Block 1 - Lab 2

mimte666 - Mim Kemal Tekin

12/3/2018

Contents

Assignment 2: Analysis of Credit Scoring

The data file creditscoring.xls contains data retrieved from a database in a private enterprise. Each row contains information about one customer. The variable good/bad indicates how the customers have managed their loans. The other features are potential predictors. Your task is to derive a prediction model that can be used to predict whether or not a new customer is likely to pay back the loan.

Task 2.1

Import the data to R and divide into training/validation/test as 50/25/25: use data partitioning code specified in Lecture 1e.

```
################################### TASK 2.1 ###################################

# read data
df_credit = read_xls("../dataset/creditscoring.xls")
df_credit$good_bad = as.factor(df_credit$good_bad)
# split data
n = dim(df_credit)[1]
set.seed(12345)
id = sample(1:n, floor(n*0.5))
train = df_credit[id,]
id1 = setdiff(1:n, id)
set.seed(12345)
id2 = sample(id1, floor(n*0.25))
valid = df_credit[id2,]
id3 = setdiff(id1,id2)
test = df_credit[id3,]
```

Task 2.2

Fit a decision tree to the training data by using the following measures of impurity
a. Deviance
b. Gini index

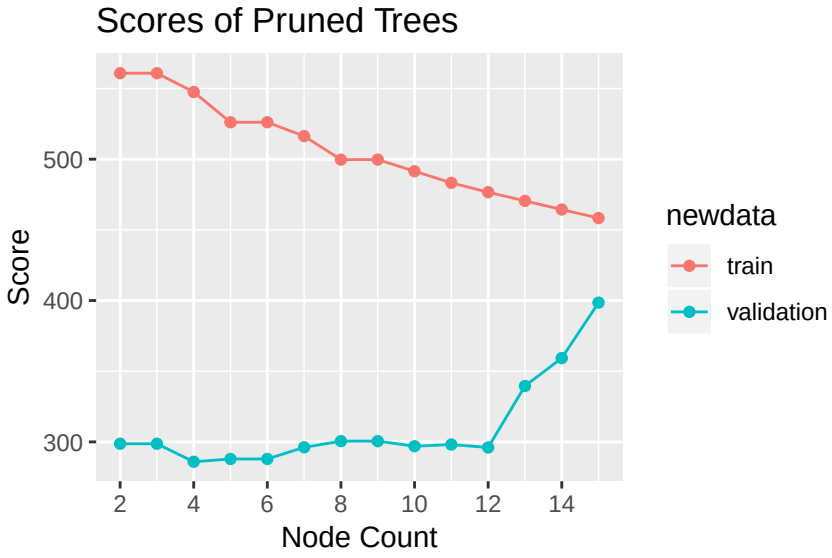
Table 1: Missclassification Rates

	test	train
Deviance	0.268	0.212
Gini Index	0.372	0.238

In this task, Deviance and Gini Index are tried as measure of impurity for the decision tree, and misclassification rates on test and train data are calculated. As we can see Deviance measurement has better rate, so we will continue to other tasks with Deviance.

Task 2.3

Use training and validation sets to choose the optimal tree depth. Present the graphs of the dependence of deviances for the training and the validation data on the number of leaves. Report the optimal tree, report it's depth and the variables used by the tree. Interpret the information provided by the tree structure. Estimate the misclassification rate for the test data.



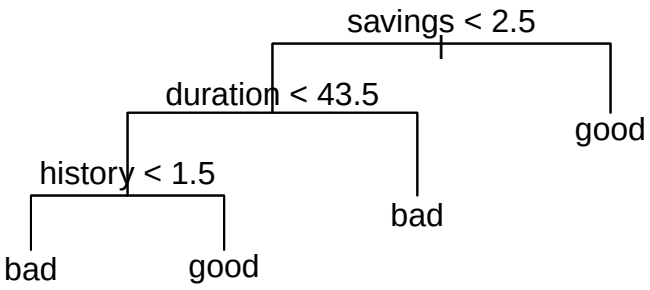


Table 2: Missclassification Rates

	test
Misclassification	0.256

Table 3: Confusion Matrix of Optimal Tree

	bad	good	Frequencies
bad	18	58	76
good	6	168	174
Frequencies	24	226	250

If we look score plot we can see the lowest test score is with 4 node in the leaves. We can choose 4 node in the leaves for an optimal tree. Above, we can see the plot of optimal tree with 4 node. Optimal tree has level=4, depth=3, and "saving", "duration", "history" variables are used in the tree. And as we can see misclassification rate is better than the tree in task 2.2.

Task 2.4

Use training data to perform classification using Naive Bayes and report the confusion matrices and misclassification rates for the training and for the test data. Compare the results with those from step 3.

Table 4: Confusion Matrix of Naive Bates / Train Data

	bad	good	Frequencies
bad	95	52	147
good	98	255	353
Frequencies	193	307	500

Table 5: Confusion Matrix of Naive Bates / Test Data

	bad	good	Frequencies
bad	46	30	76
good	49	125	174
Frequencies	95	155	250

Table 6: Missclassification Rates

	train	test
Misclassification	0.3	0.316

If we compare results of this task and results of task 2.3, misclassification rates are worse than task 2.3. But in this case if we investigate deeper with confusion matrixes, despite of higher misclassification rate Naive Bayes classified less good people who are bad in reality. In the same time, it classified more bad people who are good in reality. This classifier is not accurate as tree classifier, but it does not offer loan to people who cannot afford to pay back.

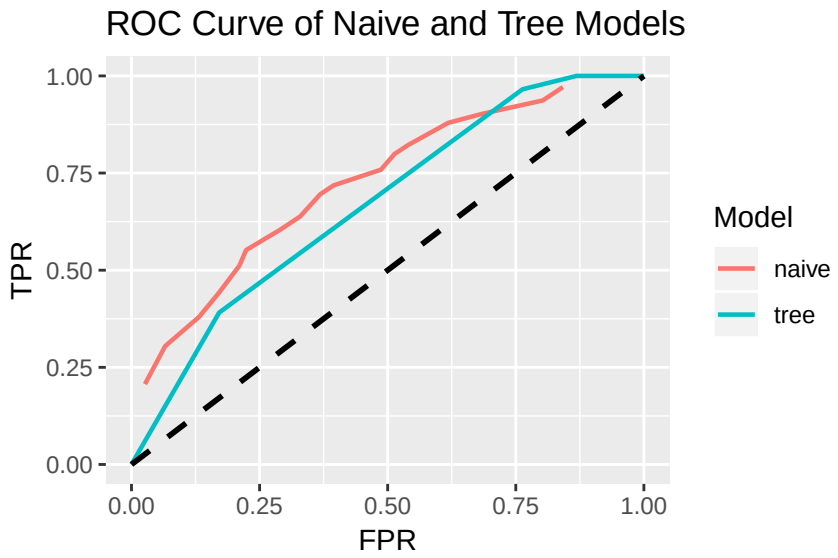
Task 2.5

Use the optimal tree and the Naive Bayes model to classify the test data by using the following principle:

$$\hat{Y} = \begin{cases} 1, & \text{if } p(Y = \text{'good'} \mid X) > \pi \\ 0, & \text{otherwise} \end{cases}$$

$$\text{where} \quad \pi = 0.05,\ 0.1,\ 0.15,\ ...,\ 0.9,\ 0.95$$

Compute the TPR and FPR values for the two models and plot the corresponding ROC curves. Conclusion?



In this task we create a ROC Curve. We examine how good naive and tree models predict positive (good) values by calculating FPR (False Positive Rate) and TPR (True Positive Rate). If the model has the greatest area under the curve, it means that model is best classifier. In our case, Naive Bayes model has more area, so we can say Naive Bayes model is better classifier than Tree according to AUC (Area Under Curve).

Task 2.6

Repeat Naive Bayes classification as it was in step 4 but use the following loss matrix:

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

and report the confusion matrix for the training and test data. Compare the results with the results from step 4 and discuss how the rates has changed and why.

Table 7: Confusion Matrix for Naive Bayes / Train Data

	bad	good	Frequencies
bad	137	10	147
good	263	90	353
Frequencies	400	100	500

Table 8: Confusion Matrix for Naive Bayes / Test Data

	bad	good	Frequencies
bad	71	5	76
good	122	52	174
Frequencies	193	57	250

Table 9: Missclassification Rates

	test	train
Misclassification	0.508	0.546

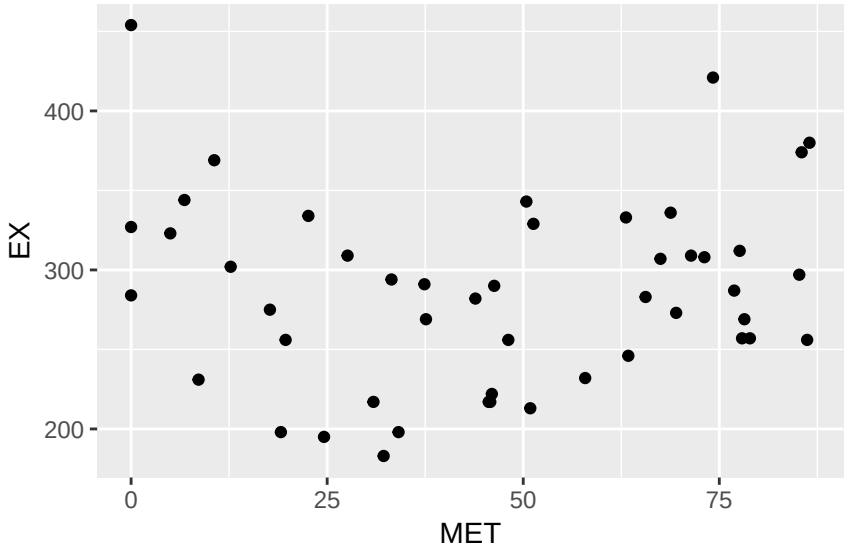
In this task we have another Loss Matrix to predict the classes by Naive Bayes. This can be done by weighting the raw predictions which we get from Naive Bayes classifier. In other words, we penalized the misclassifications. We multiplied misclassifying goods as bad by 1 and misclassifying bads as good by 10. By this way, while probability of being good is same as before, probability of being bad is 10 times higher.
Missclassification rate is higher than task 2.4, but when we examine frequencies in confusion matrixes it is clear to see that there is a significant drop classifying good. Opposite effect can be seen in count of bad classifications, it is more than double. This model is not accurate as the models as before, but it makes hard to get a loan from the bank. This reason makes this model suitable for some cases.

Assignment 3: Incertainty Estimation

The data file State.csv contains per capita state and local public expenditures and associated state demographic and economic characteristics, 1960, and there are variables
- MET: Percentage of population living in standard metropolitan areas
- EX: Per capita state and local public expenditures ($)

Task 3.1

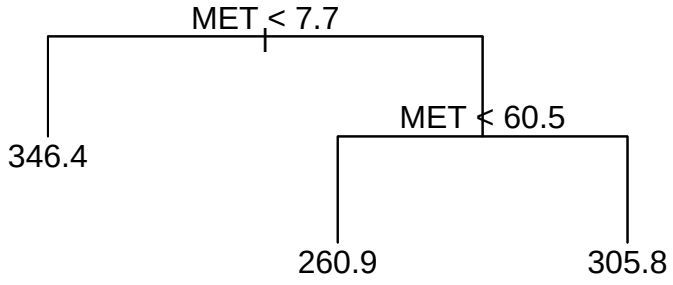
Reorder your data with respect to the increase of MET and plot EX versus MET. Discuss what kind of model can be appropriate here. Use the reordered data in steps 2-5.

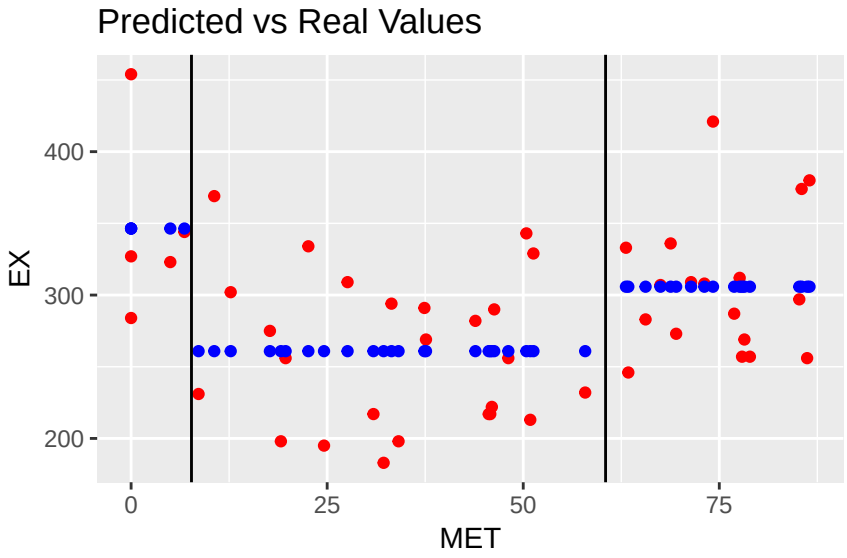


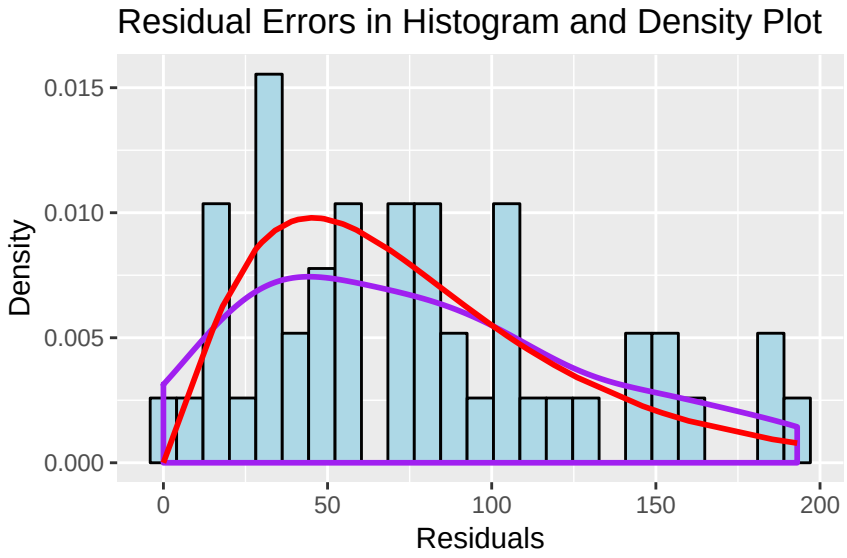
By looking these points, it is reasonable to fit a quadratic linear regression. Because in the beginning the points are in decresement movement and around middle of MET axis they start to increase.

Task 3.2

Use package tree and fit a regression tree model with target EX and feature MET in which the number of the leaves is selected by cross-validation, use the entire data set and set minimum number of observations in a leaf equal to 8 (setting minsize in tree.control). Report the selected tree. Plot the original and the fitted data and histogram of residuals. Comment on the distribution of the residuals and the quality of the fit.





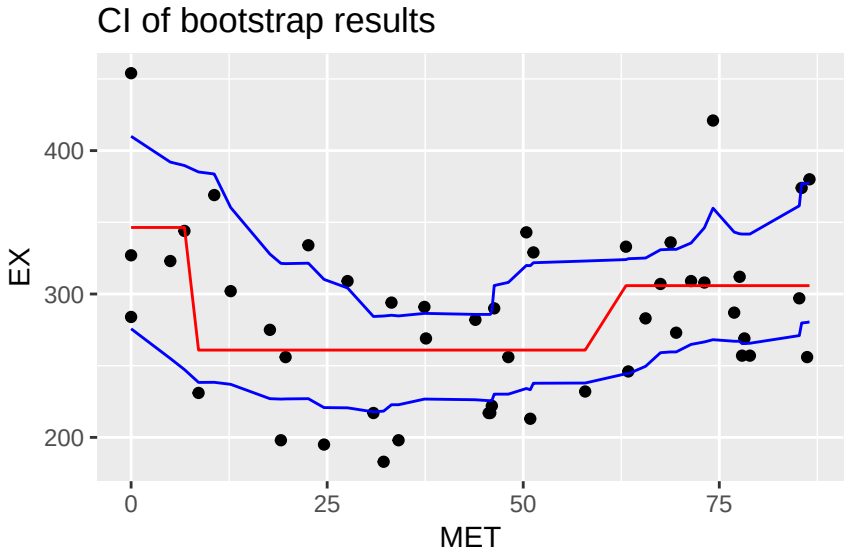



```
##      shape       rate
## 2.39825128 0.03079419
```

In this task, we use cross-validation method in order to find optimal node count in the leaves. The optimal tree has 3 nodes in the leaves and we can see the plot of its above. We have only one variable to predict the target variable. The tree takes the first split at 7.7 of MET variable. If it is less then 7.7 the prediction is 346.4, else we have another split at 60.5 and that is all for optimal tree. We can see this predictions and true values for the data in the scatter plot above. Furthermore, we can also see decision boundaries of this tree. After this fit and exploration, we calculate residual errors and plot as a histogram. The blue curve is imperical density of errors and the red curve is the gamma distribution which has 2.398 as shape and 0.0308 as rate which is calculated by fitdistrplus package in order to find the fitted gamma density function to this errors. We can say this two distribution seems similar and residual error distribution has same behaviour with gamma distribution. In regression models, it is usual to have residual errors with a normal distribution, so we cannot say the quality of the fit is not good that much.

Task 3.3

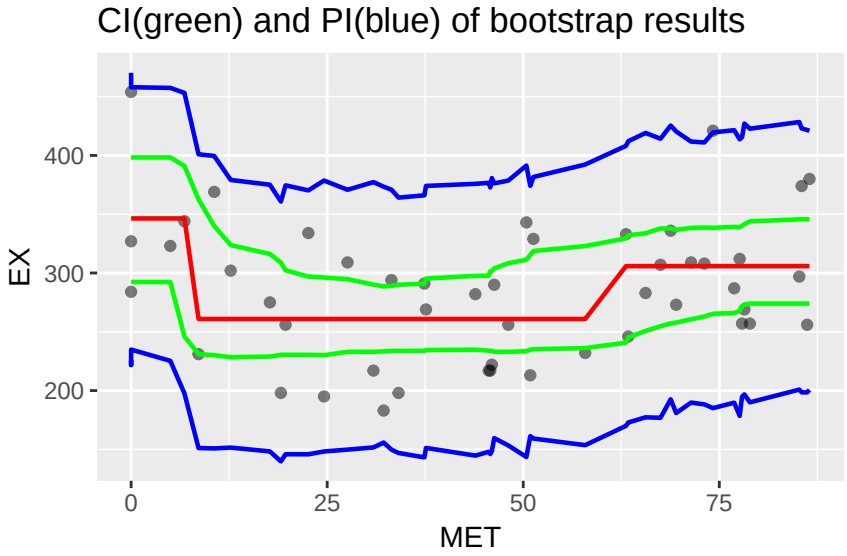
Compute and plot the 95% confidence bands for the regression tree model from step 2 (fit a regression tree with the same settings and the same number of leaves as in step 2 to the resampled data) by using a non-parametric bootstrap. Comment whether the band is smooth or bumpy and try to explain why. Consider the width of the confidence band and comment whether results of the regression model in step 2 seem to be reliable.



In this task we apply non-parametric bootstrap to compute 95% confidence band by using the same settings in task 3.2. Non-parametric bootstrap runs without knowing distribution of data, and takes samples from original data by resampling which means same observations can be inside the sample. We run bootstrap 1000 times, train tree with the sample that is taken in each iteration and get predictions for all samples. We used this predictions for computing the confidence band. We get 95% confidence interval which is in the plot above. We can see the interval band is bumpy, because we are getting samples which are correlated to our original data, but we do not have many information about distribution of our data. More over we have seen our residual distribution is not normal, so it is normal to not get a smoot interval. Additionally this regression model may not the best option, but it looks inside of the confidence bands, so we can say that the model seems reliable in this case.

Task 3.4

Compute and plot the 95% confidence and prediction bands the regression tree model from step 2 (fit a regression tree with the same settings and the same settings and the same number of leaves as in step 2 to the resampled data) by using a parametric bootstrap, assume $Y \sim N(\mu_i, \sigma^2)$ where μ_i are labels in the tree leaves and σ^2 is the residual variance. Consider the width of the confidence band and comment whether results of the regression model in step 2 seem to be reliable. Does it look like only 5% of data are outside the prediction band? Should it be?



In this case we use parametric bootstrap for computing 95% confidence and prediction bands. We assume data is Normal Distribution. We are taking samples from Normal distribution which has mean as labels in the tree leaves and variance is residual variance. We repeat this bootstrap method 1000 times in order to calculate bands. If we look confidence band (green), it looks contain all our model, so our model is still reliable. If we examine prediction band, we can see there is less data than 5%,

Task 3.5

Consider the histogram of residuals from step 2 and suggest what kind of bootstrap is actually more appropriate here.

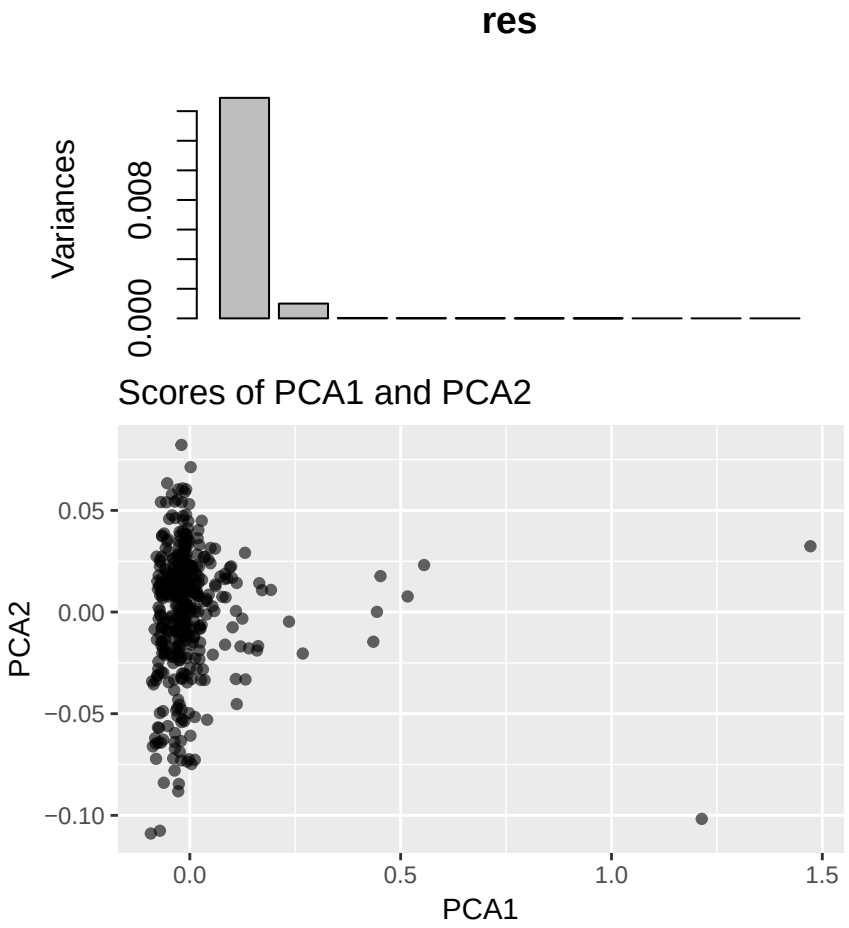
In the histogram of residuals, we can see a distribution like Gamma. Despite of low observation amount (48 observations), it is better to use non-parametric bootstrap, because we do not know exact distribution of the data. Because non-parametric bootstrap is more suitable for unknown distributed data.

Assignment 4. Principal Components

The data file NIRspectra.csv contains near-infrared spectra and viscosity levels for a collection of diesel fuels. Your task is to investigate how the measured spectra can be used to predict the viscosity.

Task 4.1

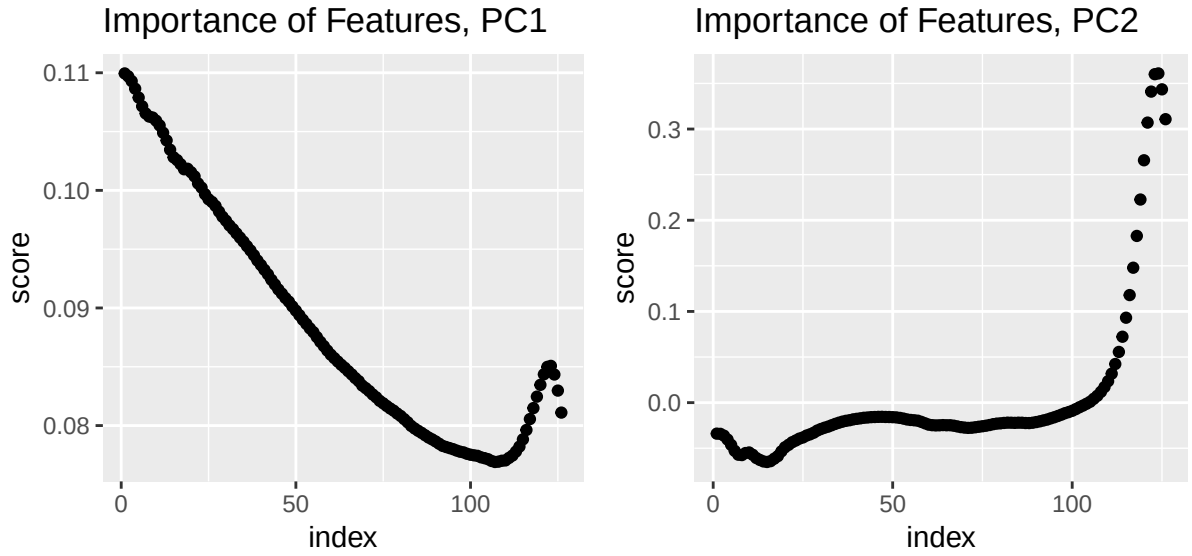
Conduct a standard PCA by using the feature space and provide a plot explaining how much variation is explained by each feature. Does the plot show how many PC should be extracted? Select the minimal number of components explaining at least 99% of the total variance. Provide also a plot of the scores in the coordinates (PC1, PC2). Are there unusual diesel fuels according to this plot?



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

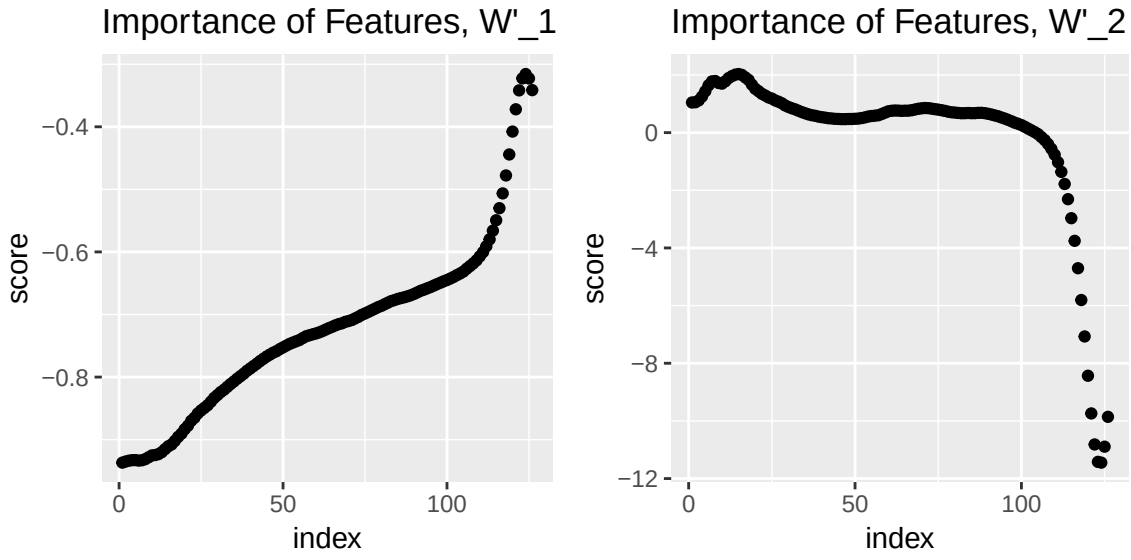
Make trace plots of the loadings of the components selected in step 1. Is there any principle component that is explained by mainly a few original features?

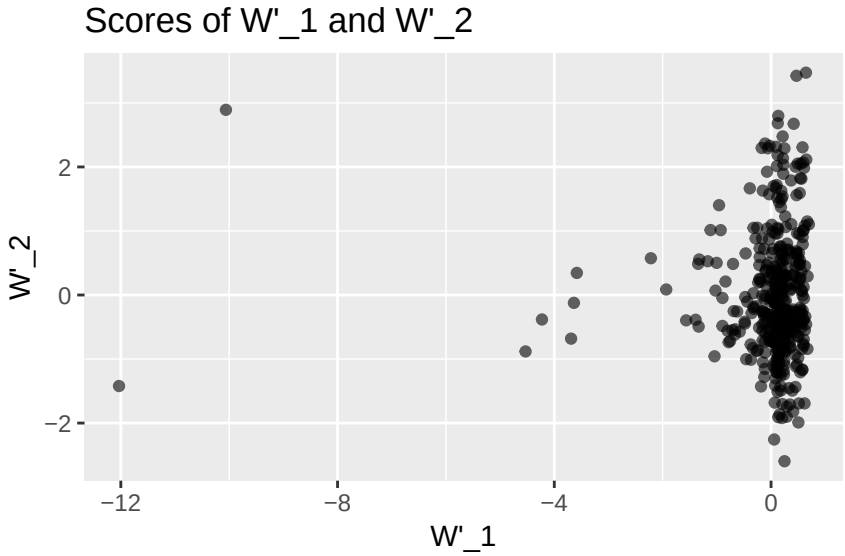


In the first plot, feature value in principle component decreases until around 110 and after that it increases again. But even the lowest one is close to 0.08.
In the second plot nearly all aff them close to zero, so it can be explained by a few original features.

Task 4.3

Perform Independent Component Analysis with the number of components selected in step 1 (set seed 12345). Check the documentation for the fastICA method in R and do the following:
a. Compute $W' = K \cdot W$ and present the columns of W' in form of the trace plots. Compare with the trace plots in step 2 and make conclusions. What kind of measure is represented by the matrix W'?
b. Make a plot of the scores of the first two latent features and compare it with the score plot from step 1.





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)   # for reading excel files
library(tree)     # for tree
library(e1071)    # for naive bayes model
library(ggplot2)  # for plots
library(boot)     # for bootstrap
library(fastICA)  # fastICA tool
library(gridExtra)
library(kableExtra)
conf_matrix = function(real_data, predicted_data){
  # make the values factor
  real_data = as.factor(real_data)
  predicted_data = factor(predicted_data, levels = levels(real_data))
  # we can have "not predicted" values in predicted data
  # make equal the levels of predicted values with real data
  levels(predicted_data) = levels(real_data)

  ct = table(real_data, predicted_data)
  df = data.frame(c(as.vector(ct[,1]), sum(ct[,1])),
                  c(as.vector(ct[,2]), sum(ct[,2])),
                  c(sum(ct[1,]), sum(ct[2,]), sum(ct)))
  rownames(df) = c("bad", "good", "Frequencies")
  colnames(df) = c("bad", "good", "Frequencies")
  return(df)
}

calculate_rate = function(conf_matrix){
  conf_matrix = as.matrix(conf_matrix)
  return(1 - sum(diag(conf_matrix[1:2,1:2]))/sum(conf_matrix[1:2,1:2]))
}
kable_cm = function(cm, capture){
  return(kableExtra::kable(cm, "latex", booktabs = T, align = "c",
             caption = capture) %>%
             row_spec(2, hline_after = T) %>%
             column_spec(c(1,3), border_right = T) %>%
             kable_styling(latex_options = "hold_position"))
}

################################### TASK 2.1 ###################################

# read data
df_credit = read_xls("../dataset/creditscoring.xls")
df_credit$good_bad = as.factor(df_credit$good_bad)
# split data
n = dim(df_credit)[1]
set.seed(12345)
id = sample(1:n, floor(n*0.5))
train = df_credit[id,]
id1 = setdiff(1:n, id)
```

```r
set.seed(12345)
id2 = sample(id1, floor(n*0.25))
valid = df_credit[id2,]
id3 = setdiff(id1,id2)
test = df_credit[id3,]

################################### TASK 2.2 ###################################

# fit decision trees with different split criterions
tree_deviance = tree(good_bad~., data = train, split="deviance")
tree_gini = tree(good_bad~., data = train, split="gini")

# predictions
pred_devi_train = predict(tree_deviance, train, type = "class")
pred_devi_test = predict(tree_deviance, test, type = "class")
pred_gini_train = predict(tree_gini, train, type = "class")
pred_gini_test = predict(tree_gini, test, type = "class")

# confusion matrix
cm_devi_train = conf_matrix(train$good_bad, pred_devi_train)
cm_devi_test = conf_matrix(test$good_bad, pred_devi_test)
cm_gini_train = conf_matrix(train$good_bad, pred_gini_train)
cm_gini_test = conf_matrix(test$good_bad, pred_gini_test)

df_misclass = data.frame(test = c(calculate_rate(cm_devi_test),
                                  calculate_rate(cm_gini_test)),
                         train = c(calculate_rate(cm_devi_train),
                                   calculate_rate(cm_gini_train)),
                         row.names = c("Deviance", "Gini Index"))

kableExtra::kable(df_misclass, align = "c",
             caption = "Missclassification Rates") %>%
  kable_styling(latex_options = "hold_position")


################################### TASK 2.3 ###################################

# we have 15 nodes in our tree. We will create
node_count = (summary(tree_deviance))$size
trainScore = c()
testScore = c()

for(i in 2:node_count){
  prunedTree = prune.tree(tree_deviance, best = i)
  pred = predict(prunedTree, newdata = valid, type = "tree")
  trainScore[i-1] = deviance(prunedTree)
  testScore[i-1] = deviance(pred)
}
# store the values into df
plot_df = data.frame(node_qty = rep(c(2:node_count),2),
                     score = c(testScore, trainScore),
                     newdata = c(rep("validation",14), rep("train",14)))
```

```r
plot_opt_scores = ggplot(plot_df, aes(x=node_qty, y = score, color=newdata)) +
  geom_point() +
  geom_line() +
  scale_x_continuous(breaks = seq(2,node_count,2)) +
  labs(title = "Scores of Pruned Trees", x="Node Count", y = "Score")

# we chose the tree which has lowest test error
# we add 1 to index, because indexes starts from 1 but index 1 has 2 nodes.
# so: node_count = index + 1
optimal_node_count = which.min(testScore) + 1
optimal_tree = prune.tree(tree_deviance, best = optimal_node_count)

pred_opt_tree = predict(optimal_tree, test, type="class")
cm_opt = conf_matrix(test$good_bad, pred_opt_tree)
mis = calculate_rate(cm_opt)


plot_opt_scores
# plot for optimal tree
plot(optimal_tree)
text(optimal_tree, pretty=0)
kableExtra::kable(data.frame(test = mis,
                             row.names = c("Misclassification")),
                  align = "c",
                  caption = "Missclassification Rates") %>%
  kable_styling(latex_options = "hold_position")

kable_cm(cm_opt, "Confusion Matrix of Optimal Tree")

################################### TASK 2.4 ###################################

# fit model
fit_naive = naiveBayes(good_bad~., data=train)
# predictions
pred_naive_train = predict(fit_naive, newdata=train)
pred_naive_test = predict(fit_naive, newdata=test)
# confusion matrix
cm_naive_train = conf_matrix(train$good_bad, pred_naive_train)
cm_naive_test = conf_matrix(test$good_bad, pred_naive_test)
# misclassification rates
df_misclass = data.frame(train = c(calculate_rate(cm_naive_train)),
                         test = c(calculate_rate(cm_naive_test)),
                         row.names = c("Misclassification"))

kable_cm(cm_naive_train, "Confusion Matrix of Naive Bates / Train Data")

kable_cm(cm_naive_test, "Confusion Matrix of Naive Bates / Test Data")


kableExtra::kable(df_misclass, align = "c",
            caption = "Missclassification Rates") %>%
  kable_styling(latex_options = "hold_position")
```

```
################################### TASK 2.5 ###################################

# function that apllies new classifier principle
# for only this dataset!
custom_predict = function(probs, threshold){
  return(as.vector(ifelse(probs>threshold, 1, 0)))
}

calculate_tp_fp_rates = function(conf_matrix){
  TPR = conf_matrix[2,2] / conf_matrix$Frequencies[2]
  FPR = conf_matrix[1,2] / conf_matrix$Frequencies[1]
  return(list(tpr = TPR, fpr = FPR))
}

roc_range = function(real, probs, from=0.05, to=0.95, step=0.05){
  if(from<0 | to>1)
    stop("from and to values have to be in range [0,1]")
  if(step<0 | step>1)
    stop("step value has to be in range [0,1]")
  fpr = c()
  tpr = c()
  for(pi in seq(from, to, by=step)){
    # confision matrix
    cm = conf_matrix(real, custom_predict(probs, pi))
    # calculate tp and fp rates
    rates = calculate_tp_fp_rates(cm)
    tpr = c(tpr, rates$tpr)
    fpr = c(fpr, rates$fpr)
  }
  return(data.frame(tpr = tpr,
                    fpr = fpr))
}

# probs from tree (which are 'good')
probs_tree_test = predict(optimal_tree, test)[,2]
probs_naive_test = predict(fit_naive, newdata=test, type="raw")[,2]
# probs_tree_train = predict(tree_deviance, train)[,2]
# probs_naive_train = predict(fit_naive, newdata=train, type="raw")[,2]

real_values_bin = ifelse(test$good_bad=="good",1,0)

# calculate roc values in range
df_tree_roc = roc_range(real = real_values_bin, probs = probs_tree_test)
df_tree_roc$model = "tree"
df_naive_roc = roc_range(real = real_values_bin, probs = probs_naive_test)
df_naive_roc$model = "naive"

# get plot df
df_plot = rbind(df_naive_roc, df_tree_roc)

plot_roc = ggplot() +
  geom_line(aes(x = df_plot$fpr, y = df_plot$tpr, color = df_plot$model),
            size = 0.8) +
```

```
  geom_line(aes(x = c(0,1), y = c(0,1)), size=1, linetype=2) +
  labs(title = "ROC Curve of Naive and Tree Models",
       x = "FPR", y = "TPR", color = "Model")

plot_roc

################################### TASK 2.6 ###################################

# implement loss matrix
loss_matrix = matrix(c(0,1,
                       10,0), ncol=2, byrow = T)
# predict original probabilities with naive bayes
probs_naive_test = predict(fit_naive, newdata=test, type="raw")
probs_naive_train = predict(fit_naive, newdata=train, type="raw")
# calculate weighted probabilities with loss_matrix
weighted_probs_test = t(loss_matrix %*% t(probs_naive_test))
weighted_probs_train = t(loss_matrix %*% t(probs_naive_train))
# predict values
# first column values are the probabilities of being good.
# because in  matrix multiplicaition we multiplied weight of good first (1st row)
predicts_test = ifelse(weighted_probs_test[,1]>weighted_probs_test[,2],
                       "good", "bad")
predicts_train = ifelse(weighted_probs_train[,1]>weighted_probs_train[,2],
                        "good", "bad")
# calculate cm
cm_test = conf_matrix(test$good_bad, predicts_test)
cm_train = conf_matrix(train$good_bad, predicts_train)

df_misclass = data.frame(test = calculate_rate(cm_test),
                         train = calculate_rate(cm_train),
                         row.names = c("Misclassification"))

kable_cm(cm_train, "Confusion Matrix for Naive Bayes / Train Data")
kable_cm(cm_test, "Confusion Matrix for Naive Bayes / Test Data")

kableExtra::kable(df_misclass, align = "c",
            caption = "Missclassification Rates") %>%
  kable_styling(latex_options = "hold_position")

################################### TASK 3.1 ###################################

df_state = read.csv2("../dataset/State.csv")

df_state = df_state[order(df_state$MET),]
rownames(df_state) = 1:dim(df_state)[1]

plot_met_ex = ggplot(df_state, aes(x = df_state$MET, y = df_state$EX)) +
  geom_point() +
  labs(x = "MET", y = "EX")

plot_met_ex
```

```
################################### TASK 3.2 ###################################

# get first regression tree
reg_tree = tree(formula = EX ~ MET, data = df_state, control = tree.control(nobs=dim(df_state)[1], mins
# apply cv to find best leave count
cv_tree = cv.tree(object = reg_tree,
                  tree.control(nobs=dim(df_state)[1], minsize = 8))
# find the optimal leave count
optimal_node_count = cv_tree$size[which.min(cv_tree$dev)]
# prune tree for optimal leave count
optimal_tree = prune.tree(reg_tree, best = optimal_node_count)
# predict values with optimal tree
predicts = predict(object = optimal_tree, newdata = df_state)
# calculate residuals
residuals = df_state$EX - predicts
# shift residuals to zero. Because distribution cannot contain negative values
shifted_residuals = residuals + abs(min(residuals))
# get shifted residuals as df
df = as.data.frame(shifted_residuals)
# I plot first, and the imperical distribution line with geom_density
# seems like gamma distribution. following lines find the parameters of gamma
# distribution for the data.
fit_gamma <- fitdistrplus::fitdist(shifted_residuals, distr = "gamma", "mme")

# Red is original gamma distribution
# blue is imperical distribution that we add geom_density fucntion
plot_hist = ggplot(df, aes(x=shifted_residuals)) +
              geom_histogram(aes(y=..density..), bins = 25,
                             col="black", fill="lightblue") +
              geom_density(aes(y=..density..), color="Purple", size=1) +
              geom_line(aes(y = dgamma(shifted_residuals, fit_gamma$estimate[1],
                               fit_gamma$estimate[2])), color = "Red", size=1) +
              labs(title="Residual Errors in Histogram and Density Plot",
                   x = "Residuals", y = "Density")

# Predicted vs Real values plot
# red is real values | blue is predicted values
df = data.frame(x=df_state$MET, real = df_state$EX, pred=predicts)
plot_pred = ggplot(df) +
              geom_point(aes(x = x, y = real), color="red") +
              geom_point(aes(x = x, y = pred), color="blue") +
              geom_vline(xintercept = c(7.7, 60.5)) +
              labs(title = "Predicted vs Real Values",
                   x = "MET", y = "EX")

plot(optimal_tree)
text(optimal_tree)
plot_pred
plot_hist
fit_gamma$estimate

################################### TASK 3.3 ###################################
```

```r
# function that we use for bootstrap
f = function(data, ind){
  data1 = data[ind,]# extract bootstrap sample
  # fit tree
  reg_tree = tree(formula = EX ~ MET, data = data1, control=tree.control(nobs=nrow(data1), minsize=8))
  # prune the optimal tree
  optimal_node_count = 3
  optimal_tree = prune.tree(reg_tree, best = optimal_node_count, newdata = data1)
  #predict values for all Area values from the original data
  predictions = predict(optimal_tree, newdata=data)
  return(predictions)
}
# run the bootstrap for 1000 samples for each observation with f function
res = boot(df_state, f, R=1000)
# calculate confidence interval
ci = (envelope(res, level = 0.95))$point
# creat plot df for confidence interval
df_ci = data.frame(x=df_state$MET,
                   real = df_state$EX,
                   ci_low = ci[2,],
                   ci_mean = res$t0,
                   ci_high = ci[1,])

# plot ci
plot_ci = ggplot(df_ci, aes(x=x)) +
  geom_point(aes(y = real), color="black") +
  geom_line(aes(y = ci_low), color="blue") +
  geom_line(aes(y = ci_mean), color="red") +
  geom_line(aes(y = ci_high), color="blue") +
  labs(title = "CI of bootstrap results", x = "MET", y = "EX")

plot_ci

################################### TASK 3.4 ###################################

# our mle is optimal tree from task 3.2
mle = optimal_tree
pred = predict(mle, df_state)
# data generator function
rng = function(data, mle){
  data1 = data.frame(EX = data$EX, MET = data$MET)
  n = nrow(data)
  # generate new EX
  pred = predict(mle, newdata = data1)
  res = data$EX - pred
  data1$EX = rnorm(n, pred, sd(res))
  return(data1)
}

f1 = function(data1){
  n = length(df_state$EX)
  # train tree
  reg_tree = tree(formula = EX ~ MET, data = data1,
```

```
                  control=tree.control(nobs=nrow(data1), minsize=8))
  # prune the optimal tree
  optimal_node_count = 3
  optimal_tree = prune.tree(reg_tree, best = optimal_node_count, newdata = data1)
  # predict
  pred = predict(optimal_tree, newdata = df_state)
  res = df_state$EX - pred
  predicted = rnorm(n, pred, sd(res))
  return(predicted)
}

# for ci use this!
f2 = function(data1){
  n = length(df_state$EX)
  # train tree
  reg_tree = tree(formula = EX ~ MET, data = data1,
                  control=tree.control(nobs=nrow(data1), minsize=8))
  # prune the optimal tree
  optimal_node_count = 3
  optimal_tree = prune.tree(reg_tree, best = optimal_node_count)
  # predict
  pred = predict(optimal_tree, newdata = df_state)
  return(pred)
}

# prediction band
res = boot(df_state, statistic=f1, R=1000, mle=mle,
           ran.gen=rng, sim="parametric")
pi = (envelope(res, level = 0.95))$point

# confidence band
res = boot(df_state, statistic=f2, R=1000, mle=mle,
           ran.gen=rng, sim="parametric")
ci = (envelope(res, level = 0.95))$point

# creat plot df for confidence interval
df_ci = data.frame(x=df_state$MET,
                   real = df_state$EX,
                   pi_low = pi[2,],
                   ci_mean = res$t0,
                   pi_high = pi[1,],
                   ci_low = ci[2,],
                   ci_high = ci[1,])
# plot ci
plot_ci = ggplot(df_ci, aes(x=x)) +
  geom_point(aes(y = real), color="black", alpha=0.5) +
  geom_line(aes(y = pi_low), color="blue", size = 0.8) +
  geom_line(aes(y = pred), color="red", size = 0.8) +
  geom_line(aes(y = pi_high), color="blue", size = 0.8) +
  geom_line(aes(y = ci_low), color="green", size = 0.8) +
  geom_line(aes(y = ci_high), color="green", size = 0.8) +
  labs(title = "CI(green) and PI(blue) of bootstrap results", x = "MET", y = "EX")
```

```
plot_ci


################################### 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")
```

```r
################################## 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
```