Lab 2 - Group A14

Stefano Toffol (steto820), Andreas Stasinakis (andst745), Mim Kemal Tekin (mimte666)

December 9, 2018

Contents

Assingment 2

Task 2.1

Spliting the data

```
#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,]
```

TASK 2.2

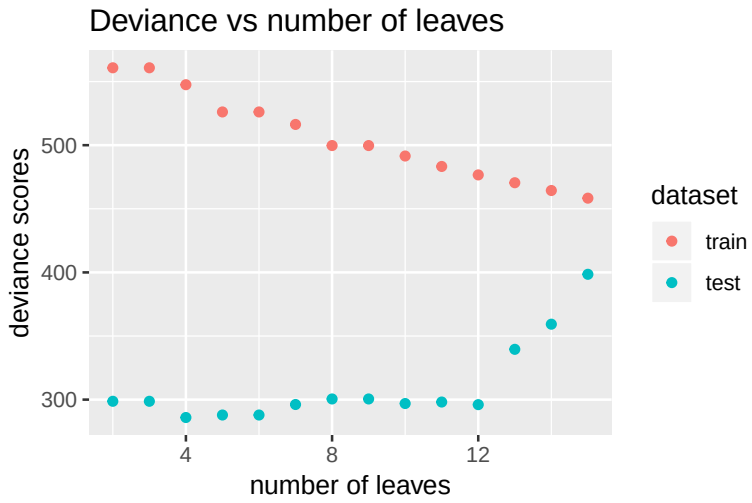
Table 1: misclassification rates

	train	test
Deviance	0.212	0.268
Gini	0.240	0.368

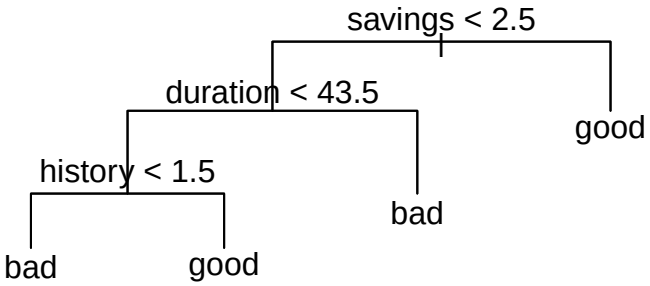
Analysis

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.

Task 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

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.

Task 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

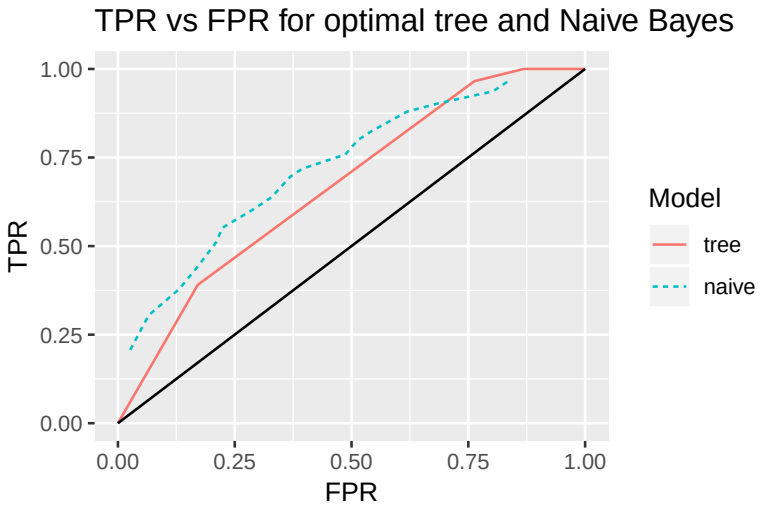
Table 5: misclassification rates for Naive

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

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.

Task 2.5



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.

Task 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

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

$$L = \text{Observed} \begin{matrix} \\ good \\ bad \end{matrix} \begin{matrix} \overset{Predicted}{} \\ \begin{pmatrix} 0 & 1 \\ 10 & 0 \end{pmatrix} \end{matrix}$$

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.

Assignment 3

Question 1

In the plot in Figure 1 is displayed the relationship of the variables *EX* (per capita state and local public expenditures in dollars) versus *MET* (the percentage of population living in standard metropolitan areas) from the dataset `state.csv`, a collection of demographic and economical informations regarding the states of the US in 1960.
The variance observed is consistent since the values od the Y variable fluctuate in a range of approximately 150 when conditioned to the X. The relationship between the two variables however is surely not linear and the smoothed curve highlight a possible quadratic dependency.

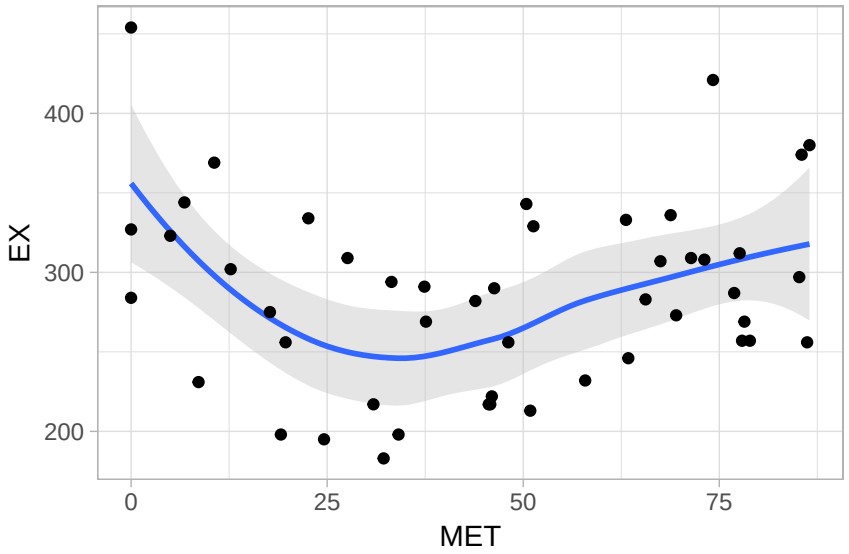


Figure 1: Scatterplot of the dependency of EX to MET.

Question 2

Through *cross-validation* and setting the minimum number of observations on the leafes to be equal to eight, a tree with just two splits is selected (Figure 2). The first, main split happens for values of *MET* of 7.7: the model is trying to capture the initial, steep decreasing trend observed in the scatterplot. The remaining observations are then modelled with a cut corresponding to values of MET of 60.5.

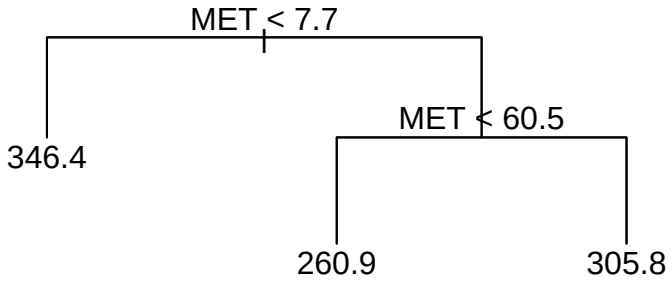


Figure 2: Selected tree using cross-validation.

As one may expect, modelling such a variable data, mainly described by a quadratic trend, leads to poor results (Figure 3). The fitted values, equal to the mean of the observed ones for each of the three splits (red cross and text), leave out a great variability of the data, not catched by the regression tree. In particular the first split, that gives to the observations with values of the X smaller than 7.7 a value of the response equal to 346, despite having just 5 observations is not capable of predicting the wide jumps of the data.

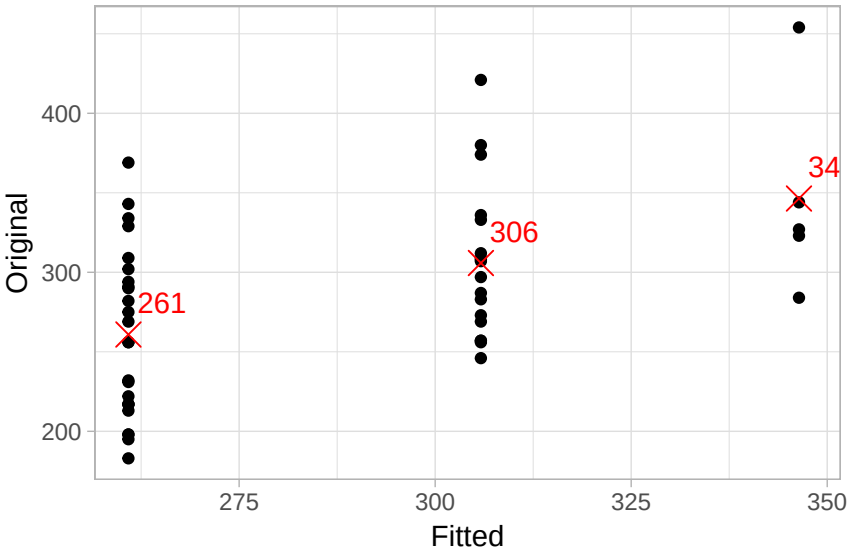


Figure 3: Original vs fitte.

If we then look to the residuals (Figure 4) it is evident how poorly the model catches the variance of the data. Moreover the histogram shows a distribution far away from the desired normality: the distribution is positively skewed and presents a heavy right tail. The shape is really similar to the one of a Gamma distribution. Shifting the residuals to non-negative values (where the gamma function is defined) and thanks to the function `fitdist` from the `fitdistrplus` package, a red line following a gamma distribution has been drawn on the plot. It closely resambles the empirical density of our residuals (brown line). The estimated parameters of the gamma are: $\hat{\alpha} = 2.3983$; $\hat{\beta} = 32.4737$.

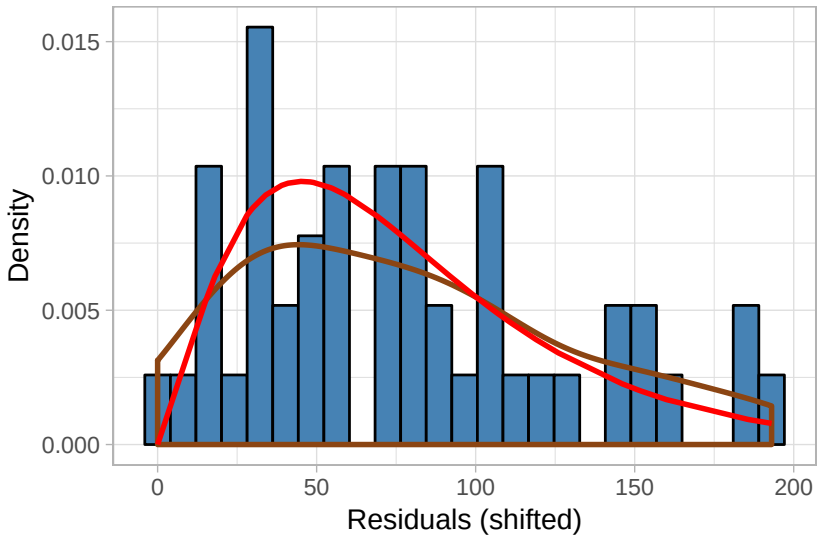


Figure 4: Selected tree using cross-validation.

Question 3

The CI computed using a non-parametric bootstrap is reported in Figure 5. The band (in blue) is quite bumpy and pretty large if compared to the distributions of the data points around the regression line of the tree (in red). More than half of observations are inside it and even the points that lay outside of it are mostly on the border. As previously commented, the regression tree does not seem appropriate for modelling this type of relationship. The tree has expecially poor performances close to the limit of the the observed domain of *MET*.

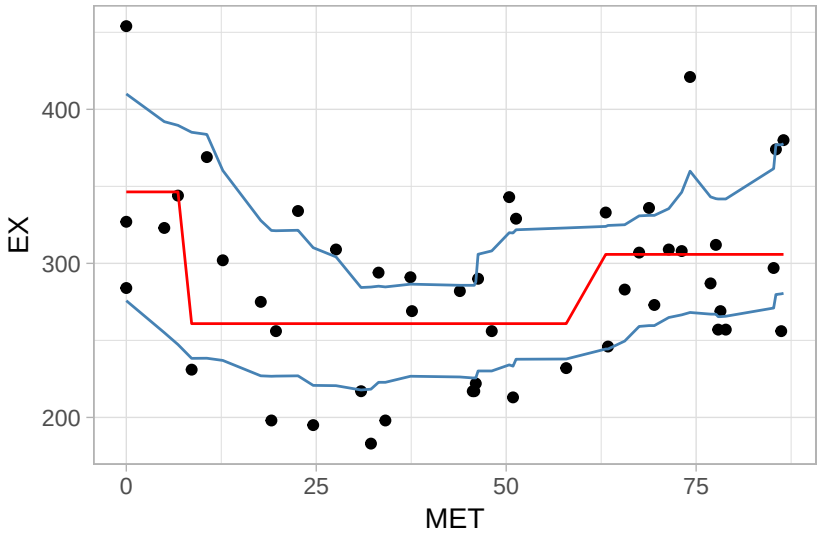


Figure 5: Non-parametric bootstrap confidence interval of the regression tree (blue line).

Question 4

As expected, not even a parametric, gaussian bootstrap gives a reliable CI (Figure 6, blue line). Compared to the non-parametric one, the only noticeable difference is that the line that defines the border of the band is now almost smooth. However the region is still pretty large, since its width ranges from 50 to 100 approximately. Moreover it seems almost unchanged from the previous estimates.
Regarding the prediction interval, it appears to be much more bumpy and, naturally, wider than the confidence one. Obviously not 5% of the data is outside this interval: the region is constructed so that when calculated for a number of successive samples from the same population, a prediction interval will contain a future observation a specified percentage (in this case 95%) of the time. The single dataset that we are observing is just a realization of the random variable, hence it's not necessary that exactly 5% of them lay outside of the interval.

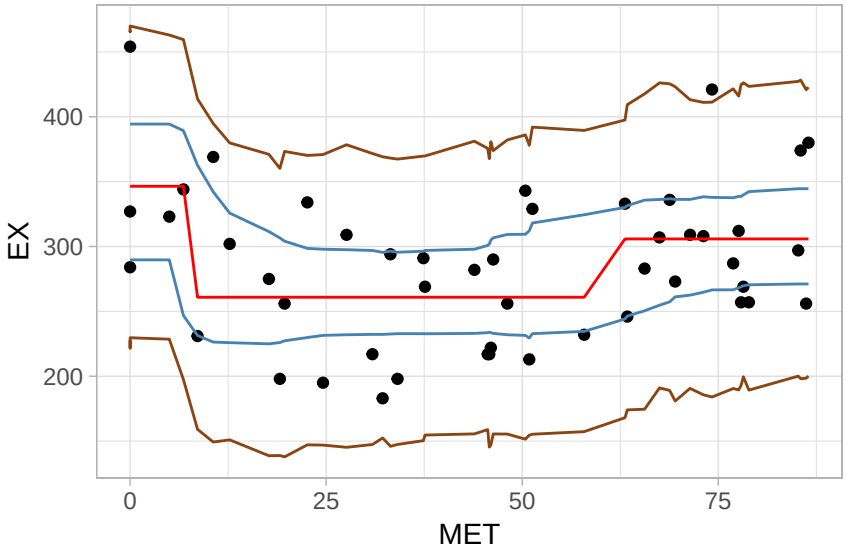


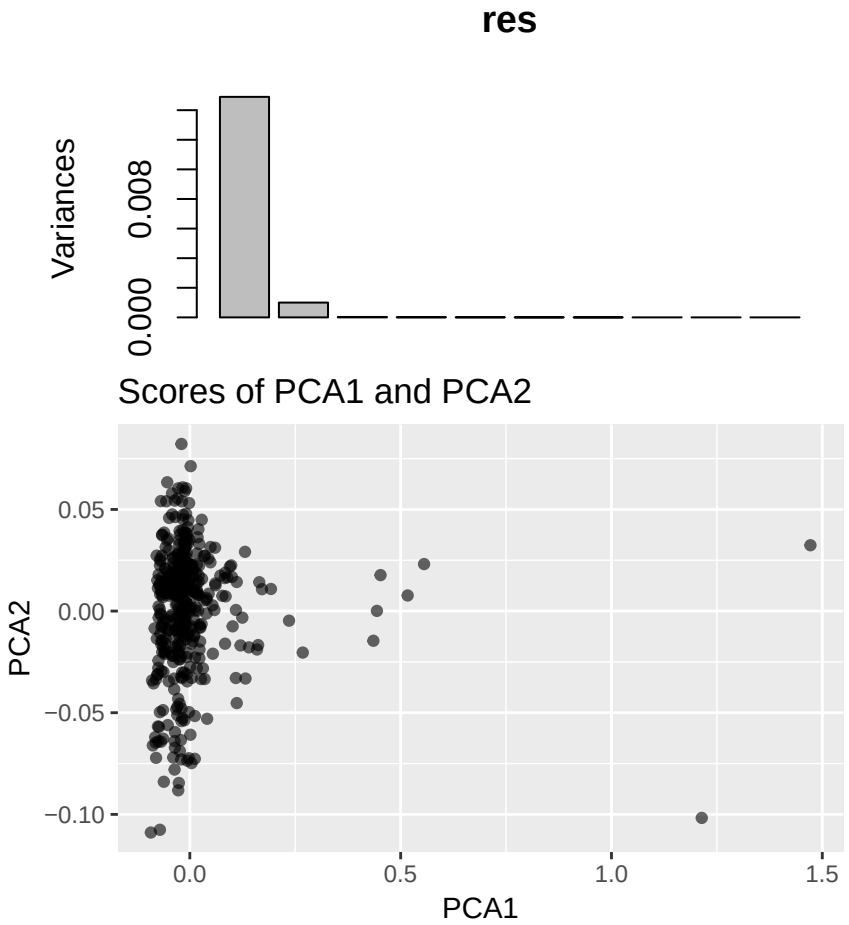
Figure 6: Parametric gaussian bootstrap confidence interval (blue line) and prediction (in brown) of the regression tree.

Question 5

Since the residuals where distributed as a Gamma variable, this type of distribution may provide better estimates of the interval, expecially on the extremes of the domain.

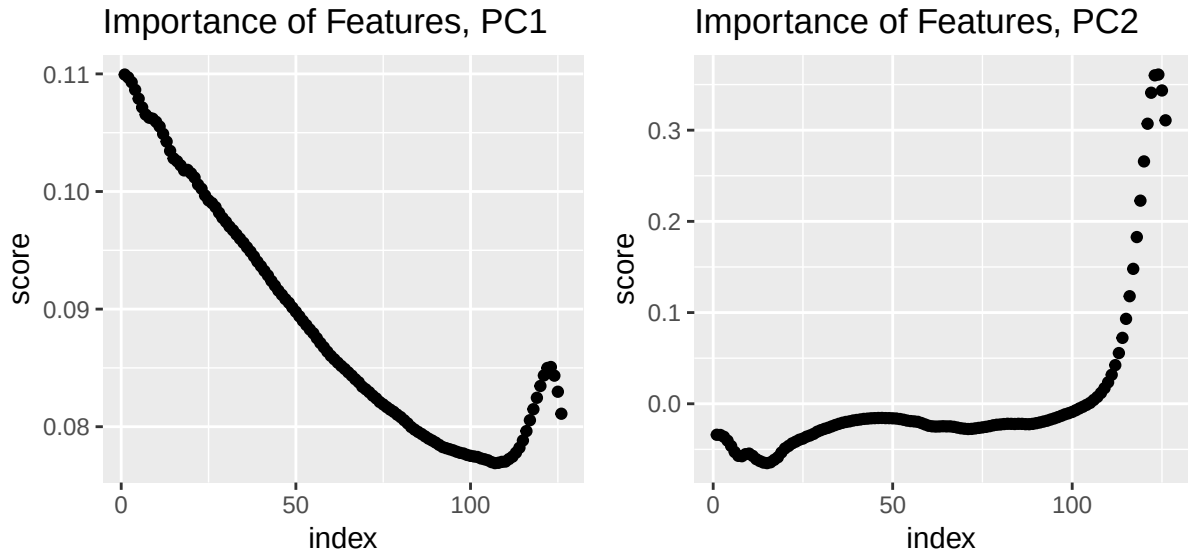
Assignment 4. Principal Components

Task 4.1



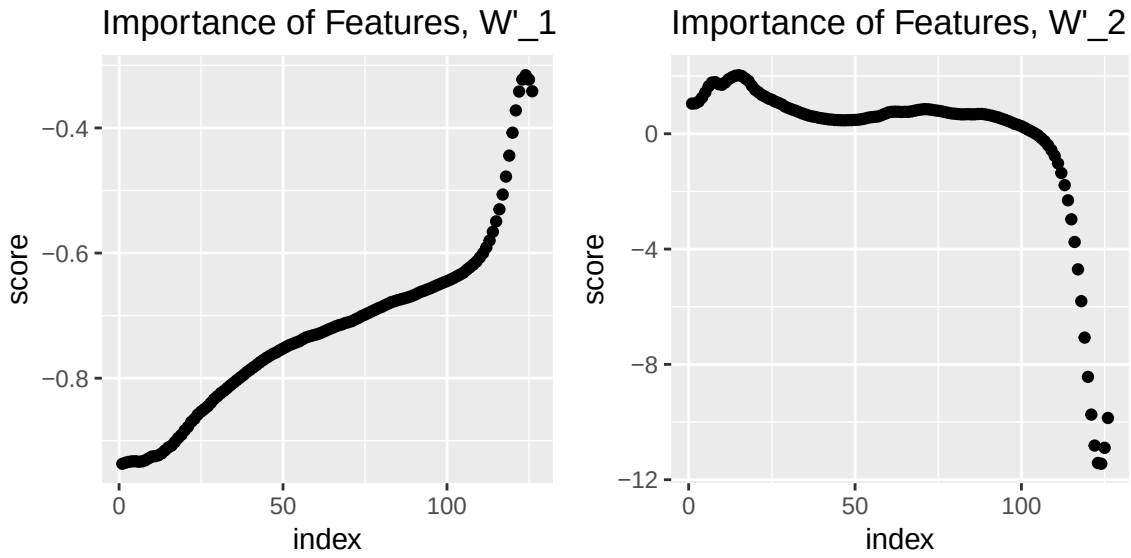
In the variation plot we can see that the first 2 feature looks important. These 2 PC has 99.5957% of the total variance and we should extract them.
If we examine the scatter plot of the scores of (PC1, PC2), we can see most of points are placed around 0 at PCA1 axis excapt the outlier cluster around 0.5 and 2 outlier greater than 1 at PCA1 axis. Those outliesr can be called unusual diesel fuels.

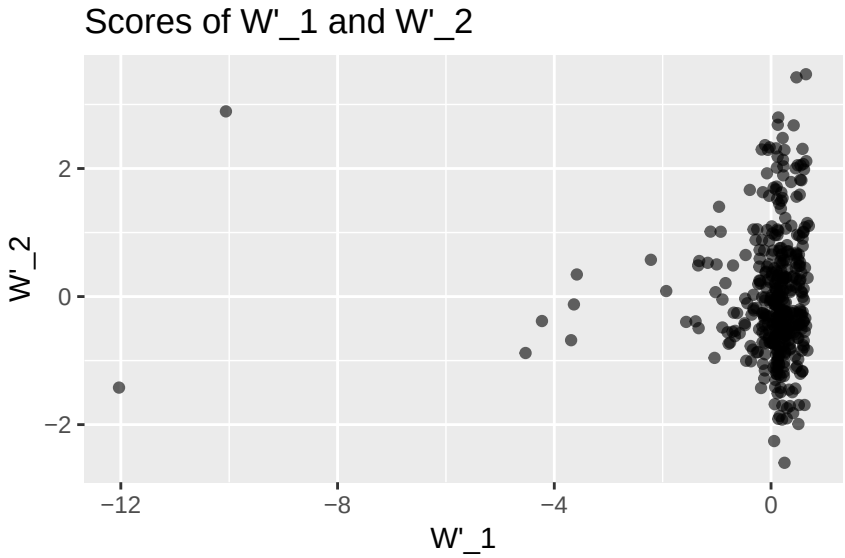
Task 4.2



In the first plot, feature value in principle component decreases until around 110 and after that it increases again. But even the lowest one is close to 0.08.
In the second plot nearly all aff them close to zero, so it can be explained by a few original features.

Task 4.3





We can see ICA and PCA results are in quite same shape, the only difference is that they look like inverted. PCA tries to find the projection which has the lowest variance between data while ICA tries to find best seperation with maximizing independency between lantent components. So we can see from the same kind of results, the features of original data are already independent. Furthermore, the projection which has lowest variation has also independent components.

Appendix

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

#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
```

```r
mis_rate =  data.frame(c(mis_dev_train,mis_gini_train),c(mis_dev_test,mis_gini_test))
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(t

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_ro
  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")


# ------------------------------------------------------------------------------
# A3 - Q1
# ------------------------------------------------------------------------------

data = read.csv2("../dataset/State.csv")
data = data[order(data$MET),]
```

```r
ggplot(aes(x = MET, y = EX), data = data) +
  geom_smooth(fill = "grey75") +
  geom_point() +
  theme_light()


# ------------------------------------------------------------------------------
# A3 - Q2
# ------------------------------------------------------------------------------

fit_tree <- tree(EX ~ MET, data = data)
cv_tree <- cv.tree(fit_tree, tree.control(nobs = nrow(data), minsize = 8))
best <- cv_tree$size[which.min(cv_tree$dev)]
# Best value is 3
fit_tree3 <- prune.tree(fit_tree, best = 3)
predicted_tree3 <- predict(fit_tree3, newdata = data)
residuals_tree3 <- data$EX - predicted_tree3
residuals_tree3_scaled <- residuals_tree3 + abs(min(residuals_tree3))
fit_gamma <- fitdistrplus::fitdist(residuals_tree3_scaled, distr = "gamma", "mme")


plot(fit_tree3)
text(fit_tree3)


df_scatter <- data.frame(Fitted = predicted_tree3, Original = data$EX)
predictions <- unique(df_scatter$Fitted)
means <- c(mean(df_scatter$Original[df_scatter$Fitted==predictions[1]]),
           mean(df_scatter$Original[df_scatter$Fitted==predictions[2]]),
           mean(df_scatter$Original[df_scatter$Fitted==predictions[3]]))
df_means <- data.frame(predictions, means)

ggplot() +
  geom_point(aes(x = Fitted, y = Original), data = df_scatter) +
  geom_point(aes(x = predictions, y = means), data = df_means, col = "red",
             shape = 4, size = 4) +
  geom_text(aes(x = predictions, y = means, label = round(means)), data = df_means,
            col = "red", hjust = -0.025, vjust = -1, nudge_x = 1) +
  theme_light()


residuals_tree3_scaled <- as.data.frame(residuals_tree3_scaled)
ggplot(aes(x = residuals_tree3_scaled), data = residuals_tree3_scaled) +
  geom_histogram(aes(y = ..density..), bins = 25, col = "black", fill = "steelblue") +
  geom_density(col = "chocolate4", size = 1) +
  geom_line(aes(y = dgamma(residuals_tree3_scaled, fit_gamma$estimate[1],
                           fit_gamma$estimate[2])), col = "red", size = 1) +
  labs(x = "Residuals (shifted)", y = "Density") +
  theme_light()


# ------------------------------------------------------------------------------
# A3 - Q3
```

```r
# ------------------------------------------------------------------------------

B <- 1000

# Function for tree fitting:
boot_nonpar <- function(data, ind, best = 3) {

  d_boot <- data[ind,]
  boot_tree <- tree(EX ~ MET, data = d_boot,
                    control = tree.control(nobs = nrow(d_boot), minsize=8))
  boot_tree <- prune.tree(boot_tree, best = best, newdata = d_boot)
  boot_pred <- predict(boot_tree, newdata = data)
  return(boot_pred)

}

boot_nonpar_tree <- boot(data, boot_nonpar, R = B)
CI_boot_nonpar <- envelope(boot_nonpar_tree)$point


df_nonpar_plot <- data.frame(upper = CI_boot_nonpar[1,], lower = CI_boot_nonpar[2,],
                             estimate = boot_nonpar_tree$t0,
                             EX = data$EX, MET = data$MET)

ggplot(aes(y = EX, x = MET), data = df_nonpar_plot) +
  geom_point() +
  geom_line(aes(y = estimate), col = "red") +
  geom_line(aes(y = upper), col = "steelblue") +
  geom_line(aes(y = lower), col = "steelblue") +
  theme_light()


# ------------------------------------------------------------------------------
# A3 - Q4
# ------------------------------------------------------------------------------

rng <- function(data, mle) {

  d_boot <- data.frame(MET = data$MET, EX = data$EX)
  n <- nrow(data)
  mu <- predict(mle, newdata = d_boot)
  sigma <- sd(d_boot$EX - mu)
  d_boot$EX = rnorm(n, mu, sigma)
  return(d_boot)

}

boot_par_CI <- function(d_boot){

  boot_tree <- tree(EX ~ MET, data = d_boot,
                    control = tree.control(nobs = nrow(d_boot), minsize=8))
  boot_tree <- prune.tree(boot_tree, best = 3)
  boot_pred <- predict(boot_tree, newdata = data)
```

```
  return(boot_pred)

}

boot_par_PI <- function(d_boot){

  boot_tree <- tree(EX ~ MET, data = d_boot,
                    control = tree.control(nobs = nrow(d_boot), minsize=8))
  boot_tree <- prune.tree(boot_tree, best = 3)
  temp_pred <- predict(boot_tree, newdata = data)
  sigma <- sd(data$EX - temp_pred)
  n <- length(data$EX)
  boot_pred <- rnorm(n, temp_pred, sigma)
  return(boot_pred)

}

boot_par_tree_conf <- boot(data, boot_par_CI, R = B, mle = fit_tree3, ran.gen = rng,
                           sim = "parametric")
boot_par_tree_pred <- boot(data, boot_par_PI, R = B, mle = fit_tree3, ran.gen = rng,
                           sim = "parametric")
CI_boot_par <- envelope(boot_par_tree_conf)$point
PI_boot_par <- envelope(boot_par_tree_pred)$point


df_par_plot <- data.frame(upper_CI = CI_boot_par[1,], lower_CI = CI_boot_par[2,],
                          estimate = boot_par_tree_conf$t0, EX = data$EX, MET = data$MET,
                          upper_PI = PI_boot_par[1,], lower_PI = PI_boot_par[2,])

ggplot(aes(y = EX, x = MET), data = df_par_plot) +
  geom_point() +
  geom_line(aes(y = estimate), col = "red") +
  geom_line(aes(y = upper_CI), col = "steelblue") +
  geom_line(aes(y = lower_CI), col = "steelblue") +
  geom_line(aes(y = upper_PI), col = "chocolate4") +
  geom_line(aes(y = lower_PI), col = "chocolate4") +
  theme_light()


################################### TASK 4.1 ###################################

df_spectra = read.csv2("../dataset/NIRSpectra.csv")
data1 = df_spectra
data1$Viscosity = c()
res = prcomp(data1)
# res2 = prcomp(df_spectra[, -dim(df_spectra)[2]])
lambda = res$sdev^2
#eigenvalues
# lambda
#proportion of variation
# sprintf("%2.3f",lambda/sum(lambda)*100)

plot_coor_pca = ggplot() +
```

```
  geom_point(aes(x=res$x[,1], y=res$x[, 2]), alpha = 0.6) +
  labs(title = "Scores of PCA1 and PCA2", x = "PCA1", y = "PCA2")
screeplot(res)
plot_coor_pca

################################### TASK 4.2 ###################################


plot_scores_pc1 = ggplot() +
  geom_point(aes(x=1:length(res$rotation[,"PC1"]), y=res$rotation[, "PC1"])) +
  labs(title = "Importance of Features, PC1 ", x = "index", y = "score")

plot_scores_pc2 = ggplot() +
  geom_point(aes(x=1:length(res$rotation[,"PC2"]), y=res$rotation[, "PC2"])) +
  labs(title = "Importance of Features, PC2", x = "index", y = "score")

grid.arrange(grobs=list(plot_scores_pc1,plot_scores_pc2), ncol=2)

################################### TASK 4.3 ###################################

set.seed(12345)

a = fastICA(data1, 2, alg.typ = "parallel", fun = "logcosh", alpha = 1,
method = "R", row.norm = FALSE, maxit = 200, tol = 0.0001) #ICA

W_prime = a$K %*% a$W


################################### TASK 4.3.1 ###################################
plot_scores_ica1 = ggplot() +
  geom_point(aes(x=1:dim(W_prime)[1], y=W_prime[,1])) +
  labs(title = "Importance of Features, W'_1", x = "index", y = "score")

plot_scores_ica2 = ggplot() +
  geom_point(aes(x=1:dim(W_prime)[1], y=W_prime[,2])) +
  labs(title = "Importance of Features, W'_2", x = "index", y = "score")


################################### TASK 4.3.2 ###################################

plot_coor_ica = ggplot() +
  geom_point(aes(x=a$S[,1], y=a$S[,2]), alpha=0.6) +
  labs(title = "Scores of W'_1 and W'_2", x = "W'_1", y = "W'_2")



grid.arrange(grobs = list(plot_scores_ica1, plot_scores_ica2), ncol=2)
plot_coor_ica
```