Lab2 Block1

Yash Pawar

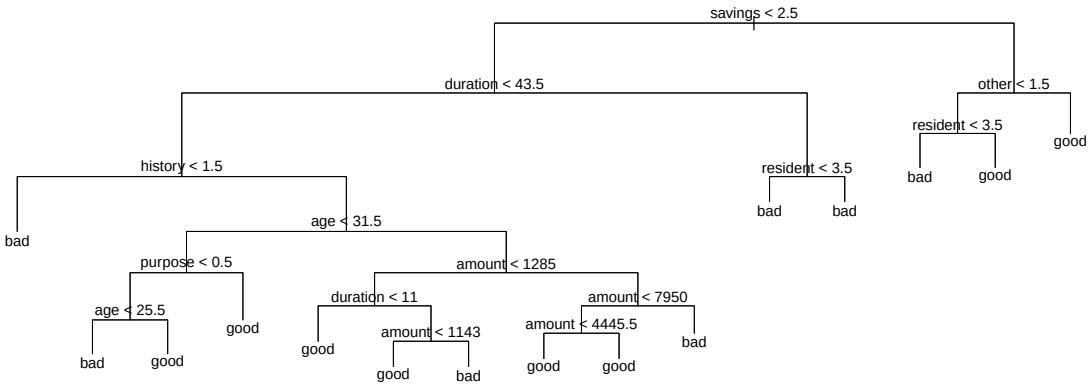
08/12/2019

Assignment 2.1

The data haas been split into training/validation/test as 50/25/25.

Assignment 2.2

The decision tree for the training data:



The misclassification rate for the decision tree with impurity Deviance

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

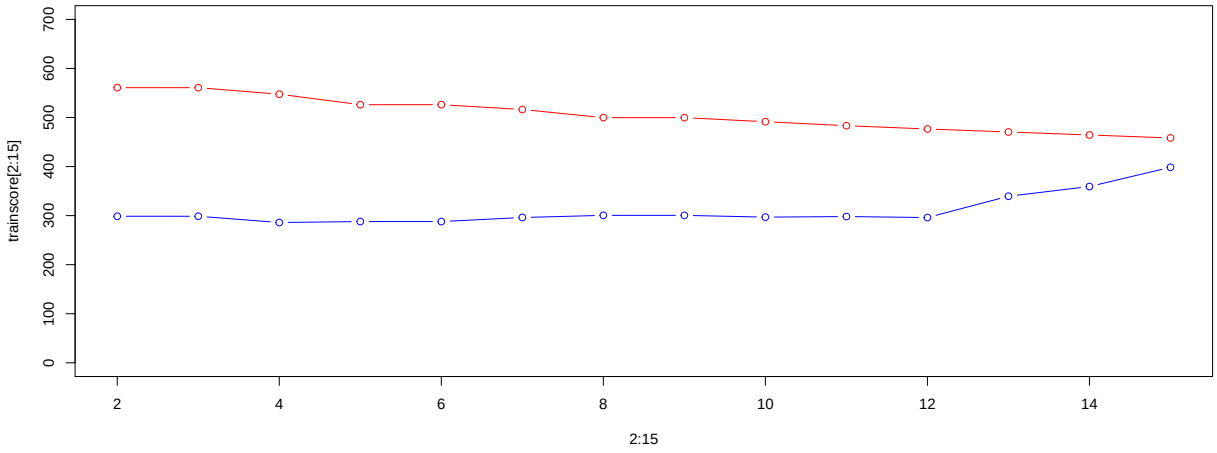
The misclassification rate for the decision tree with impurity Gini Index

```
##  bad good
##   52  198
```

```
## [1] 0.368
```

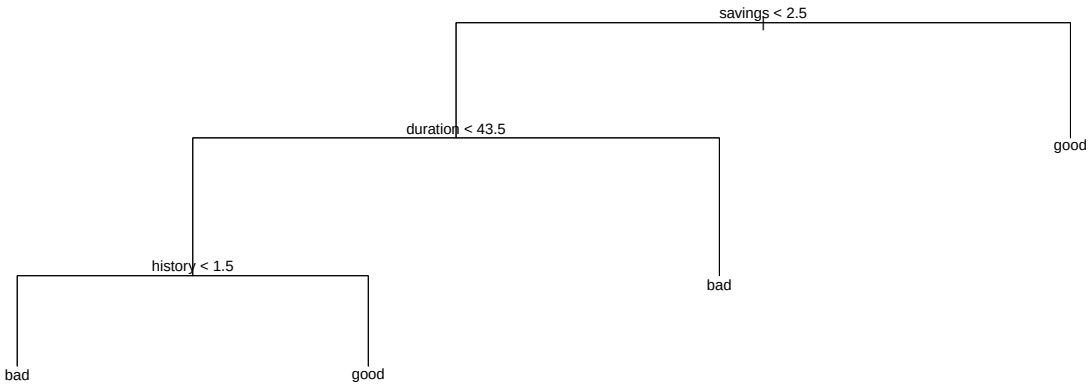
The misclassification rate for deviance is less than the misclassification for Gini Index.

Assignment 2.3



The plot for the best tree:

```
## [1] 4
```




```
## [1] 0.264
```

The optimal tree is the tree with minimum deviance which in this case is the 4th tree. The misclassification rate for the best tree is 0.264

Assignment 2.4

Misclassification rate for the naive bayes model:

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

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

The misclassification rate for naive bayes model is 0.3 which is higher than the optimal tree in case of Decision tree model.

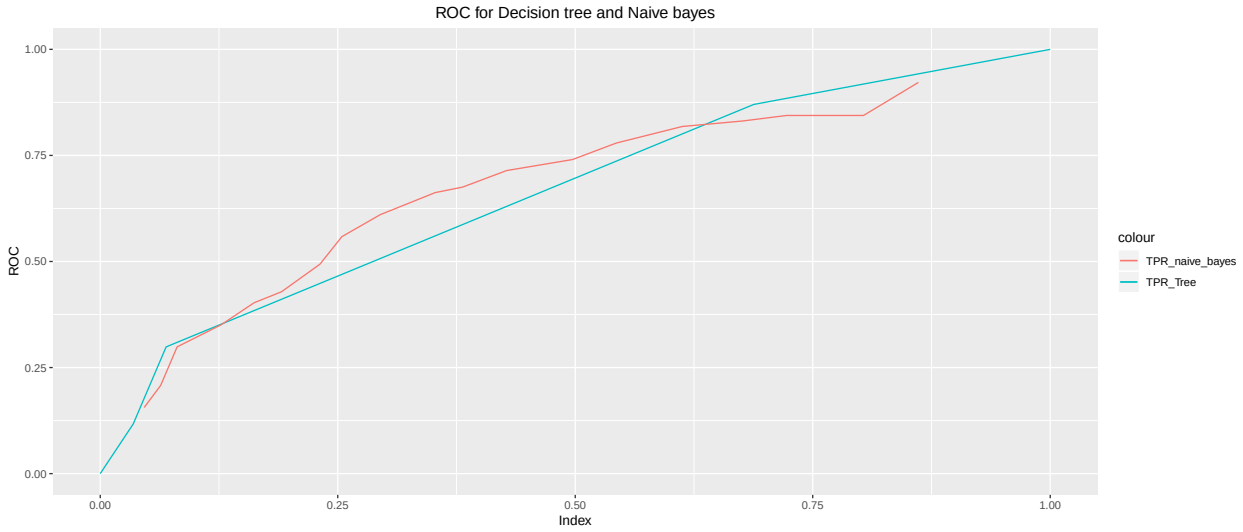
Misclassification rate for the testing data:

```
##       pred_naive_test
##        bad good
##   bad   46   30
##   good  49  125
```

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

Assignment 2.5

The ROC for the TPR for both decision trees and Naive Bayes model has been plot. The higher area under curve suggests better classifier. Thus in this case the AUC for the naive bayes model is higher than the decision tree model. It can be said that Naive bayes is a better classifier model for this particular dataset.



Assignment 2.6

The Naive Bayes classification according to the loss matrix is done, The corresponding confusion matrix for Train data:

```
##
## weighted_nb_predict bad good
##                bad  137  263
##                good  10   90
```

Misclassification rate for Training Data:

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

The Naive Bayes classification according to the loss matrix is done, The corresponding confusion matrix for Test data:

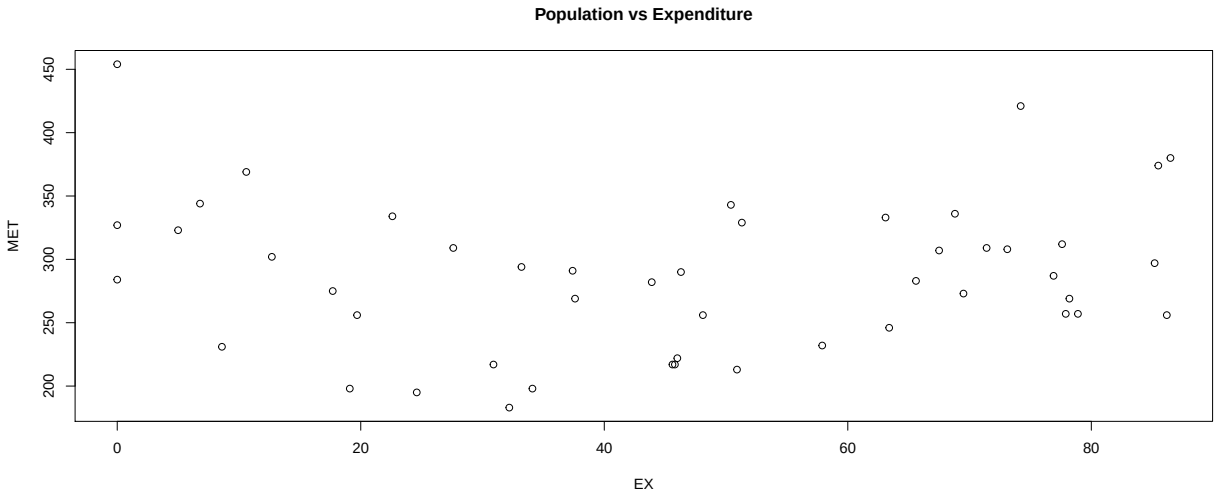
```
##
## weighted_nb_predict_test bad good
##                     bad   71  122
##                     good   5   52
```

Misclassification rate for Testing Data:

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

As compared to the Naive base model, the misclassification rate for the weighted model is higher. This is because the loss matrix adds weight to the misclassified components, The loss matrix here multiplies the predicted (bad) probablities with 10. because of which the comparitive value of bad probablity is increased as compared to the good.

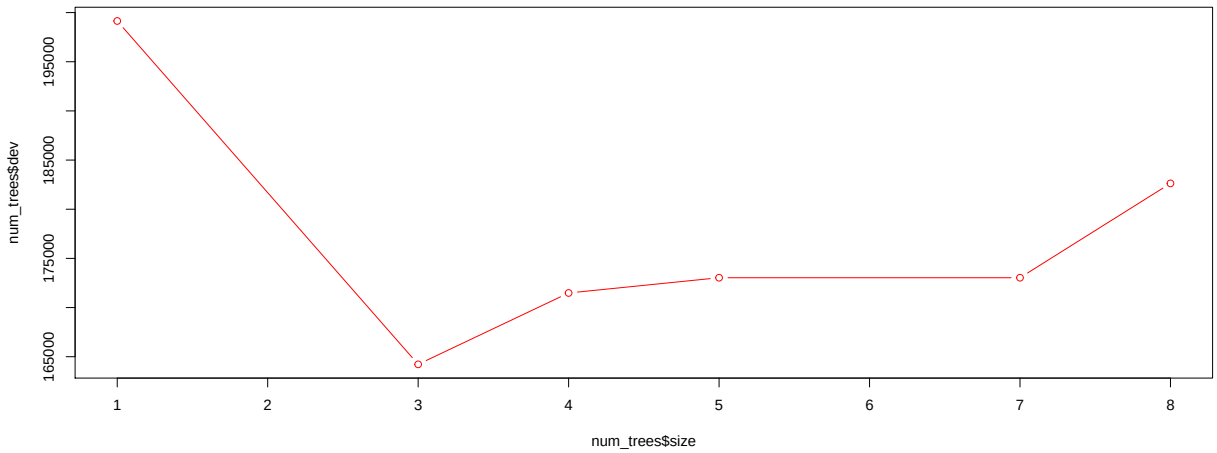
Assignment 3.1



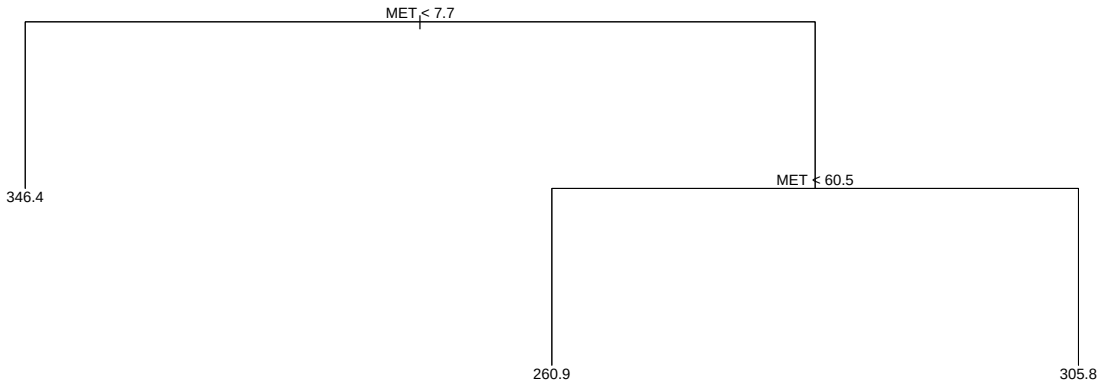
Given that there are two parameters, the model that would best describe this data would be decision tree as it would give the specifics of the expenditure of a paricular population based on the classification principle.

Assignment 3.2

The plot of deviance vs the tree size indicating that the best tree has 3 terminal nodes:

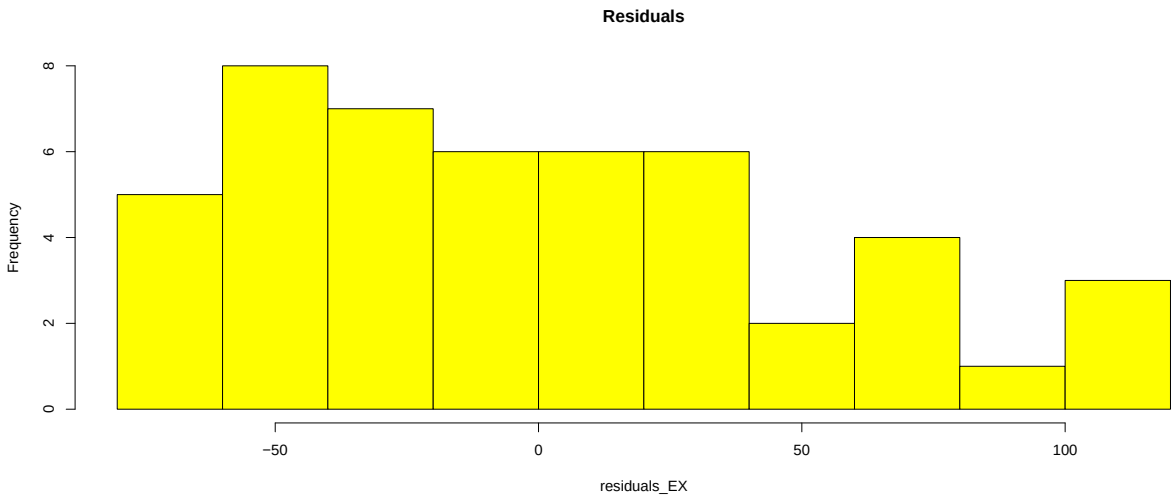


The tree with minimum deviance is selected, which in this case is 3. The tree is as follows:



The plot of original data vs Fitted data:

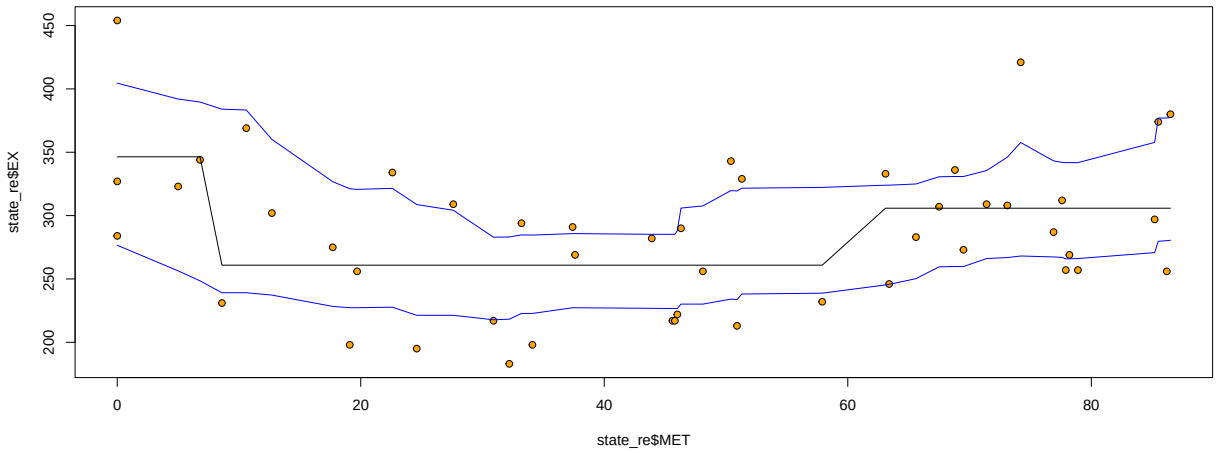
The histogram of residuals:



We can observe that the prediction is right skewed. The distribution of residuals resembles a Gamma distribution.

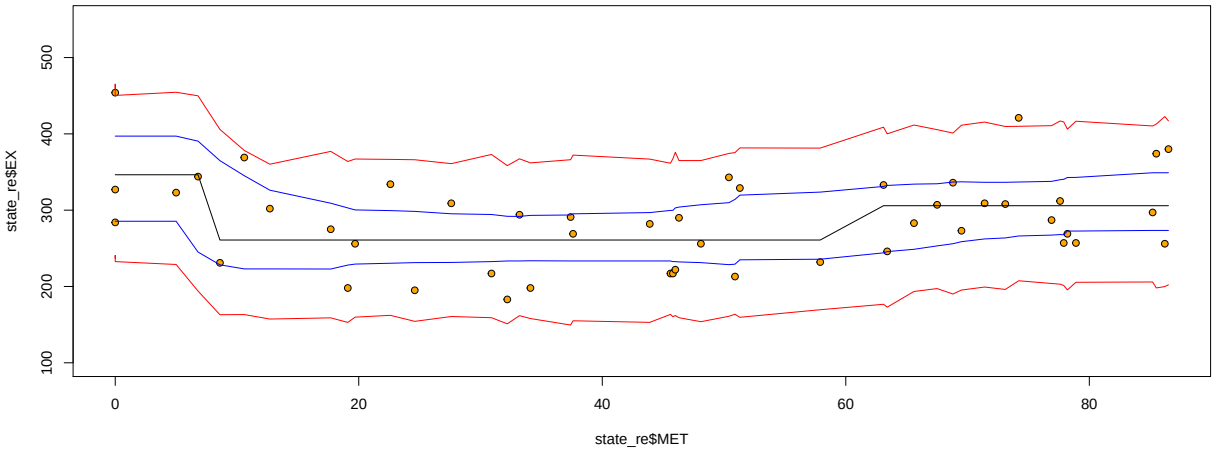
The predicted values are mostly uniform and thus has lot of variation with respect to the original data. Thus we can see that the residuals have high values.

Assignment 3.3



The confidence bands in case of non parametric bootstrap leave out a few outliers, this is because the resampling is done based on the discrete distribution of prediction values, thus we get a bumpy confidence bands as it is calculated based on just the prediction points which are dicrete in nature.

Assignment 3.4



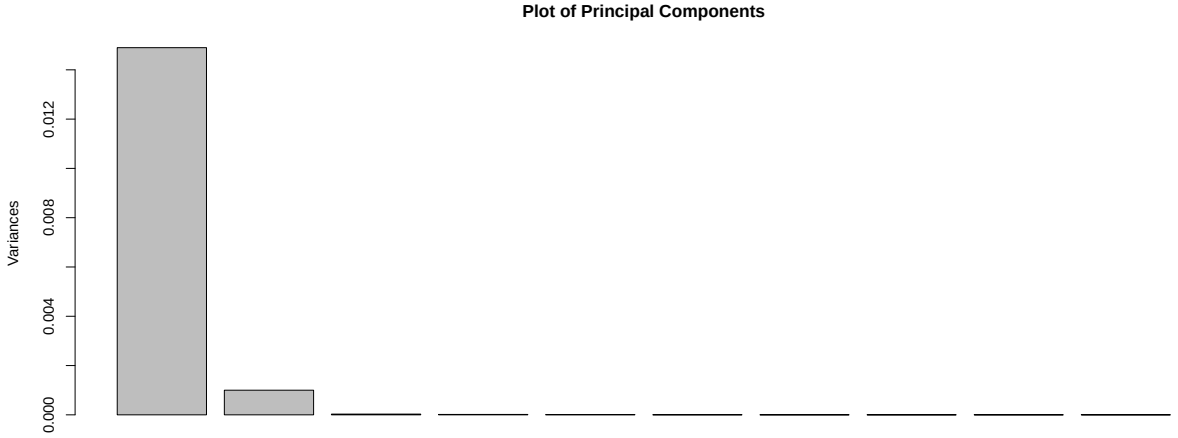
In case of parametric bootstrap, the predicitions are assumed to be normally distributed. This mean that the predicitons are assumed to be continuous. Hence it has lesser bumps and is smooth.

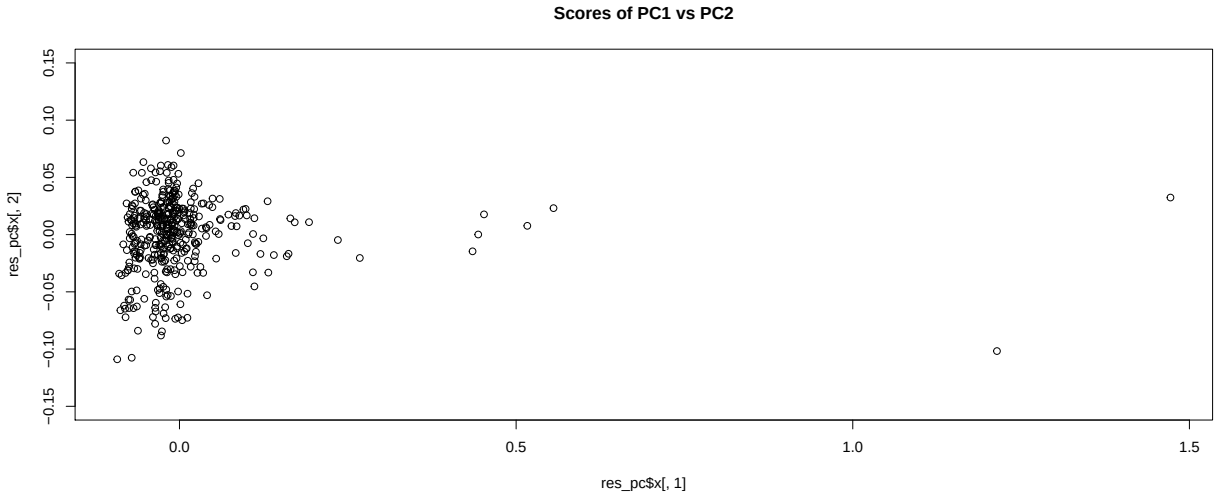
The plot above suggests that one point lies on the prediction bands and one lies outside which accounts to approximately 5% (2 out of 48 points). As the prediction bands are expected to cover 95% of the predictions

Assignment 3.5

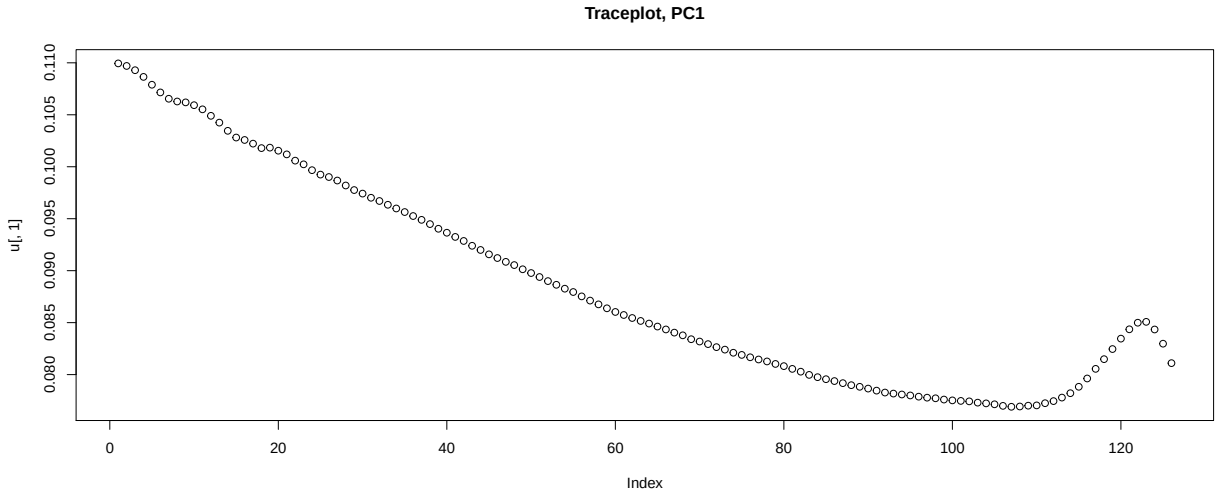
In parametric bootstrap we have assumed that the error is normally distributed which is a wrong assumption. Hence Parametric bootstrap is not a good choice. Thus, non parametric bootstrap is a more appropriate choice here.

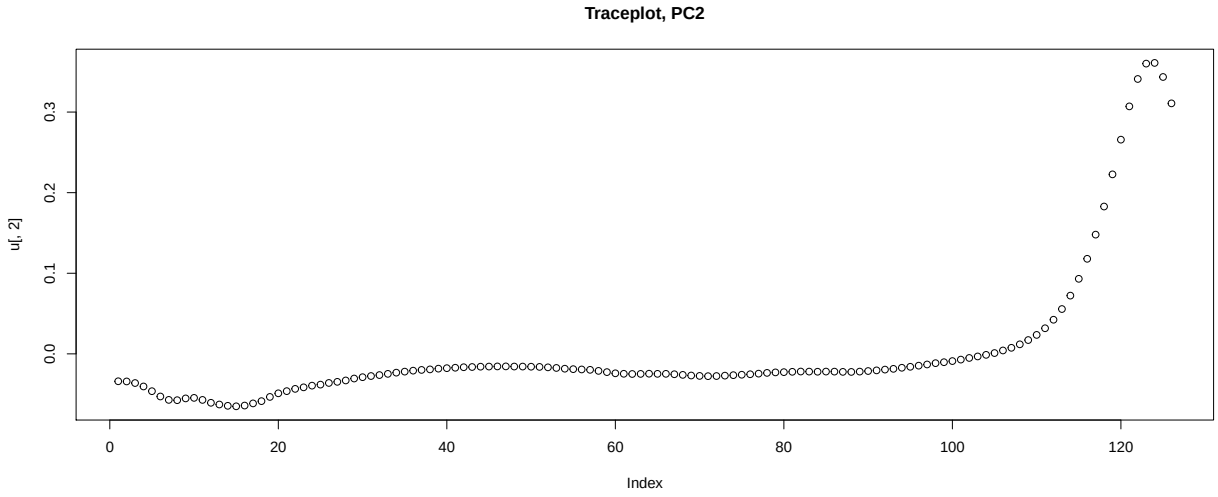
Assignment 4.1





Assignment 4.2



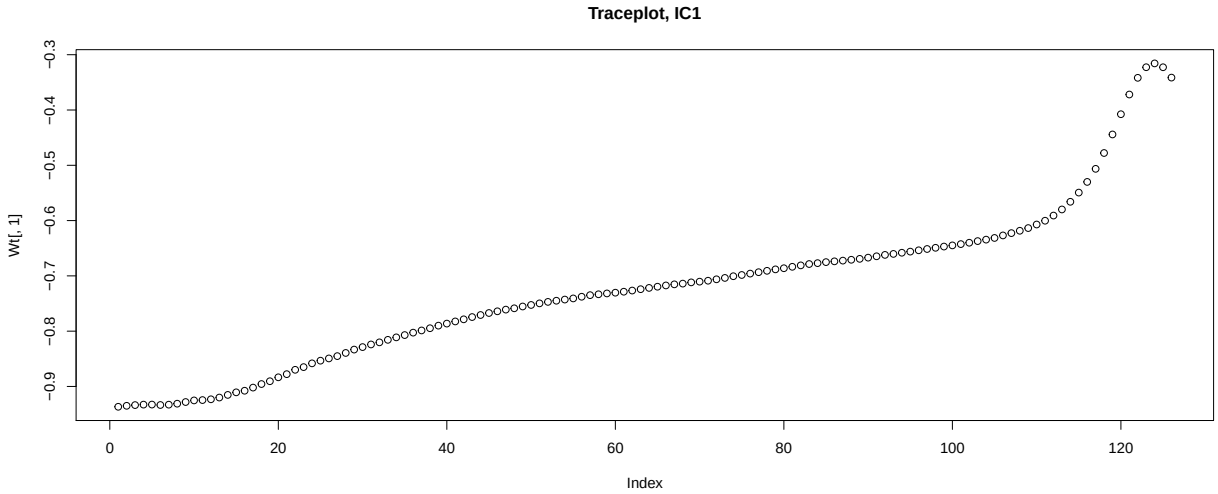


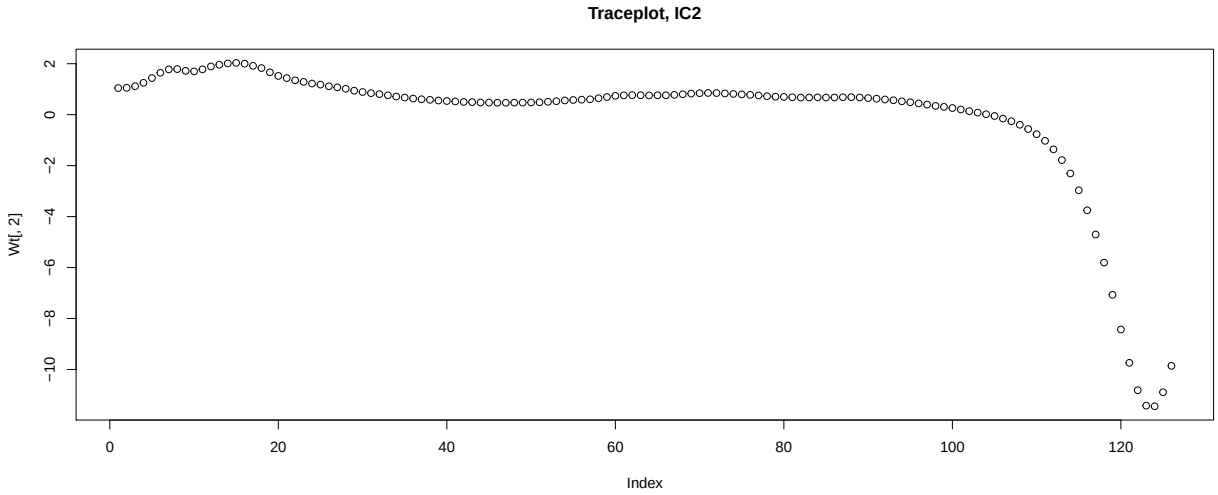
PC1 is explained by the first 100 (approximately) variables and few of the later components, The first 100 components are explained in almost equal measures and the few later ones are explained with some increased importance.

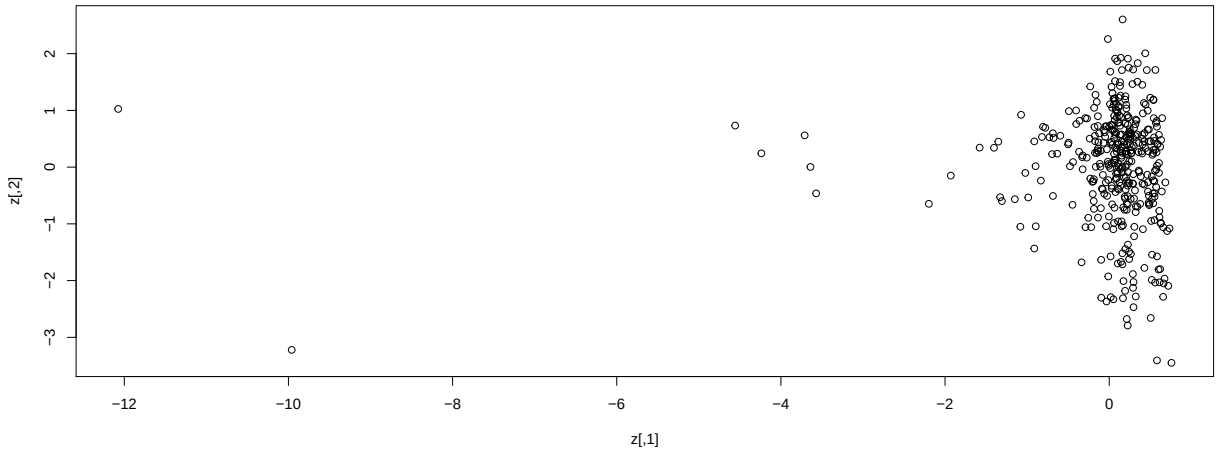
PC2 is explained by variables after approximately 110 with an increasing importance. The first 100 componts are not explained by PC2 very well.

Thus, it can be said that PC2 is explained by few original features.

Assignment 4.3







W' represents the transformation matrix. It is multiplied by the data matrix X which gives the ICA

components. The ICA algorithm estimates the matrix W which maximizes the negative entropy.

The traceplot of IC1 resembles the PC2 and the traceplot of IC2 resembles PC1. Thus data represented by variables transformed into Independent components resembles the PCs.

The scores in the plots of ICs are 180 degrees rotated as compared to the PCs, However both PC and IC perform dimentionality reduction in such a way that the first two components explain maximum variables.

Appendix:

```
knitr::opts_chunk$set(echo = TRUE,fig.width=14, fig.height=6)
## Importing Libraries
library(tree)
library(ggplot2)

## Data splitting
library(readxl)
creditscoring <- read_excel("creditscoring.xls")
creditscoring = cbind.data.frame(creditscoring[,-20], as.factor(creditscoring$good_bad))
colnames(creditscoring)[20] = "good_bad"
#View(creditscoring)
suppressWarnings(RNGversion("3.5.1"))
set.seed(12345)
n=dim(creditscoring)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
credit_train=creditscoring[id,]
id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.25))
credit_valid=creditscoring[id2,]
id3=setdiff(id1,id2)
credit_test=creditscoring[id3,]
#Training data on decision tree (impurity = deviance)
fit = tree(good_bad ~ ., data = credit_train, split = "deviance")
plot(fit)
text(fit, pretty = 0)
#summary(fit)
#predict for (impurity = deviance)
test_deviance = predict(fit, newdata = credit_test, type = "class")
#summary(test_deviance)
t1_deviance = table(test_deviance, credit_test$good_bad)
misclassification_deviance = 1 - sum(diag(t1_deviance))/sum(t1_deviance)
misclassification_deviance
#Training data on decision tree (impurity = Gini Index)
fit_gini = tree(good_bad ~ ., data = credit_train, na.action = na.exclude, split = "gini")
#plot(fit_gini)
#summary(fit_gini)

#predict for (impurity = Gini)
test_gini = predict(fit_gini, newdata = credit_test, type = "class")
summary(test_gini)
t1_gini = table(test_gini, credit_test$good_bad)
misclassification_gini = 1 - sum(diag(t1_gini))/sum(t1_gini)
```

```
misclassification_gini
trainscore=rep(0,15)
testScore=rep(0,15)
for(i in 2:15) {
  prunedTree=prune.tree(fit,best=i)
  pred=predict(prunedTree, newdata=credit_valid,
               type="tree")
  trainscore[i]=deviance(prunedTree)
  testScore[i]=deviance(pred)
}
plot(2:15, trainscore[2:15], type="b", col="red",
     ylim=c(0,700))
points(2:15, testScore[2:15], type="b", col="blue")


#Choosing optimal tree depth for validation
best_tree = which.min(testScore[-1])  + 1
##Optimal tree
best_tree
finalTree=prune.tree(fit, best=best_tree)
plot(finalTree)
text(finalTree, pretty = 0)

Yfit=predict(finalTree, newdata=credit_valid,
             type="class")
best_mat = table(credit_valid$good_bad,Yfit)
misclass_best = 1 - sum(diag(best_mat))/sum(best_mat)
misclass_best

## Problem 1.4
library(MASS)
library(e1071)


fit_naive = naiveBayes(good_bad~., data=credit_train)
#fit_naive

pred_naive = predict(fit_naive, credit_train,)
table_naive = table(credit_train$good_bad, pred_naive)
table_naive
misclass_naive = 1 - sum(diag(table_naive))/sum(table_naive)
misclass_naive
pred_naive_test = predict(fit_naive, credit_test,)
table_naive_test = table(credit_test$good_bad, pred_naive_test)
table_naive_test
misclass_naive_test = 1 - sum(diag(table_naive_test))/sum(table_naive_test)
misclass_naive_test

## Problem 1.5

pi = seq(0.05,0.95,0.05)

yfit_vector = predict(finalTree, newdata=credit_valid,
```

```
                          type="vector")

condition_tree = matrix(nrow = 250, ncol = length(pi))
col_good = which(credit_valid$good_bad=="good")
TPR_tree = c()
FPR_tree = c()
for (i in pi) {
  condition_tree[,i*20] = ifelse(yfit_vector[,2]>i,"good","bad")
  #col_pred_good[i*20] = length(which(condition_tree[,i]==credit_valid$good_bad))
  conf_matrix_good = table(factor(condition_tree[,i*20], levels = c("bad", "good")), credit_valid$good_b
  #print(conf_matrix_good)
  TPR_tree[i*20] = conf_matrix_good[1,1]/sum(conf_matrix_good[,1])
  FPR_tree[i*20] = conf_matrix_good[1,2]/sum(conf_matrix_good[,2])
}
yfit_model_nb = naiveBayes(good_bad~., data=credit_train)
yfit_nb = predict(fit_naive, newdata = credit_valid, type = "raw")

condition_nb = matrix(nrow = 250, ncol = length(pi))
TPR_nb = c()
FPR_nb = c()
for (i in pi) {
  condition_nb[,i*20] = ifelse(yfit_nb[,2]>i,"good","bad")
  #col_pred_good[i*20] = length(which(condition_tree[,i]==credit_valid$good_bad))
  conf_matrix_nb = table(factor(condition_nb[,i*20], levels = c("bad", "good")), credit_valid$good_bad)
  #print(conf_matrix_nb)
  TPR_nb[i*20] = conf_matrix_nb[1,1]/sum(conf_matrix_nb[,1])
  FPR_nb[i*20] = conf_matrix_nb[1,2]/sum(conf_matrix_nb[,2])
}
# plot(TPR, type="o", col="red")
# points(TPR_nb,type="o", col="green")

ROC_data_tree = cbind.data.frame(Index = seq(1,19,1),"TPR_tree" = TPR_tree, "FPR_Tree" = FPR_tree)
ROC_data_nb = cbind.data.frame(Index = seq(1,19,1),"TPR_Nb" = TPR_nb, "FPR_Nb" = FPR_nb)
ROC_data = merge(ROC_data_tree, ROC_data_nb)
ggplot(data = ROC_data) +
  geom_line(aes(x = FPR_tree, y = TPR_tree, color = "TPR_Tree")) +
  geom_line(aes(x = FPR_nb, y = TPR_Nb, color = "TPR_naive_bayes")) +
  ylab("ROC") + xlab("Index") + ggtitle("ROC for Decision tree and Naive bayes") +
  theme(plot.title = element_text(hjust = 0.5))




nb_predict_training = predict(fit_naive, newdata = credit_train, type = "raw")
weighted_good_nb_predict = nb_predict_training[,2]*1
weighted_bad_nb_predict = nb_predict_training[,1]*10
weighted_nb_predict = cbind(weighted_bad_nb_predict,weighted_good_nb_predict)
weighted_nb_predict = ifelse(weighted_good_nb_predict>weighted_bad_nb_predict, "good", "bad")
confusion_matrix_weighted_nb = table(weighted_nb_predict, credit_train$good_bad)
## Confusion matrix for Training data
confusion_matrix_weighted_nb
misclassification_weighted_nb = 1-(sum(diag(confusion_matrix_weighted_nb))/sum(confusion_matrix_weighte
misclassification_weighted_nb
```

```
nb_predict_testing = predict(fit_naive, newdata = credit_test, type = "raw")
weighted_good_nb_predict_test = nb_predict_testing[,2]*1
weighted_bad_nb_predict_test = nb_predict_testing[,1]*10
weighted_nb_predict_test = cbind(weighted_bad_nb_predict_test,weighted_good_nb_predict_test)
weighted_nb_predict_test = ifelse(weighted_good_nb_predict_test>weighted_bad_nb_predict_test, "good", "
confusion_matrix_weighted_nb_test = table(weighted_nb_predict_test, credit_test$good_bad)
## Confusion matrix for Training data
confusion_matrix_weighted_nb_test
## Mis classification rate test data
misclassification_weighted_nb_test =       1-(sum(diag(confusion_matrix_weighted_nb_test))/sum(confusio
misclassification_weighted_nb_test
library(readr)
library(tidyverse)
library(boot)
State <- read_csv2("State.csv")
#View(State)
state_re = arrange(State, MET)
plot(state_re$MET, state_re$EX, col = "black", xlab = "EX", ylab = "MET", main = "Population vs Expendi
tree_state = tree(EX ~ MET, state_re, control=tree.control(nobs = dim(state_re)[1],minsize=8))
# plot(tree_state)
# text(tree_state, pretty=0)
#fit
#summary(tree_state)
num_trees = cv.tree(tree_state)
#plot(num_trees$dev, type="b",col="red")
#plot(num_trees)
plot(num_trees$size, num_trees$dev, type = "b", col = "red")
#min(num_trees$dev)

prune_cv = prune.tree(tree_state, best = 3)
best_tree_predict = predict(prune_cv, state_re)
plot(prune_cv)
text(prune_cv, pretty = 0)
plot(state_re$EX, col = "red", type = "b", main = "Original Data vs Fitted Data")
points(best_tree_predict, col = "blue", type = "b")
residuals_EX = state_re$EX - best_tree_predict
hist(residuals_EX, col = "yellow", main = "Residuals")
statistic = function(data,ind){
  data1 = data[ind,]
  #state_re = data[order(State$MET),]
  tree_state = tree(EX ~ MET, data1, control=tree.control(nobs = dim(state_re)[1],minsize=8))
  prune_cv = prune.tree(tree_state, best = 3)
  pred_tree = predict(prune_cv, newdata=data)
  return(pred_tree)
}

set.seed(12345)
boot_model = boot(state_re,statistic = statistic, R=1000)
conf_interval = t(apply(boot_model$t, MARGIN = 2, quantile, probs = c(0.025, 0.975)))
#plot(conf_interval$overall)

#plotting the confidence interval
#tree_fit = tree(EX ~ MET, State, control=tree.control(nobs = dim(state_re)[1],minsize=8))
```

```
#tree_predict = predict(tree_fit,newdata = state_re, interval = "confidence")
plot(state_re$MET, state_re$EX, pch=21, bg="orange")
points(state_re$MET,best_tree_predict,type="l") #plot fitted line
#plot cofidence bands
points(state_re$MET,conf_interval[,2], type="l", col="blue")
points(state_re$MET,conf_interval[,1], type="l", col="blue")
state_data = state_re[, c('EX', 'MET')]

mle_tree = prune_cv

param_function_cb = function(boot_data){
  tree_state = tree(EX ~ MET, boot_data, control=tree.control(nobs = 48,minsize=8))
  prune_cv = prune.tree(tree_state, best = 3)
  pred_tree = predict(prune_cv, newdata=state_re)
  return(pred_tree)
}


param_function_pb = function(boot_data){
  tree_state = tree(EX ~ MET,
                    boot_data,
                    control=tree.control(nobs = nrow(boot_data),minsize=8))

  prune_cv = prune.tree(tree_state, best = 3)
  pred_tree = predict(prune_cv, newdata=state_data)
  n=length(state_re$EX)
  resid = residuals(mle_tree)
  predictedP=rnorm(n, pred_tree, sd(resid))
  return(predictedP)

}




rng_tree = function(data_rng, mle = mle_tree){
  data_1 = data.frame(EX = data_rng$EX, MET = data_rng$MET)
  n = length(data_rng$EX)
  pred_rng = predict(mle, data_1)
  resid = residuals(mle)
  data_1$EX = rnorm(n, pred_rng, sd(resid))
  return(data_1)
}




boot_pb = boot(state_data, statistic = param_function_pb, R = 1000, mle = mle_tree,
               ran.gen = rng_tree, sim = 'parametric')

boot_cb = boot(state_data,statistic = param_function_cb,R = 1000, mle = mle_tree,
               ran.gen = rng_tree, sim = 'parametric')
```

```r
cb_param = envelope(boot_cb, level = 0.95)
#pb_param = envelope(boot_pb)

pb_param = t(apply(boot_pb$t, MARGIN = 2, quantile, probs = c(0.025, 0.975)))

plot(state_re$MET, state_re$EX, pch=21, bg="orange", ylim = c(100,550))
points(state_re$MET,best_tree_predict,type="l") #plot fitted line
#plot cofidence bands
points(state_re$MET,cb_param$point[2,], type="l", col="blue")
points(state_re$MET,cb_param$point[1,], type="l", col="blue")
points(state_re$MET,pb_param[,2], type="l", col="red")
points(state_re$MET,pb_param[,1], type="l", col="red")

library(readr)
NIRSpectra <- read_csv2("NIRSpectra.csv")
#View(NIRSpectra)
NIRSpectra$Viscosity = c()
res_pc = prcomp(NIRSpectra)
lambda = res_pc$sdev^2
#eigenvalues
#lambda
#proportion of variation
sp = sprintf("%2.3f",lambda/sum(lambda)*100)
screeplot(res_pc, main = "Plot of Principal Components")
sp = as.numeric(sp)
sp[1]+sp[2]

u = res_pc$rotation
#head(u)
plot(res_pc$x[,1], res_pc$x[,2], ylim=c(-0.15,0.15), main = "Scores of PC1 vs PC2")

plot(u[,1], main="Traceplot, PC1")
plot(u[,2],main="Traceplot, PC2")
#ICA
library(fastICA)
set.seed(12345)
ica = fastICA(NIRSpectra,2)
Wt = ica$K %*% ica$W
#Wt
plot(Wt[,1], main = "Traceplot, IC1")
plot(Wt[,2], main = "Traceplot, IC2")

#plot latent features
v = solve(ica$W)
z = ica$X %*% Wt
z = z %*% v
plot(z)
```