Lab2_Block1

Andreas Stasinakis

November 27, 2018

Assingment 2

2.1

Spliting the data

```r
#Spliting the data in Training, validation and test data.
data = read_excel("../dataset/creditscoring.xls")
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,]
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

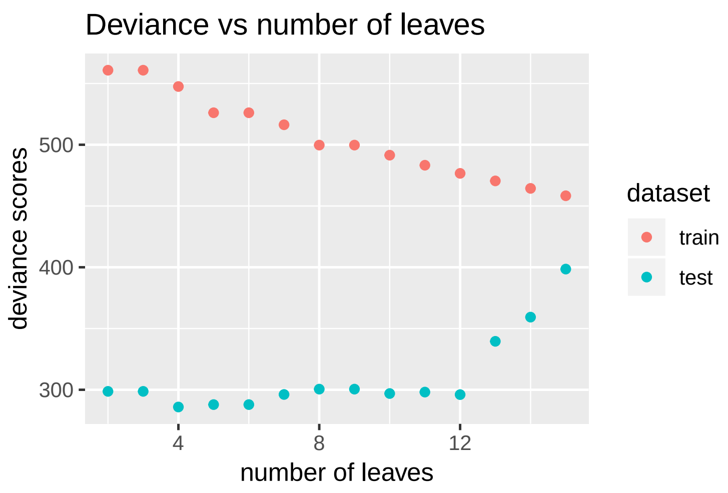
Table 1: misclassification rates

	train	test
Deviance	0.212	0.268
Gini	0.240	0.368

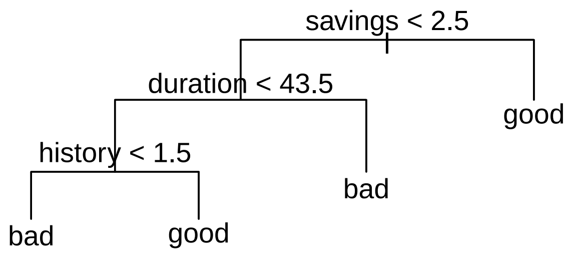
Analysis

In this task we fit a decision tree by using two different measures of impurity(Deviance and Gini). From the misclassification rates we can see that the decision tree we fit with deviance provides better results than the gini for both training and test data.

2.3



```
## $`Misclassification rate for optimal tree`
## [1] 0.256
```



```
## $`structure of the optimal tree`
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
## 1) root 494 598.00 good ( 0.2935 0.7065 )
##   2) savings < 2.5 346 446.70 good ( 0.3468 0.6532 )
##     4) duration < 43.5 325 405.90 good ( 0.3169 0.6831 )
##       8) history < 1.5 22  27.52 bad ( 0.6818 0.3182 ) *
##       9) history > 1.5 303 365.10 good ( 0.2904 0.7096 ) *
##     5) duration > 43.5 21  20.45 bad ( 0.8095 0.1905 ) *
##   3) savings > 2.5 148 134.40 good ( 0.1689 0.8311 ) *
```

Table 2: confusion matrix for test data

	bad	good
bad	18	6
good	58	168

Analysis

From the first plot is not that clear but the optimal tree is for $i = 4$, which is a tree with 4 terminal nodes. The variables used by the tree, as we can see from the second plot, is only 3 (“Savings”, “duration”, “history”). Both from the structure of the tree and the plot we can conclude that from these 3 predictors, “savings” is the most important because is the only variable which can directly classify. More specific, if one’s savings are more than 2.5, they can automatically be classified as “good” customers. Moreover, if their savings are less than 2.5, duration plays important role in their classification. Finally the last node is history where they classified as “bad customers” if the variable “history” is less than 1.5 or “good” if it is more.

The misclassification rate for the test data is 0.256 which is slightly less than the misclassification rate in the task above,which is logical because in this task we calculate the misclassification rate for the optimal tree.

2.4

Table 3: confusion matrix for train data for Naive

	bad	good
bad	95	98
good	52	255

Table 4: confusion matrix for test data for Naive

	bad	good
bad	46	49
good	30	125

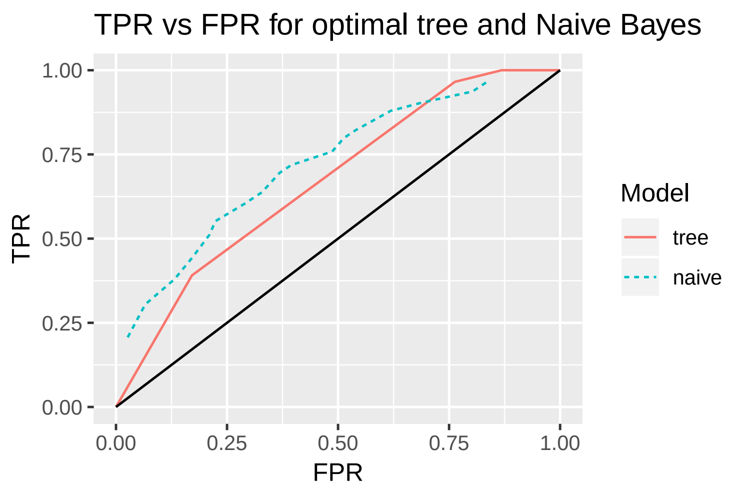
Table 5: misclassification rates for Naive

misclassification.rate.train.	0.300
misclassification.rate.test.	0.316

Analysis

From the confusion matrices it can be said that naive’s model makes quite enough wrong predictions for both the training and the test data. The proportion of misclassification rate is almost the same for the two data sets, if we take into consideration that the training data is double than the test data. One interesting notation is that “bad” customers who misclassified as “good” are almost double than the other way around. This seems bad because the private enterprise gave loans to people who in the end could not pay back. In the contrary, the optimal tree classifies “bad” customers as “good” only 6 times. This detail makes the difference for this model, despite the fact that the misclassification rates between the tree and the naive are not that different. In conclusion, the private enterprise should choose the optimal tree because not only has lower misclassification rate than the naives, but also the customers whom misclassified are not that costly for the business.

2.5



Analysis

In this task we use different principle to classify the test data both using the optimal tree and the Naive Bayes classifier and we plot the ROC curve for them. In general, the best classifier is the one which when FPR is the same for all models, its TPR is higher. Therefore, the classifier with the greatest area under its curve (also called AUC) is the best one. In this case the best classifier is Naive Bayes in contrast to the tasks above.

2.6

Table 6: confusion matrix for train data

	bad	good
bad	137	10
good	263	90

Table 7: confusion matrix for test data

	bad	good
bad	71	5
good	122	52

Table 8: misclassification rates

misclassification.rate.train.	0.546
misclassification.rate.test.	0.508

Analysis

In this task we implement naive Bayes classification but we use as loss function :

$$L = \text{Observed} \begin{array}{c c} & \textit{Predicted} \\ \begin{array}{c} \textit{good} \\ \textit{bad} \end{array} & \begin{pmatrix} 0 & 1 \\ 10 & 0 \end{pmatrix} \end{array}$$

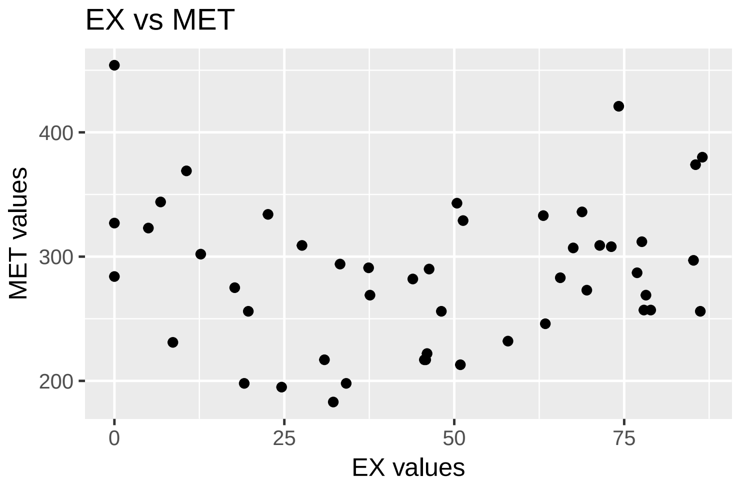
Using that loss function, we change the weight of each misclassification. Moreover, every "bad" customer who misclassified as "good" costs 10 but when a "good" customer misclassified as "bad" costs only 1. Using this kind of loss function makes total sense in some cases because the two types of misclassification have different impact to the problem.

One logical result of this loss function is that the misclassificaiton rates for both training and test data are almost double than in task 2.4 in which we do not use this loss function. For example if the probability of a customer being "bad" is 0.3 and the probability of being "good" is 0.7, the model in 2.4 classifies them as "good". On the other hand, if we use the above function to classify the customer the first probability will be $0.3 * 10 = 3$ as a result the customer will be classified as "bad".

Despite the fact that the misclassification rate is quite high, using this loss function for this model is a really good idea. As mentioned before, in 2.4 many "bad" customers misclassified as "good" which is really costly for the company because many citizens will get loan that they can not pay back. On the contrary, using this loss function eliminates this kind of error but of course the company may lose "good" customers who misclassified as "bad". In conclusion, using a loss function really makes sense in this problem but we should consider maybe a different cost in order to reduce the misclassification rate.

Task 3

3.1



Analysis

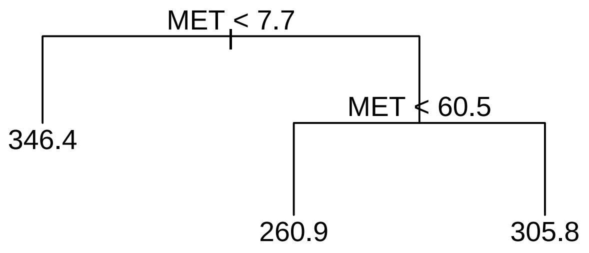
In general, it is not possible to take a decision about the appropriate model for a dependent variable only from a plot. We have to search deeper, fit maybe the data in different models, calculate their errors and after that decide. If i had to choose one model from the plot though, i believe that a quadratic model of the form

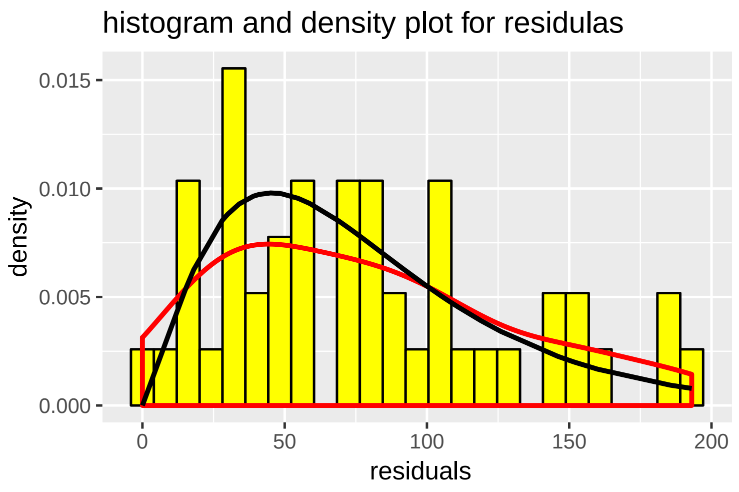
$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \varepsilon.$$

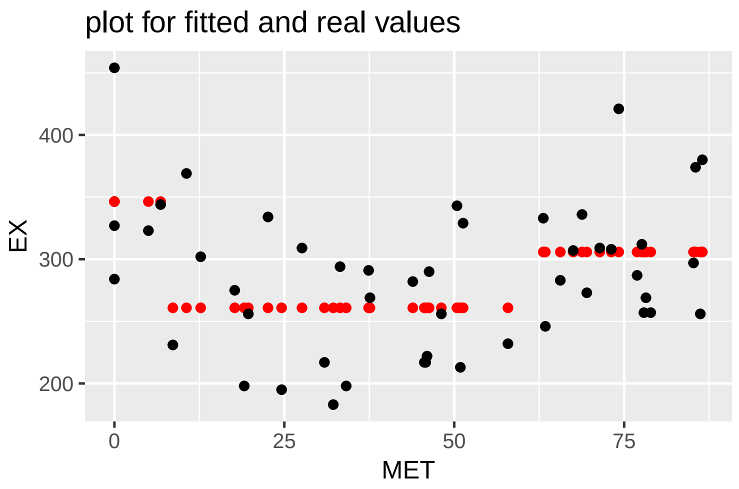
could fit the data efficient enough. Of course maybe other methods such as decision trees or else.

3.2

```
##
## Regression tree:
## snip.tree(tree = tree1, 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
```







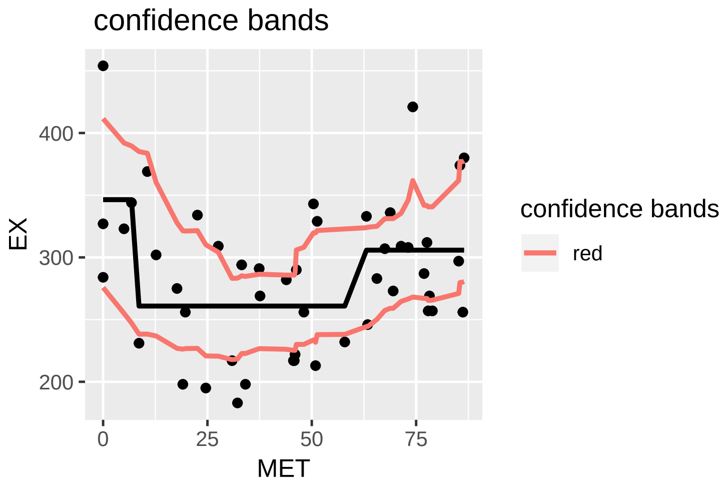
Analysis

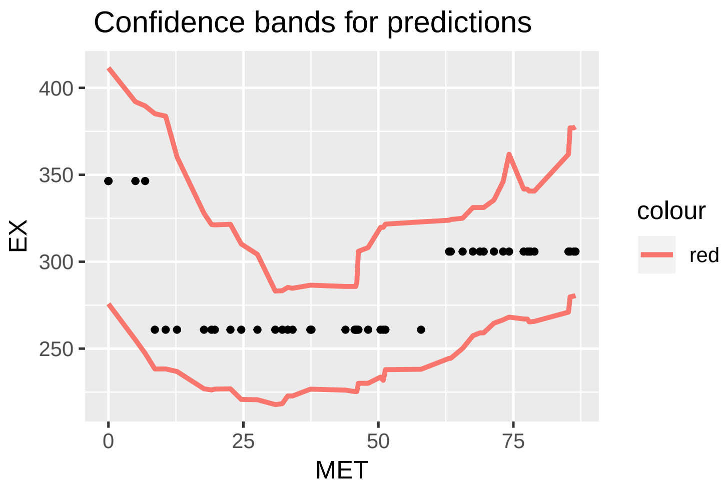
In this task we fit a regression tree with target EX and only one predictor MET. We also perform cross validation in order to find the optimal number of leaves given that the minimum number of observations should be 8. From the summary we can see that the optimal tree has 3 terminal nodes. We also plot the original and the fitted data vs the predictor MET and also the density of the residuals.

From the first plot it is obvious that we have only 3 different prediction values(346.4, 260.88, 305.8333) despite the fact that our target value is continuous. Although these predictions may look inaccurate, we have to mention that the reason why this is happening is the way we fit the tree and because we use cross validation to take the optimal number of nodes. As mentioned before, we set as minimum number of observations in a leaf equal to 8 and the optimal nodes are 3.

Moreover, in the residuals' plot, we also plot a Gamma distribution in order to show their similarities. Therefore, in this case the residuals' distribution is Gamma with parameters $\alpha = 2.3983$ and $\beta = 0.03079$. About the quality of fit we can say that it is not that good. This is why, as mentioned before, the residuals are Gamma distributed while it would be much better if they are normally distributed.

3.3

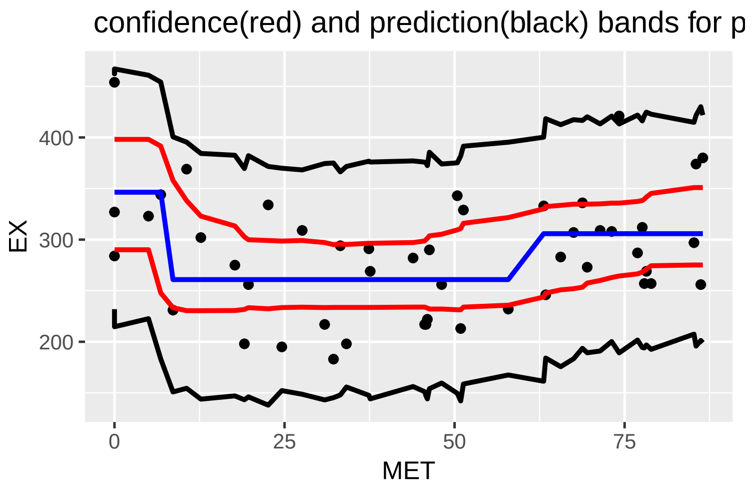




Analysis

In this task, we compute 95% confidence bands for the regression tree with target EX and MET as an independent variable with the same features as in task 3.2. We use non - parametric bootstrap and we perform 1000 iterations. From the first plot it is clear that the confidence band is bumpy. Although our target variable is continuous there are many equal values for different MET values. Therefore we have only 3 possible predictions(leaves). These values affect the confidence bands because many of the predictions have same EX -coordinate but different MET - coordinate as a result lines are created and the confidence band is not smooth. Moreover, in the second plot, it is obvious that our predictions are inside the confidence interval so we could say that the predictions in task 3.2 seem to be reliable.

3.4



Analysis

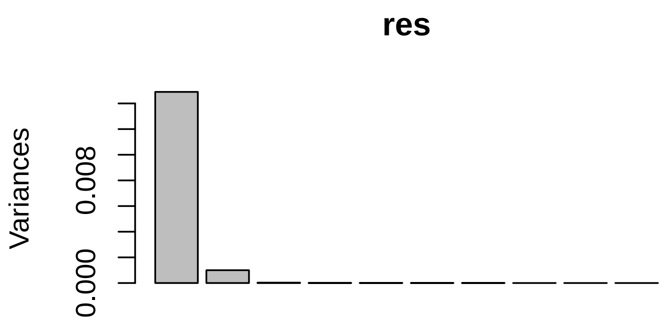
As in the task above, the results from the regression model seem to be reliable. It is also obvious that less than 5% of the data is outside the prediction bands. This is normal because the prediction interval is a random variable which calculates the expected value of the dependent variable EX without taking in consideration the variable MET. We have to mention also that in the plot we only have 48 observations and not all the predictions we made.

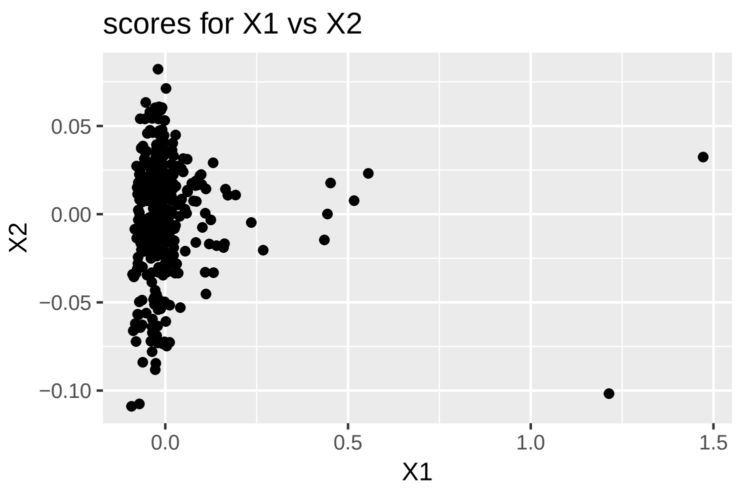
3.5

In task 3.2 we mention that the distribution of the residuals could be a Gamma distribution. In general, when we do not know the distribution of our data, it is better to use nonparametric bootstrap, while parametric bootstrap is better when we are sure about the distribution form. In every case non parametric is less precise when the parametric assumption is true but more accurate when it is false. The only reason why one could choose the parametric bootstrap is because of the number of observations because the nonparametric can not handle small samples(n < 40) while the parametric one can. In conclusion, despite the small sample i think that non parametric is better for this example because we are not sure neither for the data distribution nor for the residuals' one.

Task 4

4.1

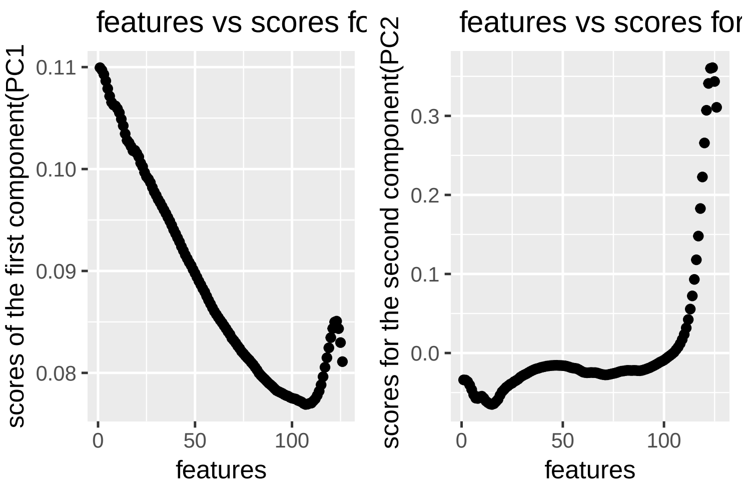




Analysis

In this task we perform a standard PCA trying to predict the viscosity levels for a collection of diesel fuels. From the first plot it is clear that the minimal number of components explaining the 99% of the total variance is 2. More specific one component consist the 93.332% of the variance and the second one 6.263%. Therefore if we sum them, they consist the 99.595% of the total variance. From the second plot 2 outliers can be easily detected on with X1 - coordinate close to 1.3 and one close to 1.5. We can also say that maybe the points close to X1 = 0.5 are outliers, creating a cluster.

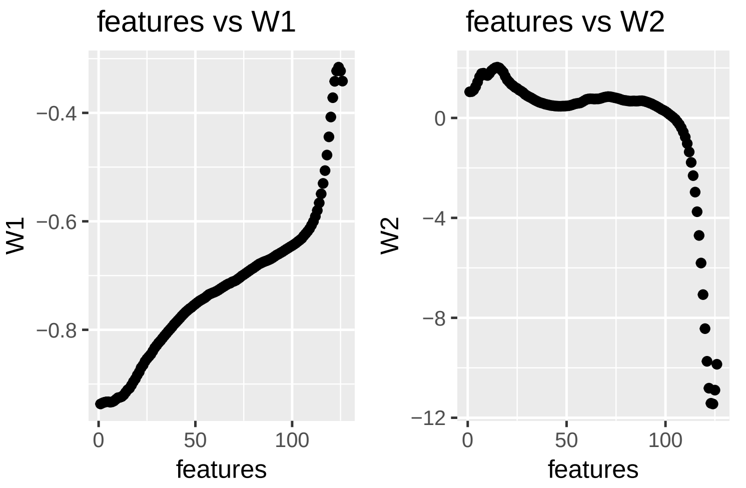
4.2

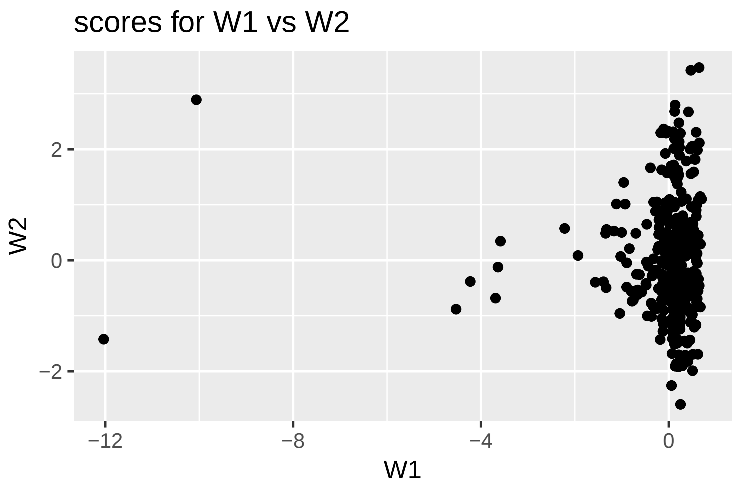


Analysis

In this task we make trace plots of the loadings of the two components selected in the task above (PC1,PC2). For the first component it is hard to say that it can be explained only by few origin features. Most of the features have high scores so we can not extract them. On the contrary, most of the features of PC2 are close to 0 regardless the number of features. More specific, only when the number of features is more than 100 we can find scores more than 0.05.

4.3





Analysis

First of all we have to mention that PCA does not find onlt one solution. On the other side, ICA choose the components(in this case 2) in order for them to be independent each other and also non - Gaussian. From the trace plots it is obvious that we have the same plots but they are reversed. This is happening because ICA searchs for the coorelation between the data and in this case the data coorelated linearly. And this is the reason why the trace plots are the same. Finaly for the last plot, it is also othe same but reverse for ICA and PCA with the same outliars but this time in the left side of the plot.

Appendix

```r
knitr::opts_chunk$set(echo = FALSE, fig.width = 4.5, fig.height = 3,
                      fig.align = "center",
                      warning = F, error = F, message = F)
library(readxl)
library(tree)
library(MASS)
library(e1071)
library(ggplot2)
library(gridExtra)
library(mboost)


#Spliting the data in Training, validation and test data.
data = read_excel("../dataset/creditscoring.xls")
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,]
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,]
#Create the decision tree
tree_deviance = tree(formula = good_bad~.,data = train,split = "deviance")
tree_gini = tree(formula = good_bad~., data = train,split = "gini")

# Predictions for deviance and gini for test data and test data
predict_deviance_test = predict(object = tree_deviance, newdata = test,type = "class")
predict_gini_test = predict(object = tree_gini, newdata = test, type = "class")
predict_deviance_train = predict(object = tree_deviance, newdata = train,type = "class")
predict_gini_train = predict(object = tree_gini, newdata = train, type = "class")

# Confusion matrices for training and test data
deviance_table_test = table(test$good_bad,predict_deviance_test)
gini_table_test = table(test$good_bad, predict_gini_test)
deviance_table_train = table(train$good_bad,predict_deviance_train)
gini_table_train = table(train$good_bad,predict_gini_train)

# misclassification rates for test data
mis_dev_test = (deviance_table_test[1,2] + deviance_table_test[2,1]) / sum(deviance_table_test)
mis_gini_test = (gini_table_test[1,2] + gini_table_test[2,1]) / sum(gini_table_test)

#misclassification rates for training data
mis_dev_train = (deviance_table_train[1,2] + deviance_table_train[2,1]) / sum(deviance_table_train)
mis_gini_train =  (gini_table_train[1,2] + gini_table_train[2,1]) / sum(gini_table_train)

# Data frame for all misclassification rates
mis_rate =  data.frame(c(mis_dev_train,mis_gini_train),c(mis_dev_test,mis_gini_test))
```

```r
colnames(mis_rate) = c("train", "test")
rownames(mis_rate) = c("Deviance", "Gini")

knitr::kable(x = mis_rate, caption = "misclassification rates")
trainScore = rep(0,15)
validScore = rep(0,15)

# A loop to calculate deviance for different trees
for (i in 2:15) {
  prunedTree = prune.tree(tree_deviance, best = i)
  pred = predict(prunedTree, newdata = valid, type = "tree")
  trainScore[i] = deviance(prunedTree)
  validScore[i] = deviance(pred)
}

#data frames for the plot
df1 = data.frame("leaves" = c(2:15), "Scores"= trainScore[2:15], "dataset" = "train" )
df2 = data.frame("leaves" = c(2:15), "Scores"  = validScore[2:15], "dataset" = "test" )
plot_df = rbind(df1,df2)

#plot for deviance vs number of leaves
ggplot(plot_df) + geom_point(aes_string(x = plot_df$leaves , y = plot_df$Scores, colour = "dataset")) +

# optimal tree with 4 nodes
optimal_tree = prune.tree(tree_deviance, best = 4)

#prediction for the test data
optimal_pred = predict(optimal_tree, newdata = test, type = "class")

# confusion matrix for test data
conf_test = table(optimal_pred, test$good_bad)

#misclassification rate for test data
mis_rate_test = (conf_test[1,2] + conf_test[2,1])/sum(conf_test)

print(list('Misclassification rate for optimal tree' = mis_rate_test))

#plot the optimal tree
plot(optimal_tree)
text(optimal_tree)

#structure of the optimal tree
print(list('structure of the optimal tree' = structure(optimal_tree)))

#print the confusion matrix
knitr::kable(x = conf_test, caption = "confusion matrix for test data")

#fit the naive bayes model
naive_model = naiveBayes(formula = good_bad ~., data = train)

#prediction with the training data
pred_train = predict(naive_model, newdata = train)
```

```r
#confusion matrix for the training data
conf_train_naive = table(pred_train,train$good_bad)

#misclassification rates for training data
mis_train_naive = (conf_train_naive[1,2] + conf_train_naive[2,1])/sum(conf_train_naive)

#prediction with the test data
pred_test = predict(naive_model, newdata = test)

#confusion matrix for the test data
conf_test_naive = table(pred_test,test$good_bad)

#misclassification rates for test data
mis_test_naive = (conf_test_naive[1,2] + conf_test_naive[2,1])/sum(conf_test_naive)

# print the confustions matrices and the misclassification rates both for training and test data

knitr::kable(x = conf_train_naive, caption = "confusion matrix for train data for Naive")

knitr::kable(x = conf_test_naive, caption = "confusion matrix for test data for Naive ")

mis_rate_naive = data.frame("misclassification rate(train)" = mis_train_naive,"misclassification rate(te

knitr::kable(x = t(mis_rate_naive), caption = "misclassification rates for Naive")

#function convert the probabilities to 0 and 1

binary = function(threshold,pred){
  pred = pred[,2]
  pred[which(pred > threshold)] = 1
  pred[which(pred <= threshold)] = 0
  return(pred)
}


#function computing the confusion matrix

conf_matrix = function(pred,real){

  #change the  good,bad to 1,0 and factorize
  real = as.factor(ifelse(real == "good", 1, 0))
  pred = factor(pred, levels = levels(real))

  #confusion matrix
  conf = table(real,pred)
  return(conf)
}

#function for tpr,fpr calculation
tpr_fpr = function(conf){

  #True positive rates
  TPR = conf[2,2] / rowSums(conf)[2]
  #False negative rates
  FPR = conf[1,2] / rowSums(conf)[1]

  return(list(tpr = TPR, fpr = FPR))

}
#function calculate TPR and FPR for each different pi

ROC_values = function(model){
  k = 1
  tpr = c()
  fpr = c()

  # loop for all different pi's
  for (i in seq(from = 0.05,to = 0.95, by = 0.05)) {

    # use the functions to calculate the conf matrix
    temp = binary(i,model)
    conf = conf_matrix(temp,test$good_bad)
    tpr[k] = tpr_fpr(conf)[[1]]
    fpr[k] = tpr_fpr(conf)[[2]]
    k = k + 1
  }
  df = data.frame(tpr,fpr)
  return(df)
}

# use the optimal tree and the Naive bayes

#predict for optimal tree
opt_pred = predict(object = optimal_tree, test, type = "vector")

#predict for Naive Bayes
naive_pred = predict(object = naive_model, test, type = "raw")

# calculate tpr and fpr for each model

tree_ROC = data.frame(ROC_values(opt_pred), "method" = rep( "tree"))
naive_ROC = data.frame(ROC_values(naive_pred),"method" = rep("naive"))

#ROC plot
df_roc = rbind(tree_ROC,naive_ROC)

ggplot() + geom_line(aes_string(x= df_roc$fpr, y = df_roc$tpr, colour = df_roc$method, linetype = df_roc
  geom_line(aes(x = c(0,1), y = c(0,1)),color = "black") +
  labs(title = "TPR vs FPR for optimal tree and Naive Bayes", x = "FPR", y = "TPR", linetype = "Model",

# function which use Loss function to classify

loss_class = function(pred){
  if(pred[2] > 10*pred[1]){
    a = "good"
  }else{
    a =  "bad"
    }
  return(a)
}
#function for calculate all the predicted values for a dataset

all_loss_pred = function(dataset){
  all_pred = c()
  for (i in 1:nrow(dataset)) {
    all_pred = c(all_pred,loss_class(dataset[i,]))
  }

  return(all_pred)
}
# naive model
naive_model = naiveBayes(formula = good_bad ~., data = train)

#predict for both test and training data
naive_pred_test = predict(object = naive_model, test, type = "raw")

naive_pred_train = predict(object = naive_model, train, type = "raw")

#predictions using the loss function

loss_test = all_loss_pred(naive_pred_test)
loss_train = all_loss_pred(naive_pred_train)

#confusion matrices for test and train data

conf_loss_train = table(train$good_bad,loss_train)
conf_loss_test = table( test$good_bad,loss_test)

#misclassification rates for test and training data
mis_test_loss = (conf_loss_test[1,2] + conf_loss_test[2,1])/sum(conf_loss_test)

mis_train_loss = (conf_loss_train[1,2] + conf_loss_train[2,1])/sum(conf_loss_train)

knitr::kable(x = conf_loss_train, caption = "confusion matrix for train data")

knitr::kable(x = conf_loss_test, caption = "confusion matrix for test data")

mis_rate_naive = data.frame("misclassification rate(train)" = mis_train_loss,"misclassification rate(te

knitr::kable(x = t(mis_rate_naive), caption = "misclassification rates")


#import the data
data <- read.csv2("../dataset/State.csv")

#reorder the data with respect to MET
data = data[order(data$MET),]
rownames(data) = c(1:nrow(data))
```

```
#plot EX vs MET
ggplot(data) + geom_point(mapping = aes(x = data$MET, y = data$EX)) + labs(title = "EX vs MET" , x = "E

# fit the tree with train data

tree1 = tree(formula = EX ~ MET, data = data,control = tree.control(nobs = nrow(data),minsize = 8))
#cross validation for the tree
cv_tree = cv.tree(object = tree1)

#choose the most efficinet number of nodes
nodes = cv_tree$size[which(cv_tree$dev==min(cv_tree$dev))]

#fit the  optimal model
optimal_tree = prune.tree(tree1, best = nodes)

# prediction for the optimal tree
pred = predict(optimal_tree,newdata = data)


#plot the optimal tree
summary(optimal_tree)
plot(optimal_tree)
text(optimal_tree)

# calculate the residuals
res = data$EX - pred

# shift the res in order to plot the distribution
res = res + abs(min(res))

# fit a gamma distribution to compare
fit_gamma <- fitdistrplus::fitdist(res, distr = "gamma", "mme")

#fit_gamma$estimate

#Histogramm  plot for the residuals

ggplot(as.data.frame(res),aes(x = res)) +
geom_histogram(aes(x= res, y=..density..), bins = 25, col = "black" , fill = "yellow" )+   geom_density
geom_line(aes(y = dgamma(res, fit_gamma$estimate[1],fit_gamma$estimate[2])), col = "black", size = 1) +

# Plot for the original and fitted data
df = data.frame(data$MET , pred, data$EX)
ggplot(df) +
  geom_point(mapping = aes(x = df[,1],y = df[,2]), color = "red")+
  geom_point(mapping = aes(x = df[,1], y = data$EX),color = "black") +
  labs(title = "plot for fitted and real values", x = "MET", y = "EX")

library(boot)

#optimal_tree = prune.tree(tree1, best = 3)
# prediction for the optimal tree
#pred = predict(optimal_tree,newdata = data)
```

```r
# computing bootstrap samples
bootstrap = function(data,ind){
  data1 = data[ind,] # extract bootstrap sample

  # fit the tree
  tree1 = tree(formula = EX ~ MET, data = data1,control = tree.control(nobs = nrow(data),minsize = 8))

  # take the optimal one
  optimal_tree = prune.tree(tree1, best = 3)  # fit regression tree model

  #predict values for all Area values from the original data

  my_pred = predict(optimal_tree , newdata = data)
  return(my_pred)
}


res = boot(data, bootstrap, R=1000) #make bootstrap

#compute the confidence bands

conf_int = envelope(res)

# plot the confidence interval
alldata = data.frame( x = data$MET,data$EX, conf_int$point[1,], conf_int$point[2,])

ggplot(alldata) + geom_point(mapping = aes(x = x ,y = data$EX)) +
  geom_line(mapping = aes(x= x, y = conf_int$point[1,],col = "red"),size = 1)  +
  geom_line(mapping = aes(x= x, y = conf_int$point[2,],col = "red"),size = 1) +
  geom_line(mapping = aes(x, y = res$t0),size = 1) +
  labs(title = " confidence bands ", x = "MET", y = "EX", col = "confidence bands")

#plot for the confidence band
ggplot(alldata) +
  geom_line(mapping = aes(x= x, y = conf_int$point[1,],col = "red"),size = 1)  +
  geom_line(mapping = aes(x= x, y = conf_int$point[2,],col = "red"),size = 1) +
  geom_point(mapping = aes(x = x , y = pred), color = "black",size = 1) +
  labs(title = " Confidence bands for predictions ", x = "MET", y = "EX")


tree1 = tree(formula = EX ~ MET, data = data,control = tree.control(nobs = nrow(data),minsize = 8))
mle = prune.tree(tree1, best = 3)
my_pred = predict(optimal_tree , newdata = data)

rng = function(data, mle) {
  data1 = data.frame(EX = data$EX, MET = data$MET)
  n=length(data$EX)
  predicts = predict(mle,newdata = data1)
  r = data$EX - predicts

  #generate new Price
  data1$EX = rnorm(n, predicts, sd = sd(r))
  return(data1)
```

```
}

#Function for the confidence interval
confidence = function(data1){
  tree1 = tree(formula = EX ~ MET, data = data1,control = tree.control(nobs = nrow(data1),minsize = 8))
  mle = prune.tree(tree1, best = 3)

  #predict values for all Area values from the original data
  pred = predict(mle, newdata = data)
  return(pred)
}

prediction = function(data1){

  tree1 = tree(formula = EX ~ MET, data = data1,control = tree.control(nobs = nrow(data1),minsize = 8))
  mle = prune.tree(tree1, best = 3)
  pred = predict(mle, newdata = data)

  #predict values for all Area values from the original data
  y = data$EX
  n = nrow(data)
  residual = y - pred

  #generate new Price
  data1$EX = rnorm(n, pred, sd = sd(residual))
  return(data1$EX)
}

res_pred = boot(data, statistic = prediction, R=1000, mle=mle,ran.gen=rng, sim="parametric")

res_conf = boot(data, statistic= confidence, R=1000, mle=mle,ran.gen=rng, sim="parametric")

#confidence and prediction interval for parametric bootstrap
pred_int_par = envelope(res_pred)
conf_int_par = envelope(res_conf)

# plot the confidence interval
data_parametric = data.frame( x = data$MET, data$EX, conf_int_par$point[1,], conf_int_par$point[2,], pr

ggplot(data_parametric) + geom_point(mapping = aes(x = x , y = data$EX), col = "black") +
  geom_line(mapping = aes(x= x, y = conf_int_par$point[1,]),color = "red", size = 1)  +
  geom_line(mapping = aes(x= x, y = conf_int_par$point[2,]),color = "red",size = 1) +
  geom_line(mapping = aes(x, y = my_pred),color = "blue",size = 1) +
  geom_line(mapping = aes(x= x, y = pred_int_par$point[1,]),size = 1) +
  geom_line(mapping = aes(x= x, y = pred_int_par$point[2,]),size = 1) +
  labs(title = " confidence(red) and prediction(black) bands for parametric bootstrap", x = "MET", y =


fuels <- read.csv2("../dataset/NIRSpectra.csv")
my_fuels = fuels
my_fuels$Viscosity = c()
res = prcomp(my_fuels)
lambda=res$sdev^2
```

```r
#proportion of variation
a = sprintf("%2.3f",lambda/sum(lambda)*100)
screeplot(res)

# plot for scores in PC1 PC2
ggplot() +
  geom_point(mapping=aes(res$x[,1],res$x[,2])) +
  labs(title = "scores for X1 vs X2", x = "X1" , y = "X2")


# plot the scores in PC1,PC2

a = ggplot() +
  geom_point(mapping = aes( x = c(1:126), y = res$rotation[,"PC1"])) +
  labs(title = " features vs scores for the first component", x = "features", y = "scores of the first 

b = ggplot() +
  geom_point(mapping = aes( x = c(1:126), y = res$rotation[,"PC2"])) +
  labs(title = " features vs scores for the second component ", x = "features", y = " scores for the se

grid.arrange(grobs = list(a,b), ncol = 2)

library("fastICA")

my_fuels = fuels
my_fuels$Viscosity = c()
set.seed(12345)

# Perform ICA
ICA = fastICA(my_fuels, 2, alg.typ = "parallel", fun = "logcosh", alpha = 1, method = "R", row.norm = F

# Compute posterior W

W_post = (ICA$K) %*% (ICA$W)

# traceplot of W posterior

w1plot = ggplot() +
  geom_point(mapping = aes( x = c(1:126), y = W_post[,1])) +
  labs(title = " features vs W1", x = "features", y = "W1")


w2plot = ggplot() +
  geom_point(mapping = aes( x = c(1:126), y = W_post[,2])) +
  labs(title = " features vs W2", x = "features", y = "W2")

grid.arrange(grobs =list(w1plot , w2plot),ncol = 2)

# plot of the scores for the two first latent features

ggplot() +
  geom_point(mapping = aes(ICA$S[,1],ICA$S[,2])) +
  labs(title = "scores for W1 vs W2", x = "W1" , y = "W2")
```