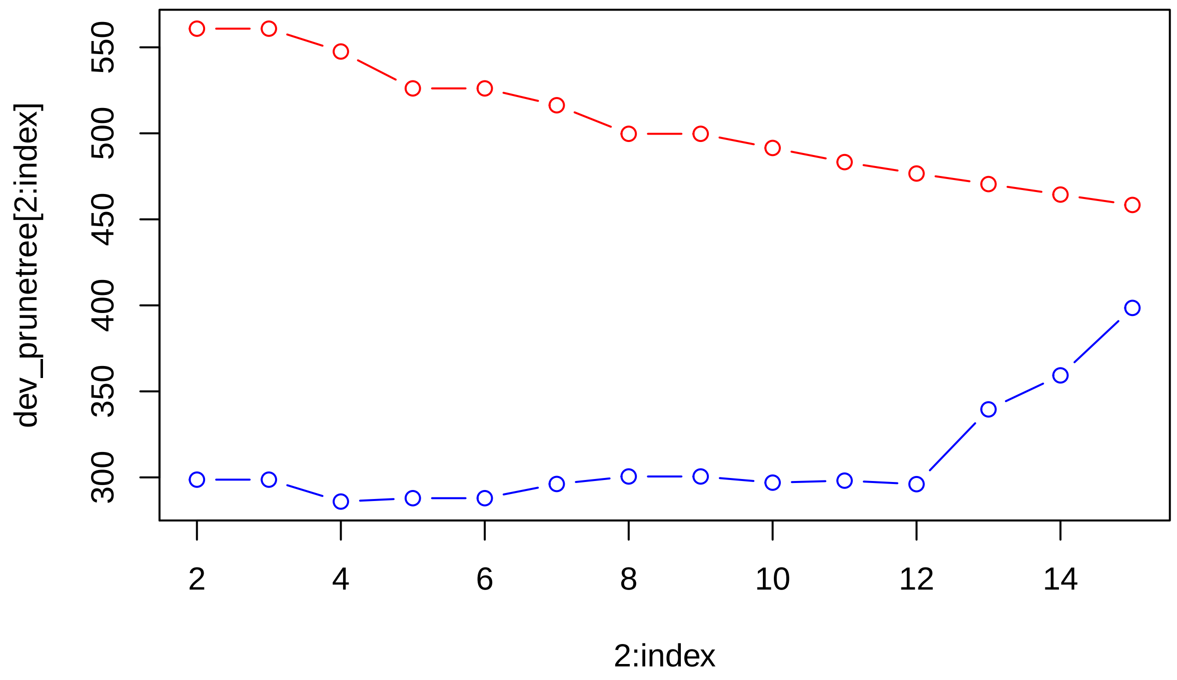
```
## [1] 0.372
```

Misclassification rate for tree fit using Deviance measure of impurity for training data is 21.2% Misclassification rate for tree fit using Gini measure of impurity for training data is 23.8%

Misclassification rate for tree fit using Deviance measure of impurity for test data is 26.8% Misclassification rate for tree fit using Gini measure of impurity for test data is 37.2%

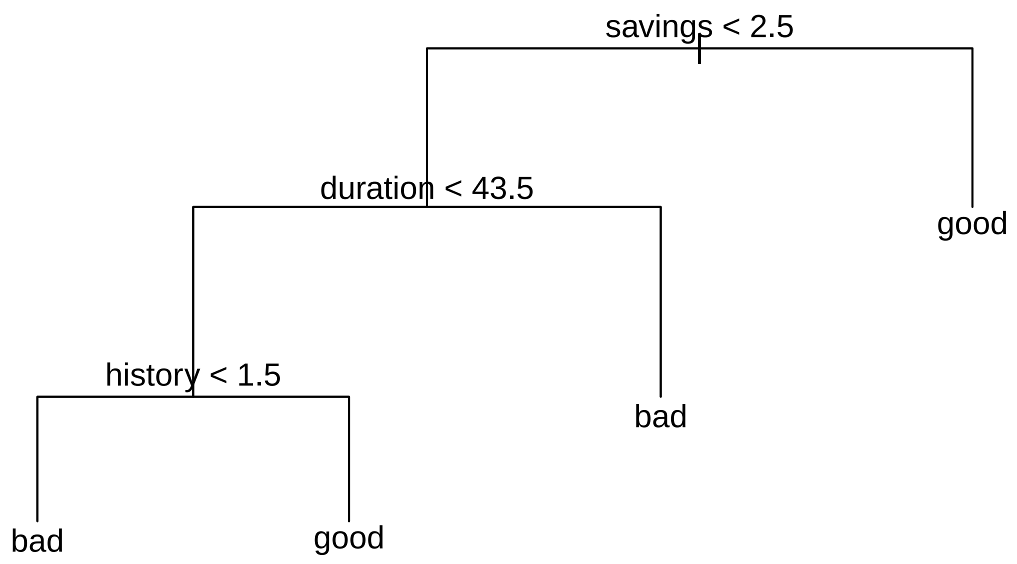
The model fit by taking Deviance as the measure of impurity seems to provide better results.

3



```
##        yfit
##          bad good
##    bad    18   58
##    good    6  168
```

```
## [1] 0.256
```



Misclassification rate is 25.6%. The better tree is found to be the tree in which deviance is considered to be the measure of impurity. The optimal tree depth is found to be 4 from the plot. The variables used by the tree were savings,duration and history.

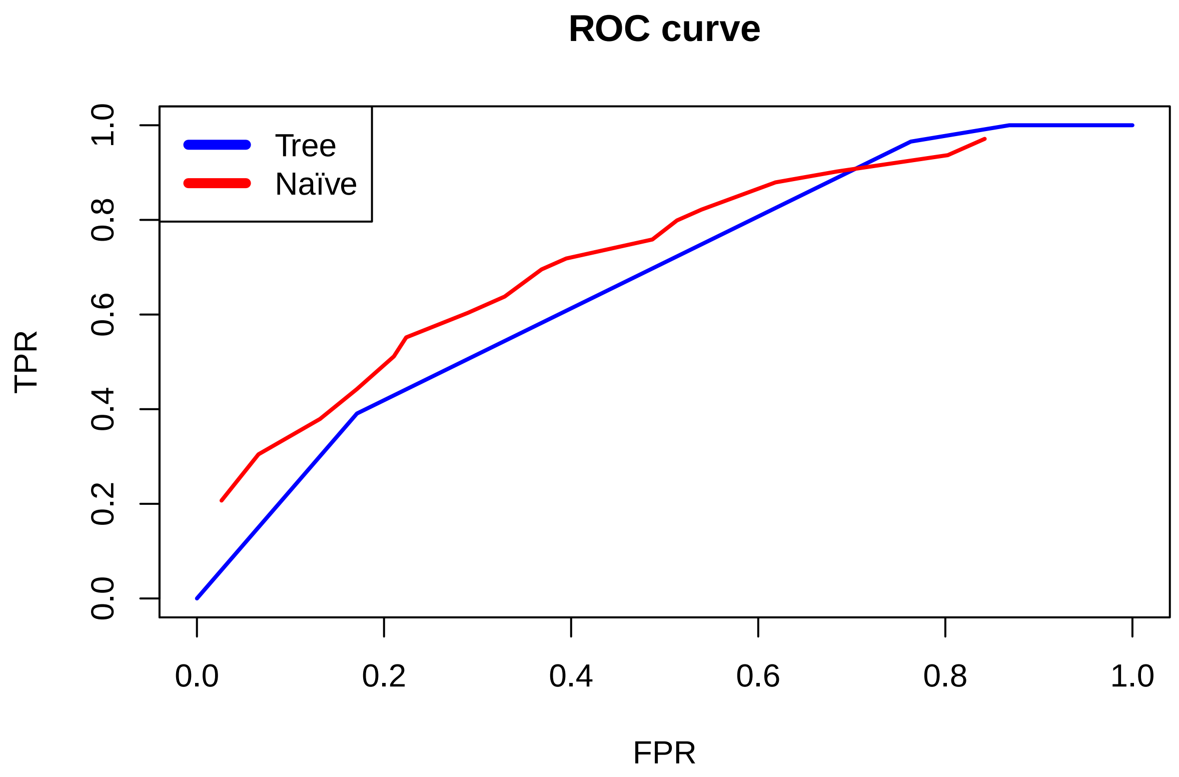
4

```
## [1] 0.3
```

```
## [1] 0.316
```

Misclassification rate for train data is 30%. Misclassification rate for test data is 31.6%. Misclassification rate in step 3 was found to be lesser than that of step 4.

5



It can be seen from the plot that Naive Bayes is a better model compared to the Decision tree with Deviance as an impurity measure since the area under the Naive Bayes curve is greater than the area under the Decision Tree curve.

6

```
##       nbtrain6
##        FALSE TRUE
##   bad    137   10
##   good   263   90
```

```
##       nbtest6
##        FALSE TRUE
##   bad     71    5
##   good   122   52
```

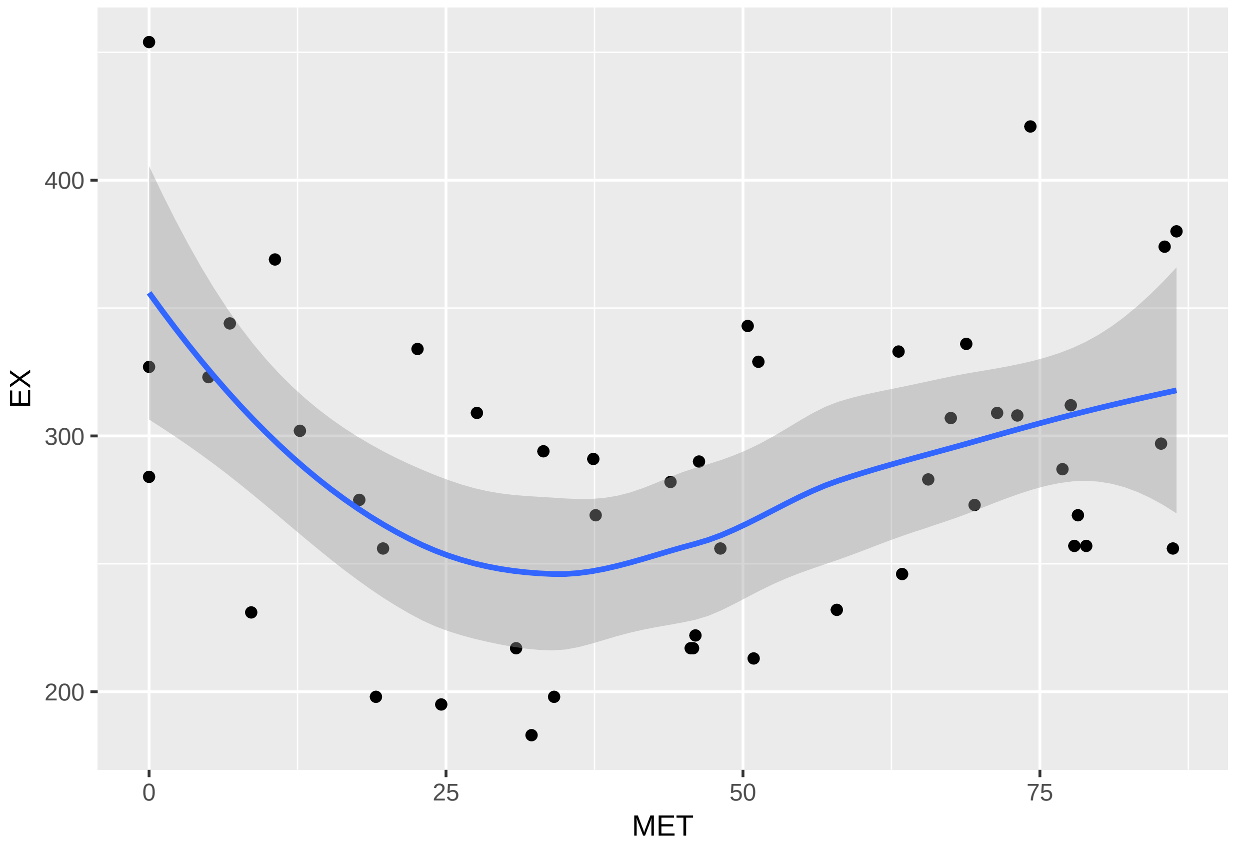
```
## [1] 0.546
```

```
## [1] 0.508
```

Missclasification rate for train data is 54.6%. Missclasification rate for test data is 50.8. Misclassification rate in step 4 was found to be lesser than that of step 6. The increase in the misclassification rate is due to the implimentation of new loss matrix which penalizes the classification of a bad customer as a good customer heavily. Due to this the model tends to classify more good customers as bad customers which results in the increase of error rate.

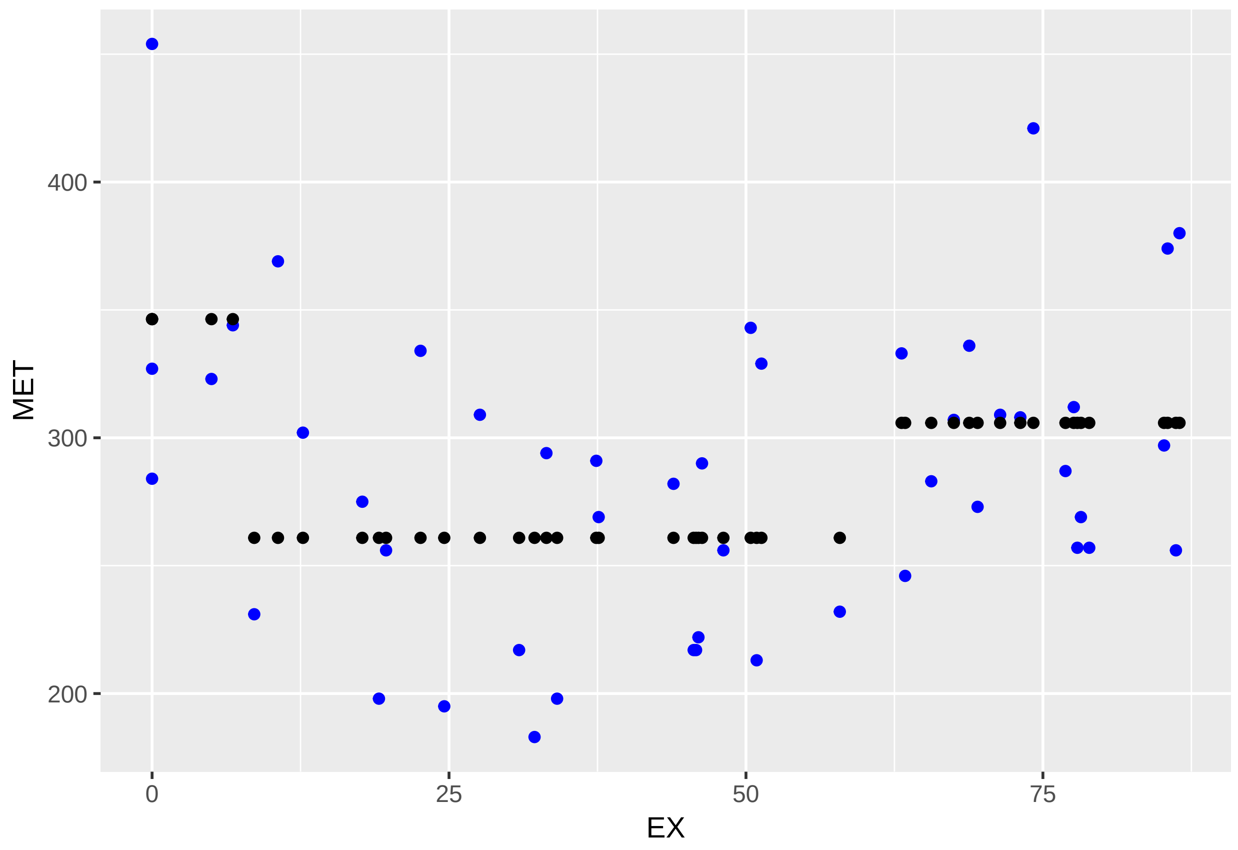
Assignment 3

1

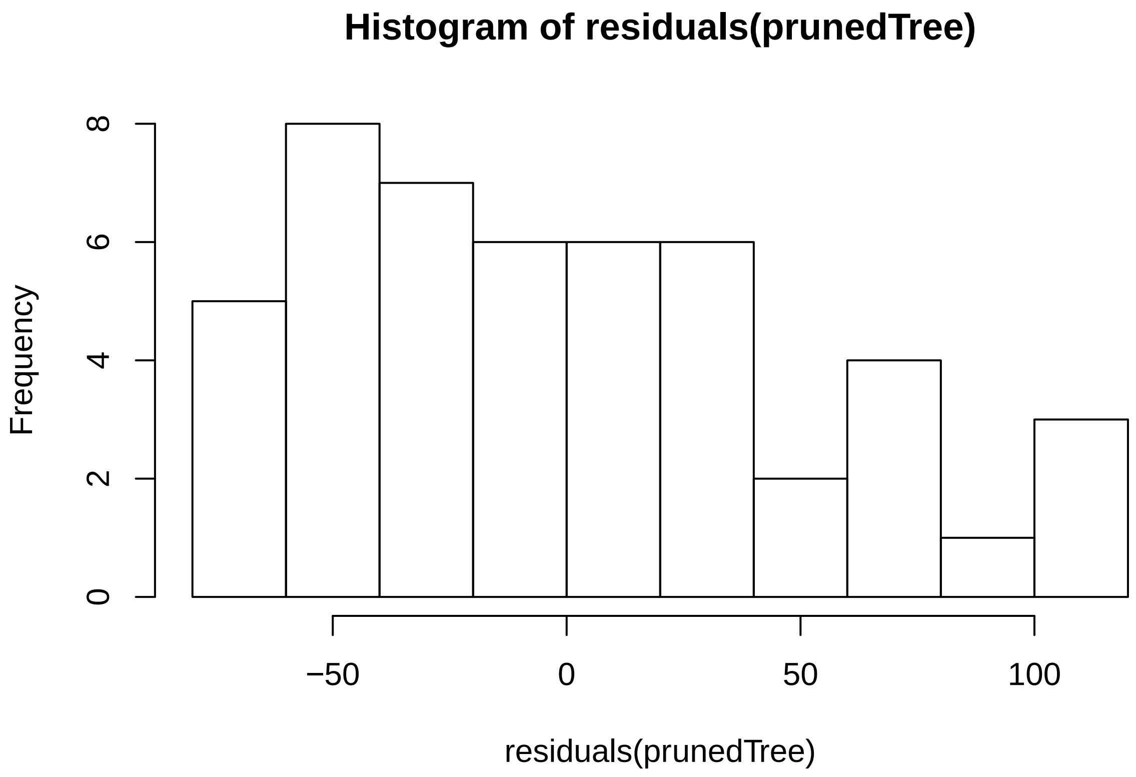


By inspecting the plot it is evident that a linear model with a higher degree polynomial can be fit to this data.

2



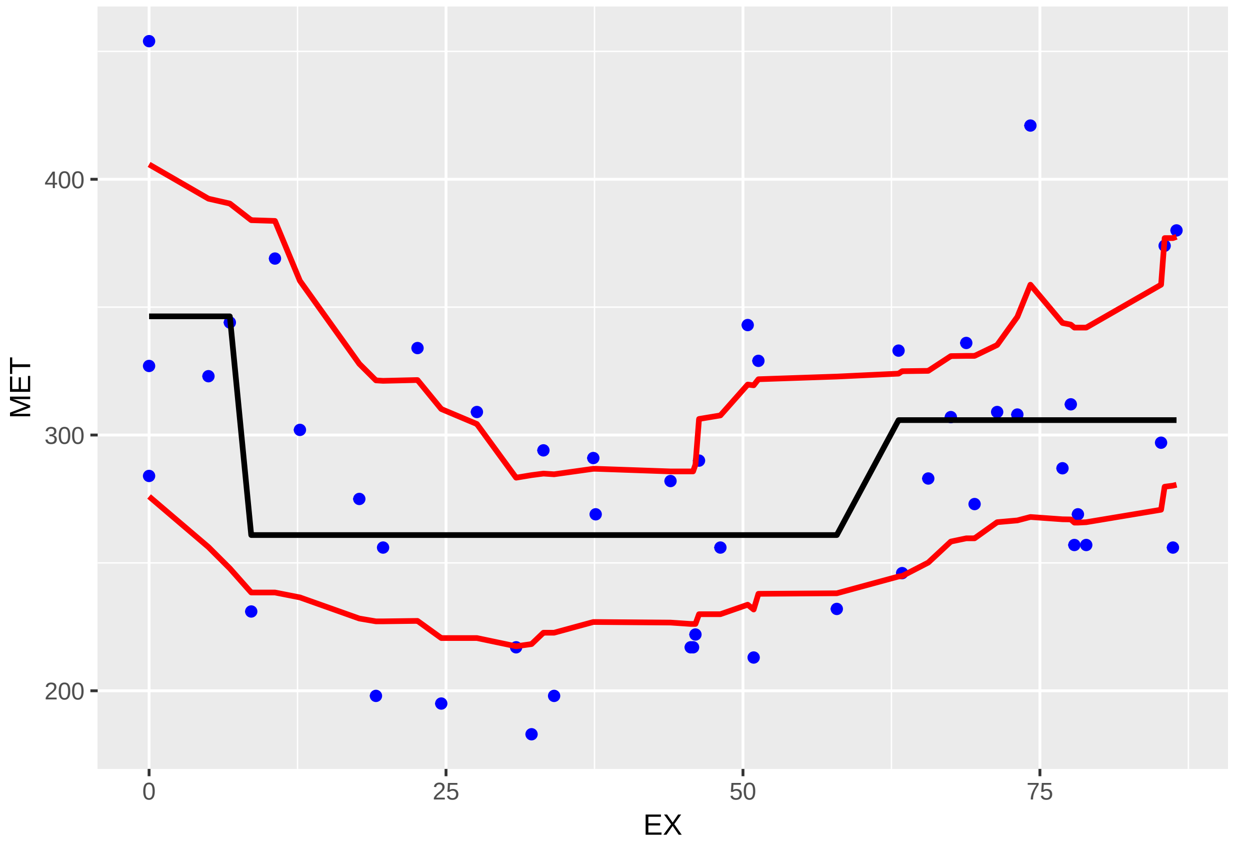
MET
400
300
200
0
25
50
75
EX



Histogram of residuals(prunedTree)
Frequency
0
2
4
6
8
−50
0
50
100
residuals(prunedTree)

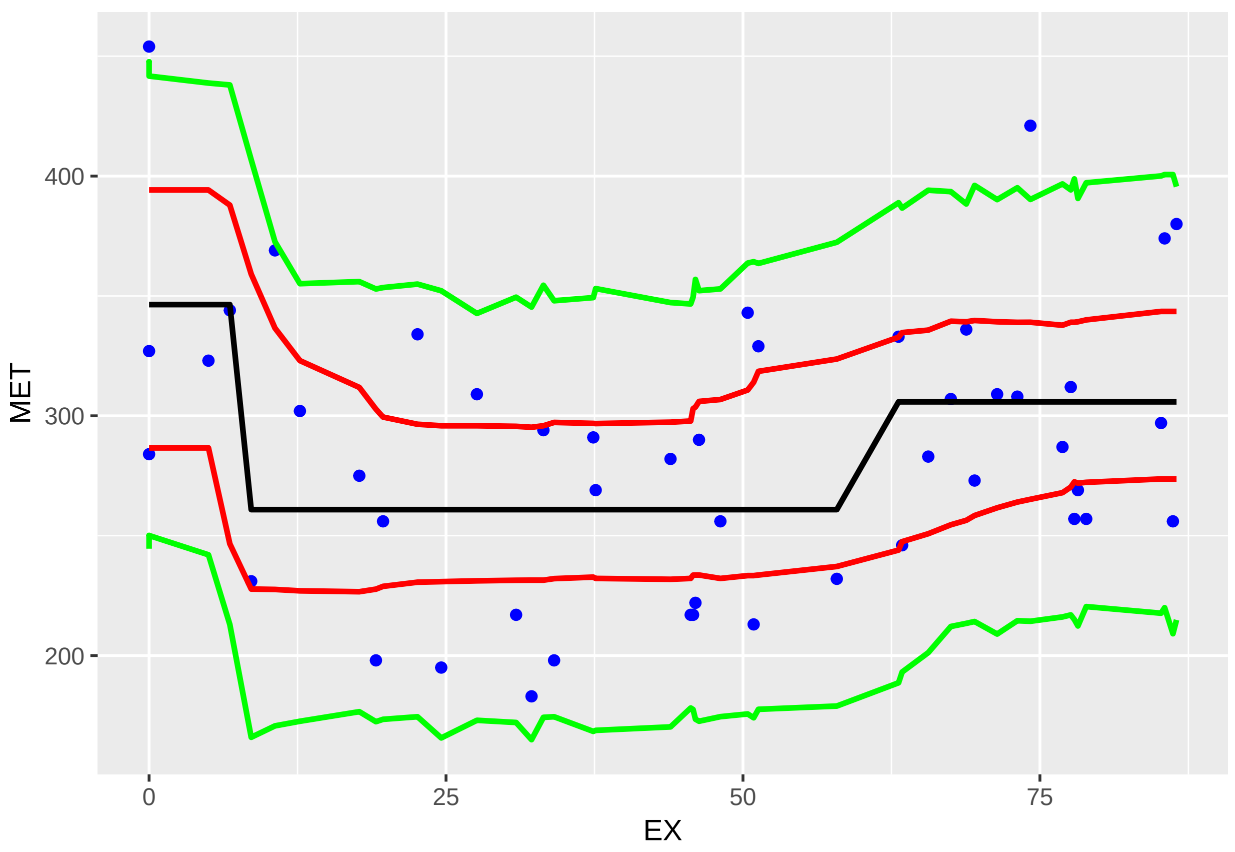
The tree with the size equal to the size of the tree with the least devience is selected. The distribution of residuals seen in the histogram plot goes in conjunction with the data points in the plot for predicting variable and actual (blue) and predicted (black) variable values. The model does seem to be a decent fit.

3



The confidence band is bumpy. This may be due to bias affecting the bootstrap. The results of the regression model in step 2 seems to be reliable since the prediction lies well within the confidence band.

4



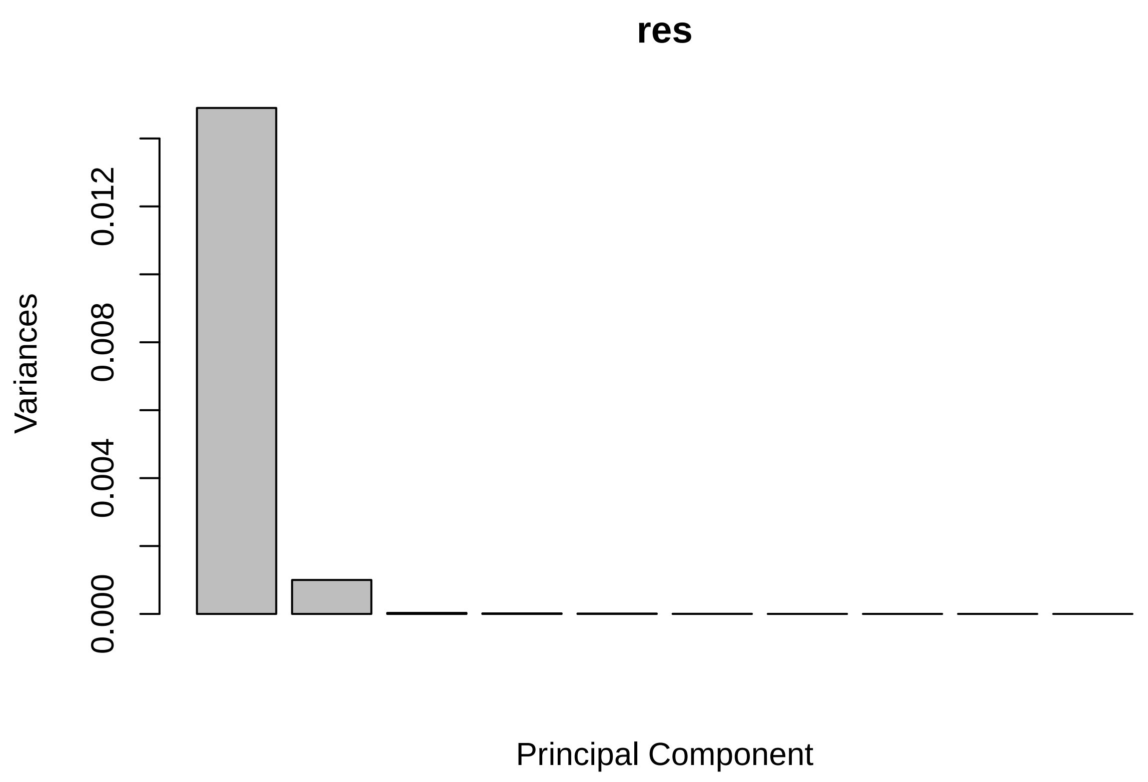
By considering the width of the confidence band and the results of the regression model in step 2 we can conclude that it seems to be reliable since the prediction curve lies well within the confidence band. It looks like not more than 5% of data are outside the prediction band because the prediction band is the band within which the actual value lies. It is so because we have considered 95% prediction interval.

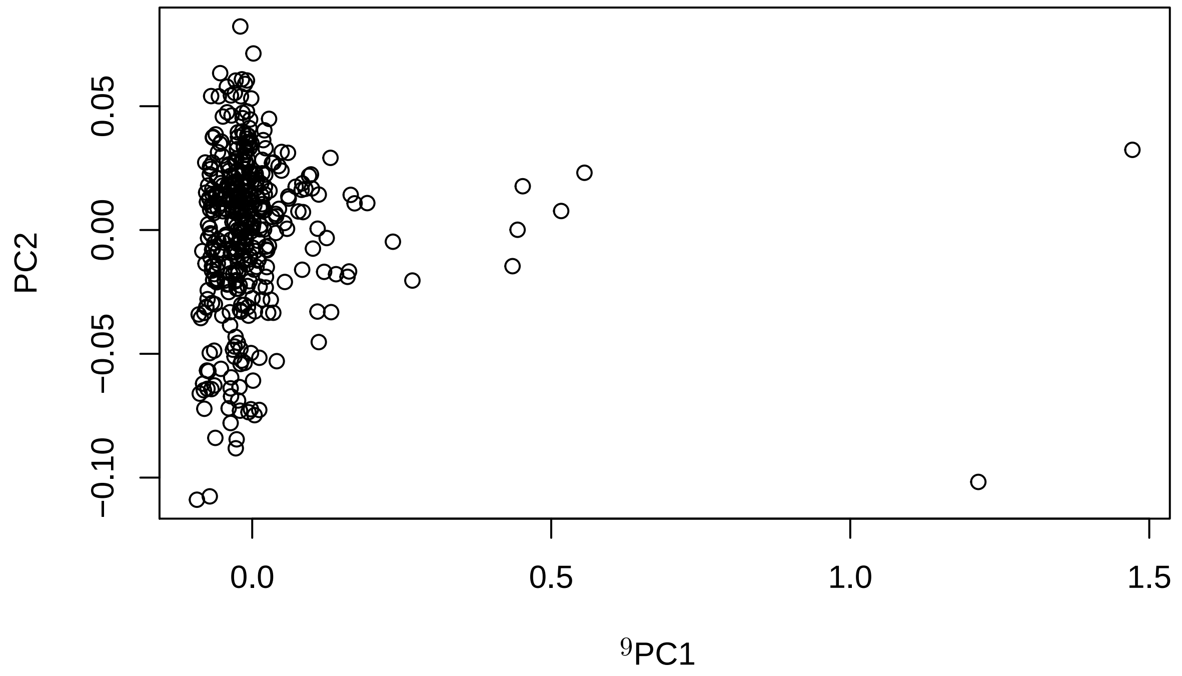
5

The Non-parametric bootstraping is more appropriate here since the number of observations is pretty low and the residuals are not normally distributed but the observations in parametric bootstrapping is normally distributed.

Assignment 4

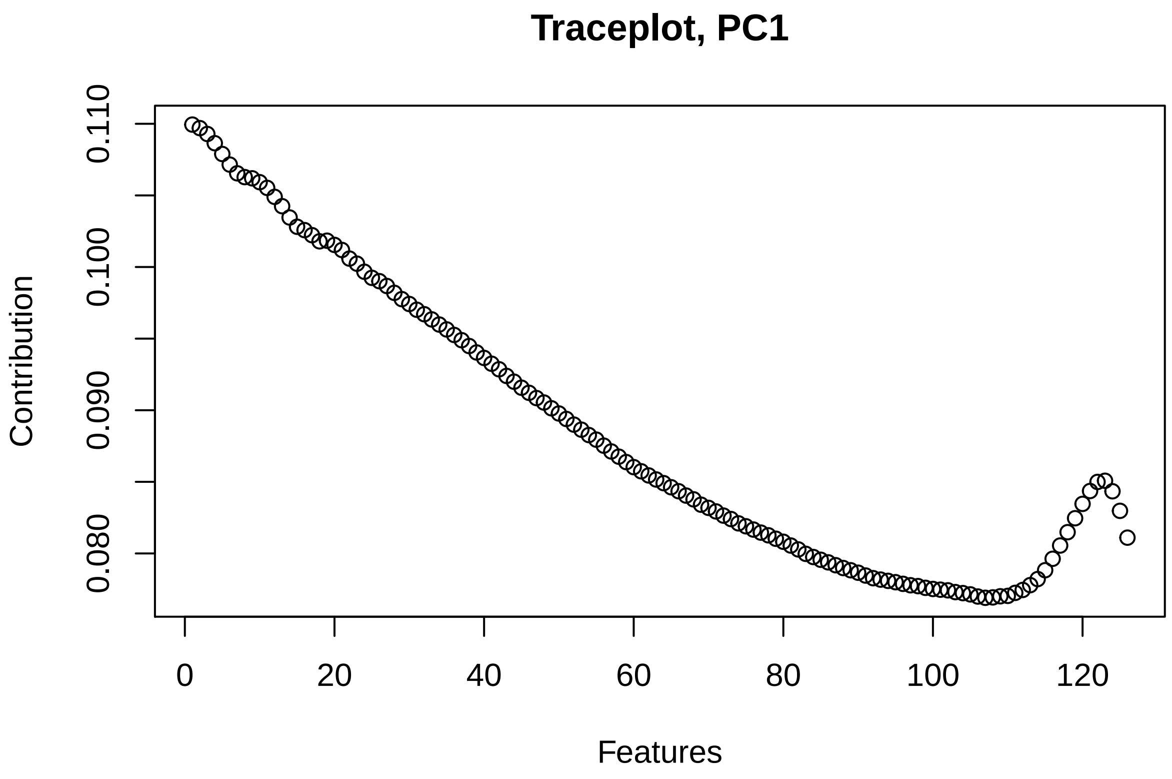
1



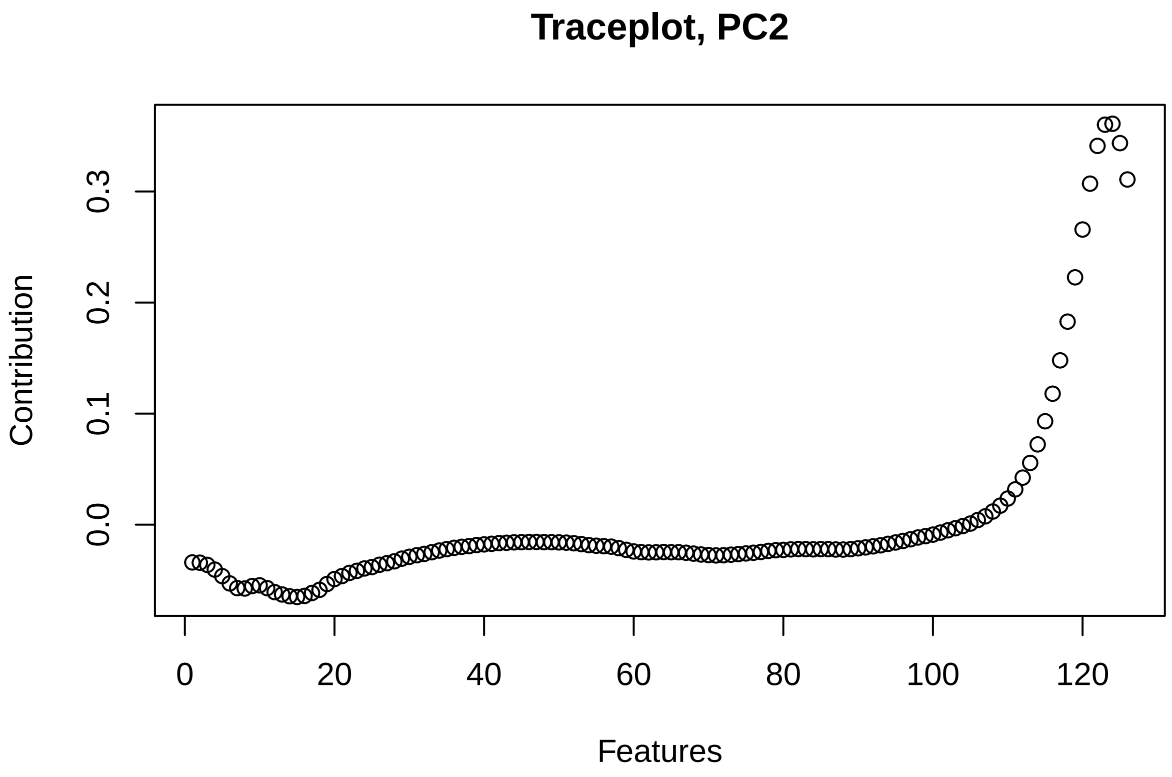


The first plot shows how many PC should be extracted. According to the plot the first 2 principal components should be extracted. Yes there are unusual diesel fuels which is visible as outliers in the second plot.

2



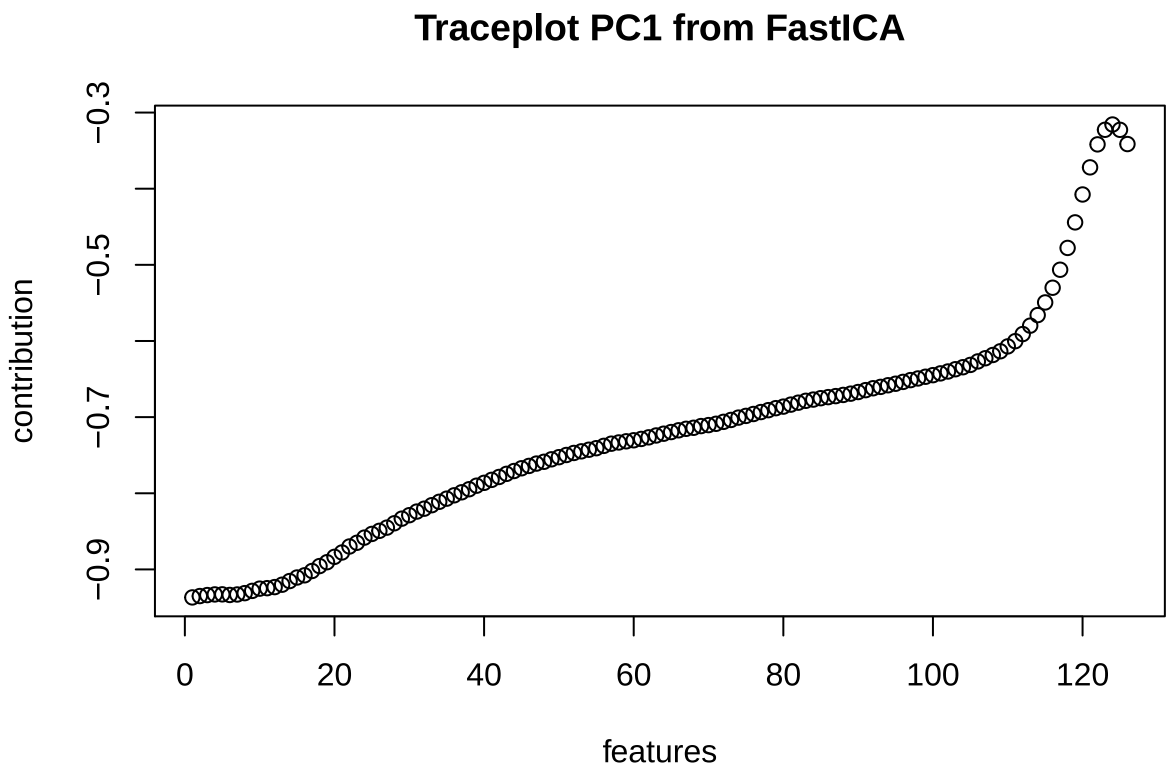
Traceplot, PC1
Contribution
0.110
0.100
0.090
0.080
0
20
40
60
80
100
120
Features

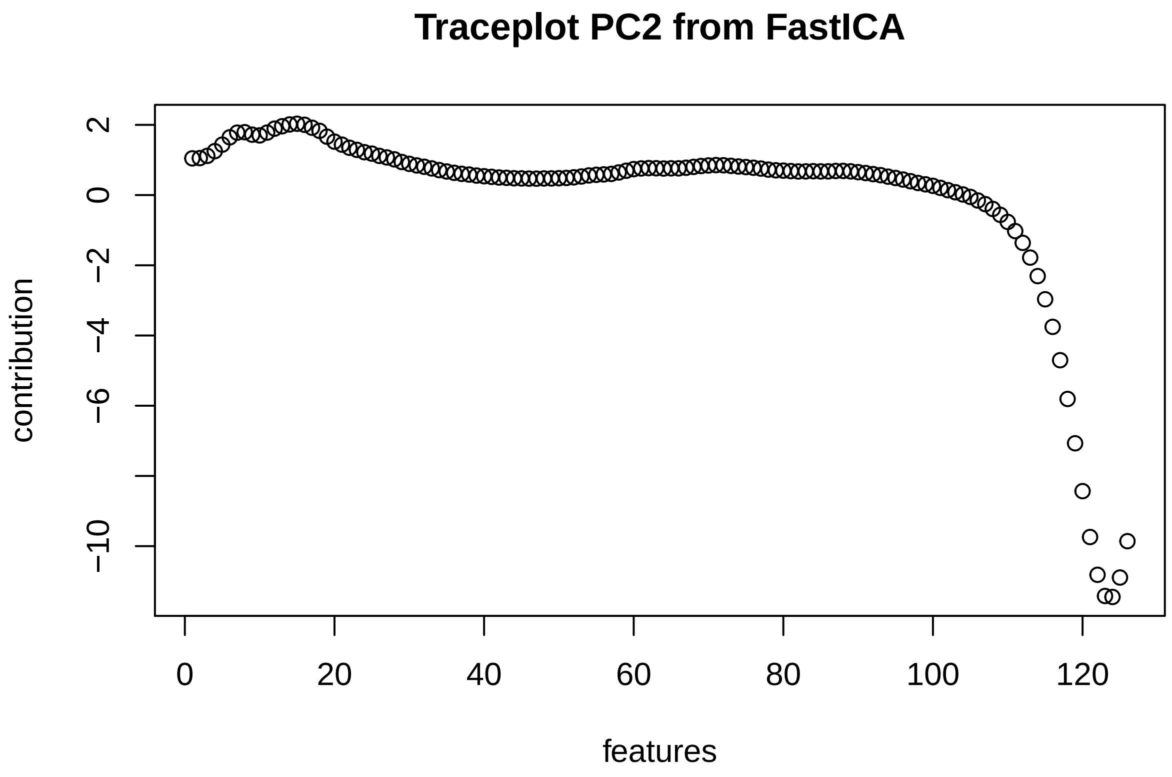


Traceplot, PC2
Contribution
0.3
0.2
0.1
0.0
0
20
40
60
80
100
120
Features

By looking at the second plot it can be seen that PC2 can be explaied by a few features which fall in the columns after 120 in the actual dataset i.e last 7-8 features.

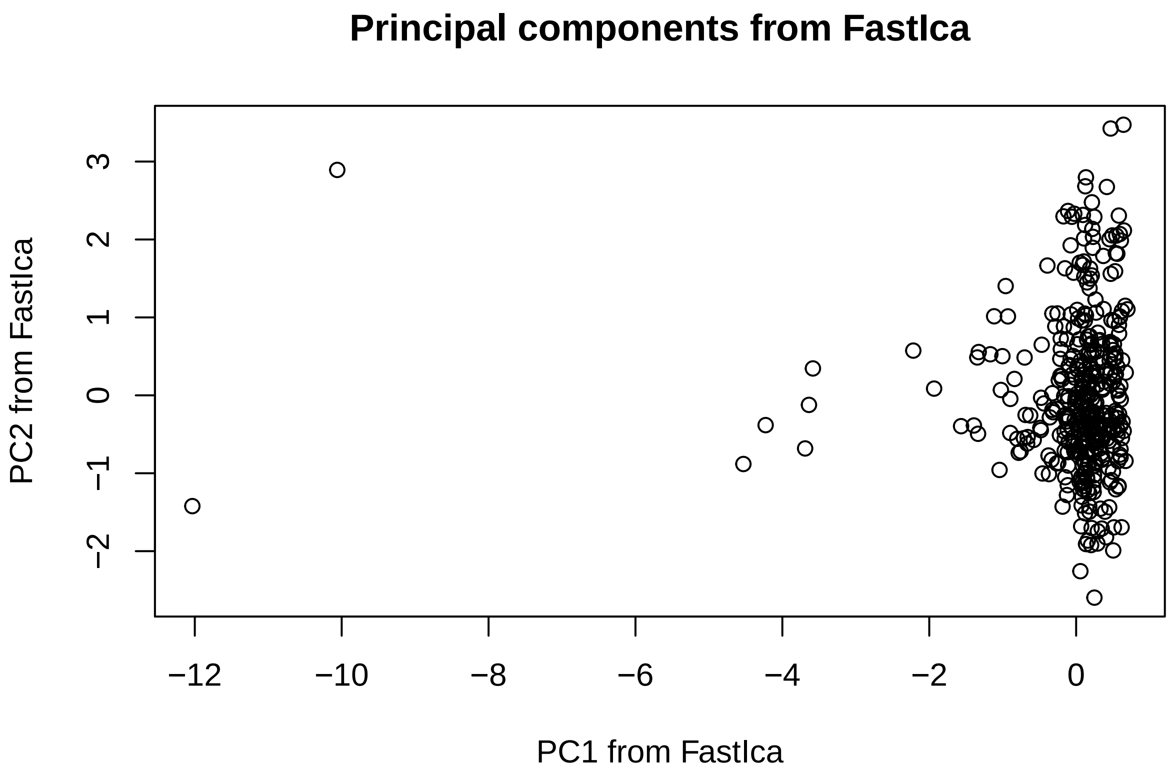
3a


Traceplot PC1 from FastICA
contribution
features
−0.3
−0.5
−0.7
−0.9
0
20
40
60
80
100
120


Traceplot PC2 from FastICA
contribution
features
2
0
−2
−4
−6
−10
0
20
40
60
80
100
120

By comparing this plot to the plot from Step 2 it can be observred that this trace plot for PC2 is the mirror image of the trace plot for PC2 from step 2 about the horizontal axis. W' in ICA is equivalent to the loadings in PCA.

3b



This plot seems to be an exact mirror image, about the vertical axis, of the score plot from step 1.

Appendix

```
#2.1

library(xlsx)
library(e1071)

data <- as.data.frame(read.xlsx("creditscoring.xls",sheetIndex = 1))
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,]
```

```r
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,]

#2.2

library(tree)
library(rpart)

#Fitting trees
tree_deviance <- tree(as.factor(good_bad) ~ ., data = train, split = "deviance")
tree_gini <- tree(as.factor(good_bad) ~ ., data = train, split = "gini")

#Predicting for train data
deviance_yfit <- predict(tree_deviance, newdata = train,type="class")
gini_yfit <- predict(tree_gini, newdata = train,type="class")

#Confusion matrix for train data
deviance_table <- table(deviance_yfit,train$good_bad)
gini_table <- table(gini_yfit,train$good_bad)

#Misclassification rate for train data
deviance_table
mr_deviance <- 1-sum(diag(deviance_table))/sum(deviance_table)
mr_deviance
gini_table
mr_gini <- 1-sum(diag(gini_table))/sum(gini_table)
mr_gini

#Predicting for test data
deviance_yfit_test <- predict(tree_deviance, newdata = test,type="class")
gini_yfit_test <- predict(tree_gini, newdata = test,type="class")

#Confusion matrix for test data
deviance_table_test <- table(deviance_yfit_test,test$good_bad)
gini_table_test <- table(gini_yfit_test,test$good_bad)

#Misclassification rate for test data
deviance_table_test
mr_deviance_test <- 1-sum(diag(deviance_table_test))/sum(deviance_table_test)
mr_deviance_test
gini_table_test
mr_gini_test <- 1-sum(diag(gini_table_test))/sum(gini_table_test)
mr_gini_test

#2.3

index <- summary(tree_deviance)[4]$size
dev_prunetree <- c(0)
```

```r
dev_pred <- c(0)

# Graph for train data and validation data
for(i in 2:index) {
  pruneTree <- prune.tree(tree_deviance,best=i) #pruning the tree that was fit
  predicted <- predict(pruneTree, newdata=valid,type="tree") #predicting using validation data
  dev_prunetree[i] <- deviance(pruneTree)
  dev_pred[i] <- deviance(predicted)
}

# Plot deviances against number of leaves for pruned tree and predictions
plot(2:index,dev_prunetree[2:index], col="Red",type = "b",
     ylim=c(min(dev_pred[2:index]),max(dev_prunetree))) #limiting the y axis to required values
points(2:index,dev_pred[2:index],col="Blue",type="b")

# Optimal tree and its depth is found out by looking at the graph
pruneTree2 <-  prune.tree(tree_deviance, best = 4)

# Predict using model with best
yfit <- predict(pruneTree2, newdata = test, type="class")
conMat <- table(test$good_bad,yfit)
conMat

# Misclassification rate for test data
1-sum(diag(conMat))/sum(conMat)

# Plot tree with best depth and the varaible at each node
plot(pruneTree2)
text(pruneTree2)

#2.4

nbTrain <- naiveBayes(good_bad ~ . , data=train)
nbTrainPredict <- predict(nbTrain, newdata = train[,-20])
nbTestPredict <- predict(nbTrain, newdata = test[,-20])

# Confusion Matrix Using Naive-Bayes for train data
nbConTrain <- table(train$good_bad,nbTrainPredict)

# Confusion Matrix Using Naive-Bayes for test data
nbConTest <- table(test$good_bad,nbTestPredict)

# Misclassification rate for train data
1-sum(diag(nbConTrain))/sum(nbConTrain)

# Missclassification rate for test data
1-sum(diag(nbConTest))/sum(nbConTest)

#2.5

rates <- data.frame(pi=double(), treeTpr=double(), treeFpr=double(), naiveTpr=double(), naiveTpr=double

for(pi in seq(0.05, 0.95, by=0.05)){
  treePrediction = as.data.frame(predict(pruneTree2, test))
  naivePrediction = as.data.frame(predict(nbTrain, test, type="raw"))

  treePi = ifelse(treePrediction$good > pi, "good", "bad")
  naivePi = ifelse(naivePrediction$good > pi, "good", "bad")

  treeTable = table(test$good_bad, factor(treePi, levels=c("bad", "good")))
  naiveTable = table(test$good_bad, factor(naivePi, levels=c("bad", "good")))

  rates = rbind(rates, c(pi,
  treeTable[4]/(treeTable[4]+treeTable[2]),
  treeTable[3]/(treeTable[3]+treeTable[1]),
  naiveTable[4]/(naiveTable[4]+naiveTable[2]),
  naiveTable[3]/(naiveTable[3]+naiveTable[1])
))
}

colnames(rates) = c("pi", "treeTpr", "treeFpr", "naiveTpr", "naiveFpr")

plot(-1, -1, xlim=c(0, 1), ylim=c(0, 1), xlab="FPR", ylab="TPR", main="ROC curve")
lines(rates$treeFpr, rates$treeTpr, lwd=2, col="blue")
lines(rates$naiveFpr, rates$naiveTpr, lwd=2, col="red")
legend("topleft", c("Tree", "Naïve"), col=c("blue", "red"), lwd=5)

#2.6

nbTrainPredict6 <- predict(nbTrain, train[,-20] , type="raw")
nbTestPredict6 <- predict(nbTrain, test[,-20] , type="raw")

nbtrain6 <- (nbTrainPredict6[, 2] / nbTrainPredict6[, 1]) > 10
nbtest6 <- (nbTestPredict6[, 2] / nbTestPredict6[, 1]) > 10

# Confusion matrix for train data
nbConTrain6 <- table(train$good_bad,nbtrain6)
nbConTrain6

# Confusion matrix for test data
nbConTest6 <- table(test$good_bad,nbtest6)
nbConTest6

# Missclasification rate for train data
1-sum(diag(nbConTrain6))/sum(nbConTrain6)

# Missclasification rate for test data
1-sum(diag(nbConTest6))/sum(nbConTest6)

#3.1

state <- read.csv("State.csv", sep = ";", dec = ",")
state <- state[order(state$MET),]

library(ggplot2)
```

```r
ggplot(state) +
  geom_point(aes(x=MET, y=EX)) +
  geom_smooth(aes(x=MET, y=EX), method="loess")

#3.2

library(tree)

set.seed(12345)

#set minimum number of observations in a leaf equal to 8
control <- tree.control(nobs = nrow(state),minsize = 8)
#tree model with target EX and feature MET
tree <- tree(formula = EX~MET,data = state,control = control)
#number of the leaves is selected by cross-validation
leaf <- cv.tree(tree)
#prune to get the best tree,size of the tree is the size of the tree with least devience
prunedTree <- prune.tree(tree, best = leaf$size[which.min(leaf$dev)])
#predict
fitValues <- predict(prunedTree, newdata = state)
#Plot the original and the fitted data
state$fitValues <- fitValues
ggplot(state, aes(EX, MET, fitValues)) +
  geom_point(aes(MET,EX), colour = "blue") + geom_point(aes(MET,fitValues))
#histogram of residuals
hist(residuals(prunedTree))

#3.3

library(boot)

# computing bootstrap samples
f <- function(state, ind){
# extract bootstrap sample
stateBoot <- state[ind,]
#set minimum number of observations in a leaf equal to 8
controlBoot <- tree.control(nobs = nrow(stateBoot),minsize = 8)
#tree model with target EX and feature MET
treeBoot <- tree(formula = EX~MET,data = stateBoot,control = controlBoot)
#prune to get the best tree,size of the tree is the size of the tree with least devience
prunedTreeBoot <- prune.tree(treeBoot, best = leaf$size[which.min(leaf$dev)])
#predict
return(predict(prunedTreeBoot, newdata = state))
}
#make bootstrap
res <- boot(state, f, R=1000)
conBand <- envelope(res, level = 0.95)
#extracting points for upper boundary of CI
state$upper <- conBand$point[1,]
#extracting points for lower boundary of CI
state$lower <- conBand$point[2,]
#plot actual & predicted values with confidence band
ggplot(state, aes(EX,MET,fitValues,lower,upper)) +
  geom_point(aes(MET,EX), colour = "blue") + geom_line(aes(MET,fitValues), lwd=1) +
  geom_line(aes(MET,lower), colour = "red", lwd=1) +
  geom_line(aes(MET,upper), colour = "red", lwd=1)

#3.4

#model <- prunedTree

rng <- function(state, model) {
  data1 <- data.frame(EX=state$EX, MET=state$MET)
  #generate new EX
  data1$EX <- rnorm(nrow(state),predict(model, newdata=state),sd(resid(model)))
  return(data1)
}

f1 <- function(state){
  control <- tree.control(nrow(state),minsize = 8)
  res <- tree(EX~MET, data=state, control=control) #fit tree model
  prunedTree <- prune.tree(res, best = leaf$size[which.min(leaf$dev)])
  #predict values for all MET values from the original data
  return(predict(prunedTree, newdata = state))
}

f2 <- function(state){
  control <- tree.control(nrow(state), minsize = 8)
  res <- tree(EX~MET, data=state, control=control) #fit tree model
  prunedTree <- prune.tree(res, best = leaf$size[which.min(leaf$dev)])
  #predict values for all MET values from the original data
  predF2 <- predict(prunedTree, newdata = state)
  return(rnorm(nrow(state),predF2,sd(resid(res))))
}

set.seed(12345)
boot4 <- boot(state, statistic=f1, R=1000, mle=prunedTree, ran.gen=rng, sim="parametric")
conBan <- envelope(boot4, level=0.95) #compute confidence bands

boot5 <- boot(state, statistic=f2, R=1000, mle=prunedTree, ran.gen=rng, sim="parametric")
preBan <- envelope(boot5, level=0.95) #compute prediction bands

# plot(boot4)
# plot(boot5)

#state$predF2 <- predF2
state$upperC <- conBan$point[1,]
state$lowerC <- conBan$point[2,]
state$upperP <- preBan$point[1,]
state$lowerP <- preBan$point[2,]

ggplot(state, aes(EX, MET, fitValues, upperC, lowerC, upperP, lowerP)) +
  geom_point(aes(MET,EX), colour = "Blue") +
  geom_line(aes(MET,fitValues), lwd=1) +
  geom_line(aes(MET,upperC), colour = "Red", lwd=1) +
  geom_line(aes(MET,lowerC), colour = "Red", lwd=1) +
  geom_line(aes(MET,upperP), colour = "Green", lwd=1) +
  geom_line(aes(MET,lowerP), colour = "Green", lwd=1)

#4.1

NIRSpectra <- read.csv2("NIRSpectra.csv", header = TRUE)

data4.1 <- NIRSpectra
data4.1$Viscosity <- c()
res <- prcomp(data4.1)
screeplot(res,xlab = "Principal Component")

#plot of scores
plot(res$x[,1],res$x[,2], xlab = "PC1", ylab = "PC2") #res$x[,1] &  res$x[,2] is pc1 & pc2 (respectivel

#4.2

U=res$rotation
plot(U[,1], main="Traceplot, PC1", xlab = "Features", ylab = "Contribution") #U[,1] loadings of PC1
plot(U[,2],main="Traceplot, PC2", xlab = "Features", ylab = "Contribution") #U[,2] loadings of PC2

#4.3a

library(fastICA)
set.seed(12345)

a <- fastICA(data4.1, 2, "parallel")
w <- a$K %*% a$W

plot(w[,1], main = "Traceplot PC1 from FastICA", xlab = "features", ylab = "contribution")
plot(w[,2], main = "Traceplot PC2 from FastICA", xlab = "features", ylab = "contribution")

#4.3b

s <- a$S
plot(s[,1], s[,2], main = "Principal components from FastIca",
     xlab = "PC1 from FastIca", ylab = "PC2 from FastIca")
```