Machine Learning Lab 02

Maximilian Pfundstein (maxpf364)

2018-12-08

Contents

1 Assignment 2: Analysis of Credit Scoring

1.1 Import `creditscoring.xls`

Let's import the data and have a look at it.

Table 1: creditscoring.xls

resident	property	age	other	housing	existcr	job	depends	telephon	foreign	good_bad
4	1	67	3	2	2	3	1	2	1	good

resident	property	age	other	housing	existcr	job	depends	telephon	foreign	good_bad
2	1	22	3	2	1	3	1	1	1	bad
3	1	49	3	2	1	2	2	1	1	good
4	2	45	3	3	1	3	2	1	1	good
4	4	53	3	3	2	3	2	1	1	bad
4	4	35	3	3	1	2	2	2	1	good

1.2 Decision Tree Fitting

Task: Fit a decision tree to the training data by using the following measures of impurity:

a. Deviance
b. Gini index

1.2.1 Deviance

The model for the decision tree using deviance.

```
##
## Classification tree:
## tree(formula = good_bad ~ ., data = train, split = "deviance")
## Variables actually used in tree construction:
## [1] "savings"  "duration" "history"  "age"      "purpose"  "amount"
## [7] "resident" "other"
## Number of terminal nodes:  15
## Residual mean deviance:  0.9569 = 458.3 / 479
## Misclassification error rate: 0.2105 = 104 / 494
```

The confusion matrix looks as follows:

	bad	good
bad	28	19
good	48	155

Therefore the error rate is:

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

1.2.2 Gini

The model for the decision tree using gini.

```
##
## Classification tree:
## tree(formula = good_bad ~ ., data = train, split = "gini")
## Variables actually used in tree construction:
##  [1] "foreign"  "coapp"    "depends"  "telephon" "existcr"  "savings"
##  [7] "history"  "property" "marital"  "duration" "employed" "age"
## [13] "housing"  "amount"   "purpose"  "resident" "job"      "installp"
## Number of terminal nodes:  72
## Residual mean deviance:  1.015 = 428.5 / 422
## Misclassification error rate: 0.2368 = 117 / 494
```

The confusion matrix looks as follows:

	bad	good
bad	18	35
good	58	139

Therefore the error rate is:

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

1.2.3 Conclusions

Question: Report the misclassification rates for the training and test data. Choose the measure providing the better results for the following steps.

Answer: The misqualification rate for the decision tree with deviance is `0.33` compared to the decision tree with gini as the classifier which has a misqualification rate of `0.3666667`. Therefore we will continue with using the decision tree that uses **deviance** as the classifier.

1.3 Finding the Optimal Tree

Task:

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

1.3.1 Optimal Tree Depth

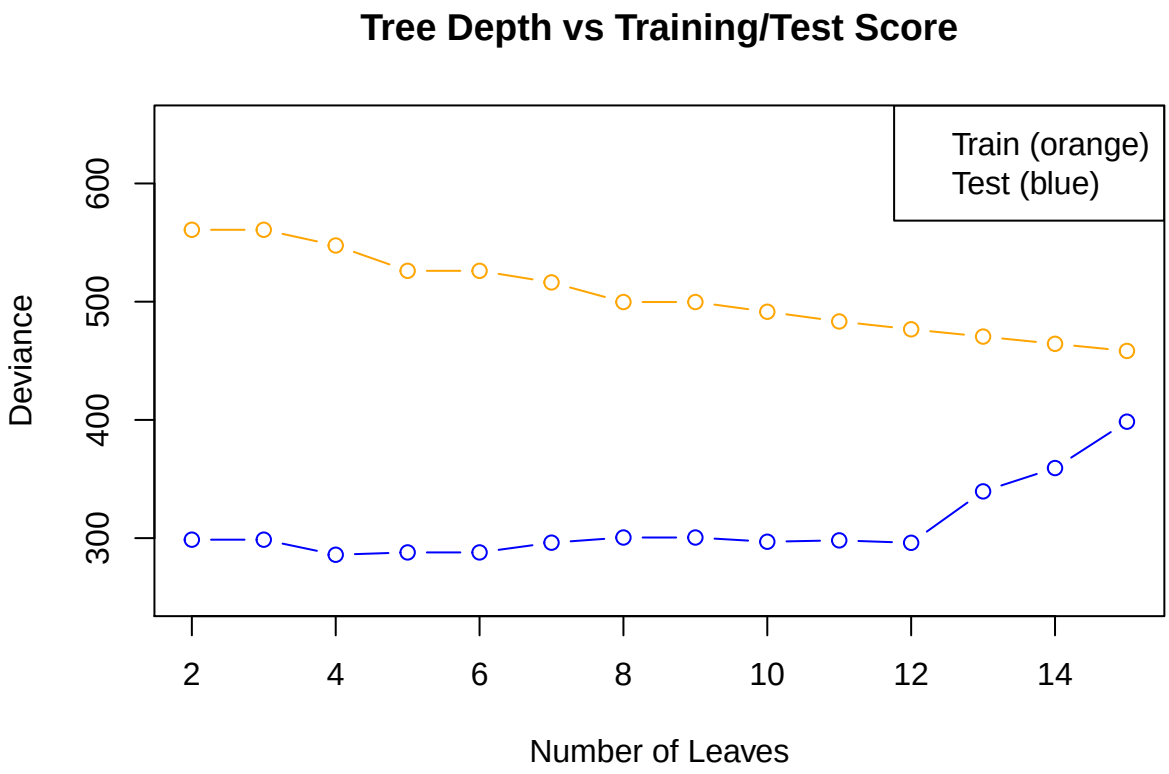
The best tree's index and deviance.

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

```
## [1] 285.9425
```

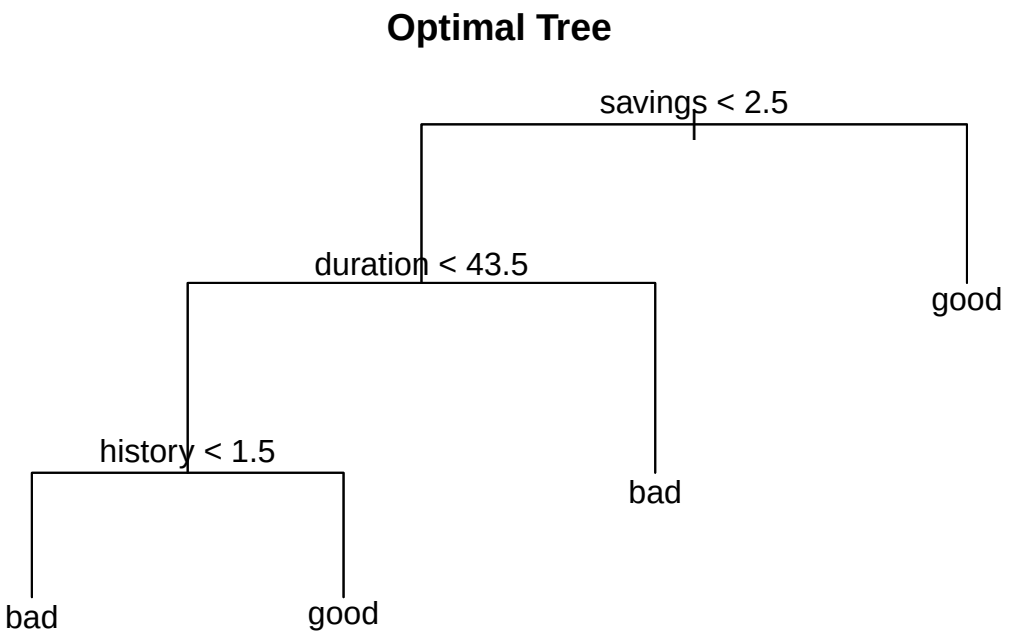
1.3.2 Dependency of Deviances

The following plot shows the number of leaves vs the deviance. The orange line indicates the training and the blue line the test deviance.



1.3.3 Optimal Tree

The following plot shows the optimal tree and it's variables. It has a depth of 4.



1.3.4 Interpretating the Tree Structure

The tree splits the data based on if the savings is smaller than `2.5`. This means this is the feature where the tree evaluated the most influence on the prediction. We can see that the right side has no further splits. On the left side the tree splits further, starting with the duration as the second most important feature. As the tree has more leaves to the left side we can see that splitting this data further makes more sense than splitting it further on the right side. The tree has a depth of 3.

1.3.5 Estimate of the Missclassification Rate

```
##
## Classification tree:
## snip.tree(tree = decisionTree_deviance, nodes = c(5L, 3L, 9L))
## Variables actually used in tree construction:
## [1] "savings"  "duration" "history"
## Number of terminal nodes:  4
## Residual mean deviance:  1.117 = 547.5 / 490
## Misclassification error rate: 0.251 = 124 / 494
```

	bad	good
bad	18	35
good	58	139

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

The last value shows the misclassification error on the test data set.

1.4 Naive Bayes

Task:

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

1.4.1 Classification with Naive Bayes

Let's train the model and have a look at the summary.

```
##         Length Class  Mode
## apriori  2     table  numeric
## tables  19     -none- list
## levels   2     -none- character
## call     4     -none- call
```

1.4.2 Naive Bayes Confusion Matrices and Misclassification Rates

Data for Naive Bayes on train:

	bad	good
bad	95	98
good	52	255

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

Data for Naive Bayes on test:

	bad	good
bad	46	49
good	30	125

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

1.4.3 Comparison with Step 3

We can see that the misqualification rate for the optimized decision tree is better than the Naive Bayes approach. We have to keep in mind that we first had to find the best tree and thus spend more time optimizing the hyper parameters.

1.5 TPR, FPR and ROC Curves

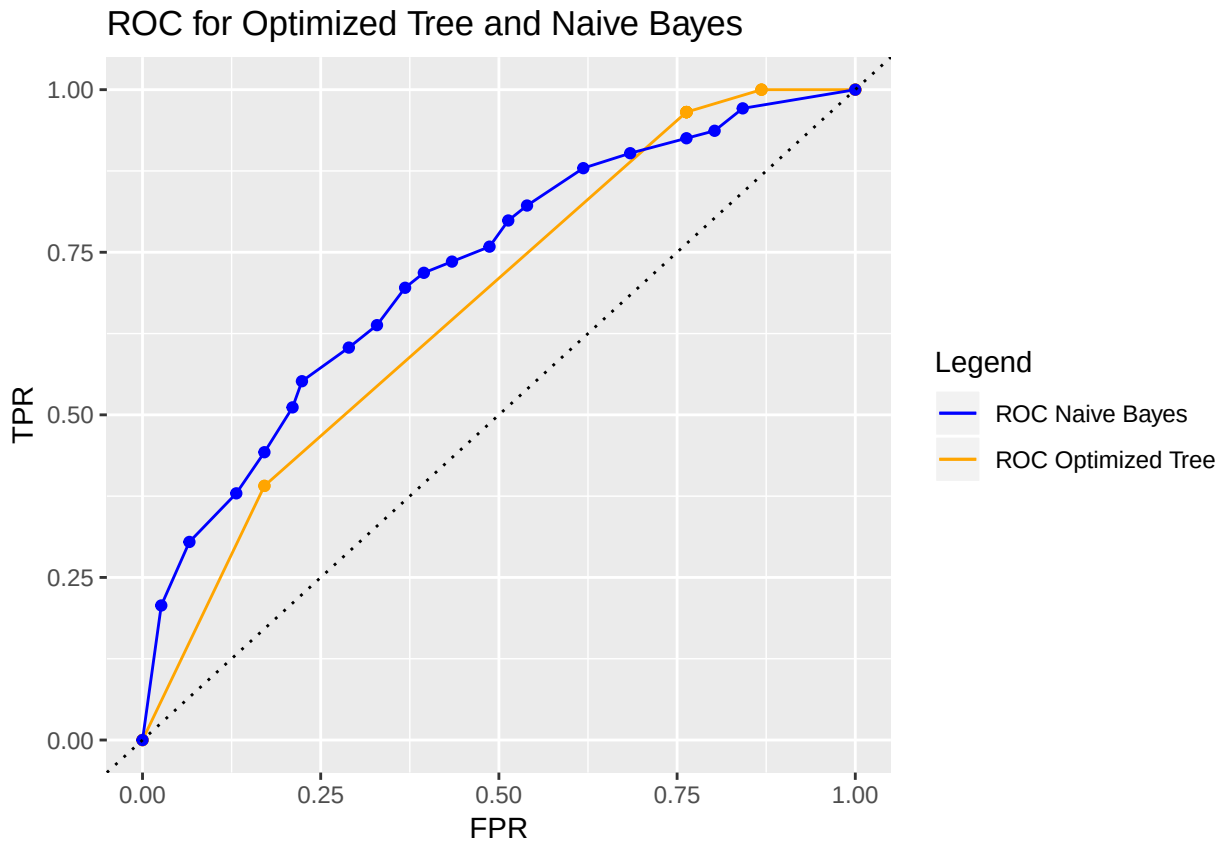
Task: Compute the TPR and FPR values for the two models.

The corresponding values for FPR and TPR can be seen in the following table.

fprs_tree	tprs_tree	fprs_bayes	tprs_bayes
1.0000000	1.0000000	1.0000000	1.0000000
1.0000000	1.0000000	0.8421053	0.9712644
1.0000000	1.0000000	0.8026316	0.9367816
1.0000000	1.0000000	0.7631579	0.9252874
0.8684211	1.0000000	0.6842105	0.9022989
0.8684211	1.0000000	0.6184211	0.8793103
0.8684211	1.0000000	0.5394737	0.8218391
0.7631579	0.9655172	0.5131579	0.7988506
0.7631579	0.9655172	0.4868421	0.7586207
0.7631579	0.9655172	0.4342105	0.7356322
0.7631579	0.9655172	0.3947368	0.7183908
0.7631579	0.9655172	0.3684211	0.6954023
0.7631579	0.9655172	0.3289474	0.6379310
0.7631579	0.9655172	0.2894737	0.6034483
0.7631579	0.9655172	0.2236842	0.5517241
0.1710526	0.3908046	0.2105263	0.5114943
0.1710526	0.3908046	0.1710526	0.4425287
0.0000000	0.0000000	0.1315789	0.3793103
0.0000000	0.0000000	0.0657895	0.3045977
0.0000000	0.0000000	0.0263158	0.2068966
0.0000000	0.0000000	0.0000000	0.0000000

Task: Plot the corresponding ROC curves.

This is the ROC curve of the Optimized Tree and Naive Bayes.



Question: Conclusion?

Answer: The ROC (receiver operating characteristic) curve shows how the models behaves for different threshold values. The greater the area under the curve the better the model can distinguish between two classes. This is due to the fact that we have overlapping distributions, which in worst case, are exactly on top of each other, which would result in a line at the 45 degree angle. As the distributations shift apart, our models will get better on average. Here we can observe, that the Naive Bayes is in general better in distinguishing the two classes, the Optimized Tree mostly performs worse. We have to keep in mind that we have few datapoints for small FPR for the tree and that we've not spent any effort interpolating the line between those two points.

1.6 Naive Bayes Classification with Loss Matrix

Task:

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

This is the given loss matrix:

	Predicted	Predicted
good	0	1
bad	10	0

Confusion Matrix for Training:

	bad	good
FALSE	137	263
TRUE	10	90

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

Confusion Matrix for Test:

	bad	good
FALSE	71	122
TRUE	5	52

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

The error rates are way higher which was to be expected. There will always be the α and β error. Choosing a loss matrix or setting a threshold for the classification will have influence on these errors. While making one of these errors small, the other one gets larger (with peaks where you classify everyhting as TRUE or FALSE). Here we defined that the confidence for being bad must be at least ten times larger than for being good which concludes in the results we've calculated and a high error rate.

2 Assignment 3: Uncertainty Estimation

2.1 Import and Plot `State.csv`

Task:

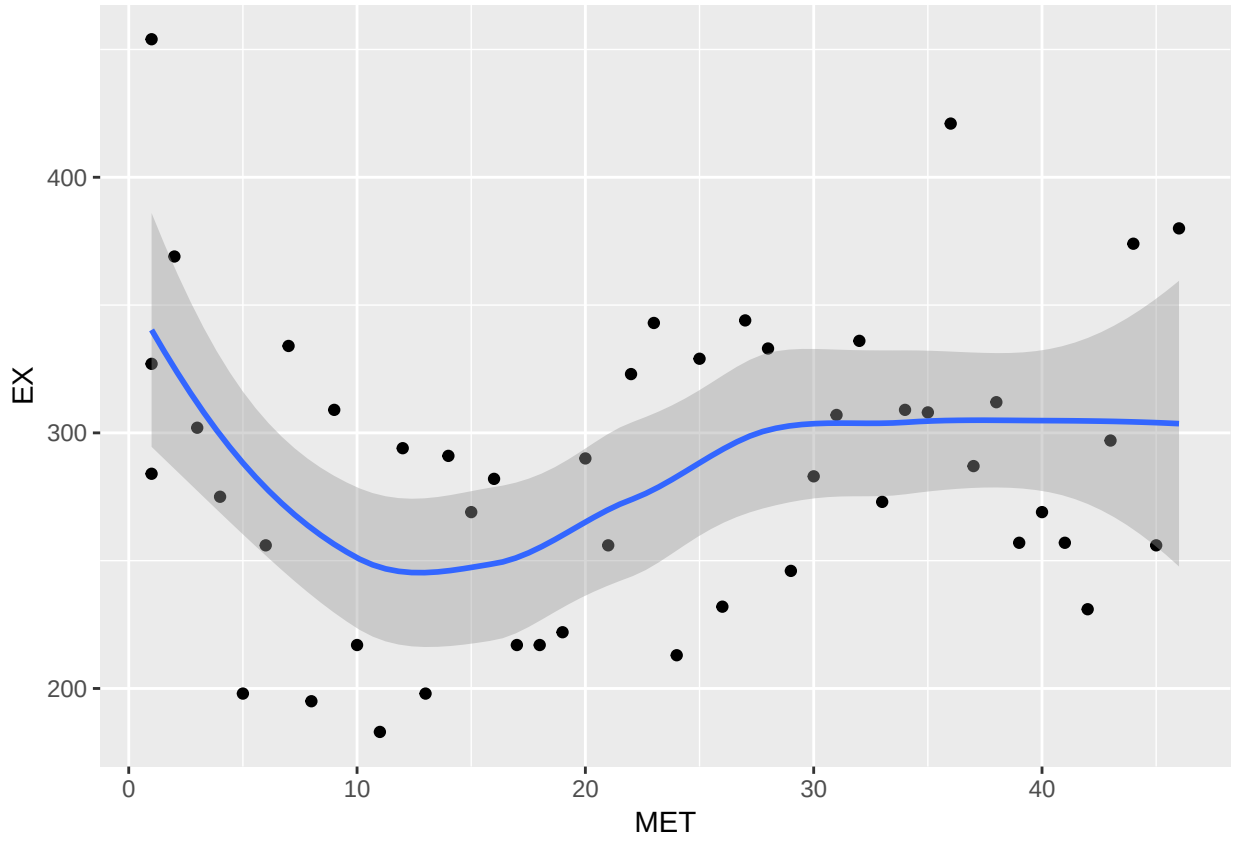
- Reorder your data with respect to the increase of MET and plot EX versus MET.
- Discuss what kind of model can be appropriate here.

Let's import the data and have a look at it:

Table 11: State.csv

EX	ECAB	MET	GROW	YOUNG	OLD	WEST	STATE
256	85,5	19,7	6,9	29,6	11	0	ME
275	94,3	17,7	14,7	26,4	11,2	0	NH
327	87	0	3,7	28,5	11,2	0	VT
297	107,5	85,2	10,2	25,1	11,1	0	MA
256	94,9	86,2	1	25,3	10,4	0	RI
312	121,6	77,6	25,4	25,2	9,6	0	CT

Let's plot the data:



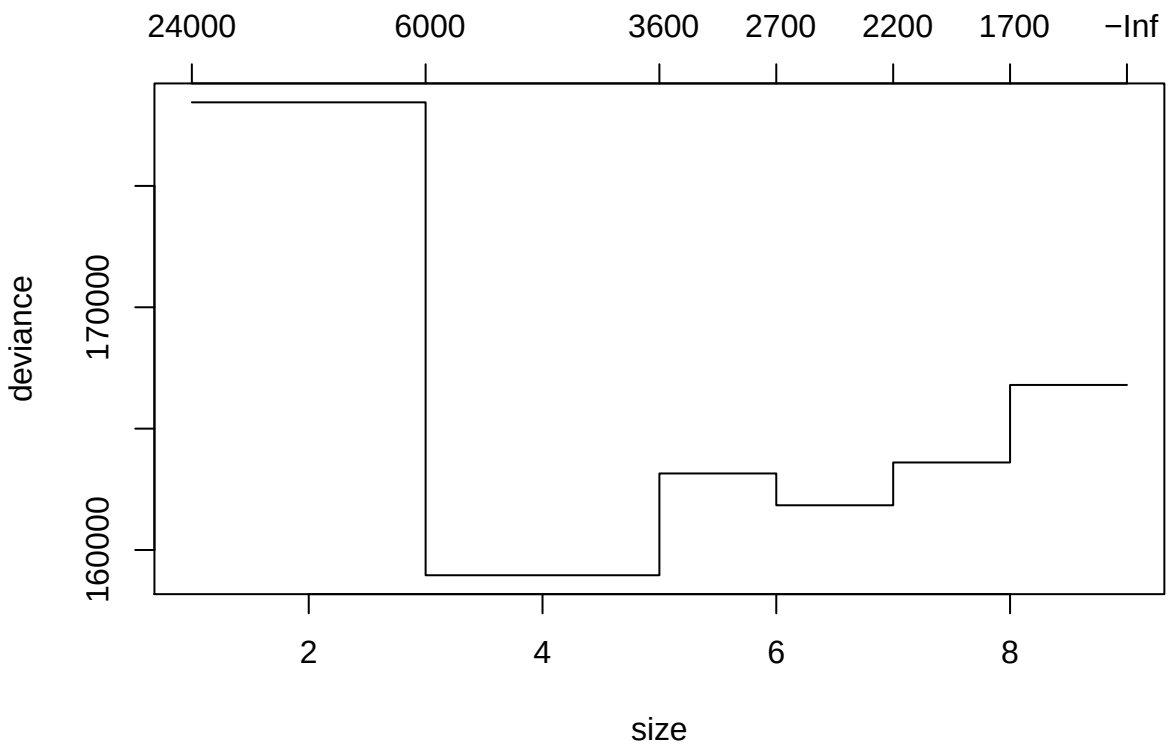
At a first glance this data looks messy, but taking a second look gives the impression that the data has different kind of levels. Therefor a Regression Tree would probably be a good model.

2.2 Regression Tree Model

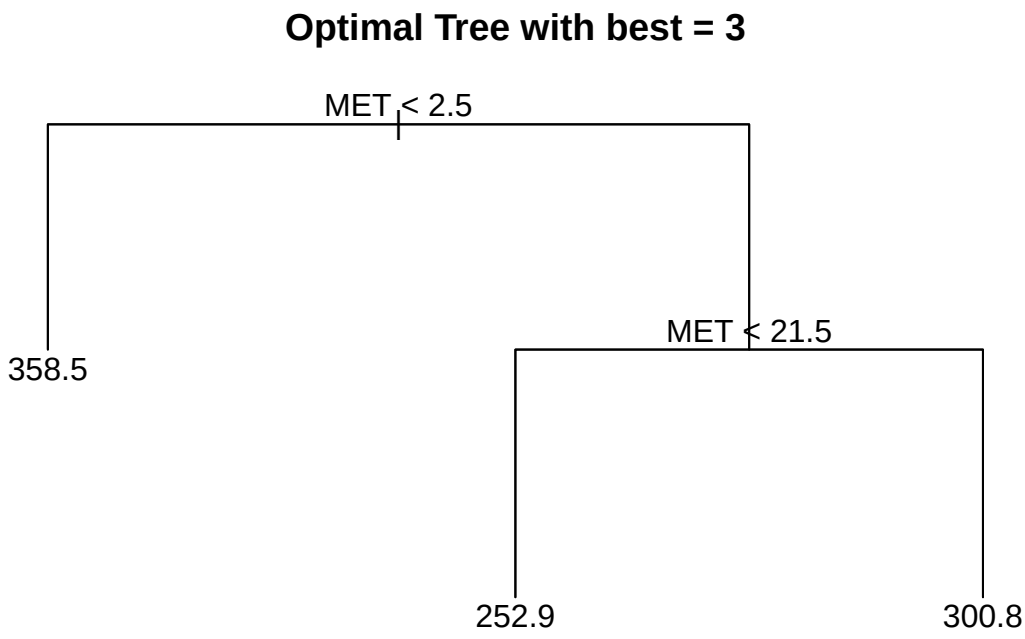
Task:

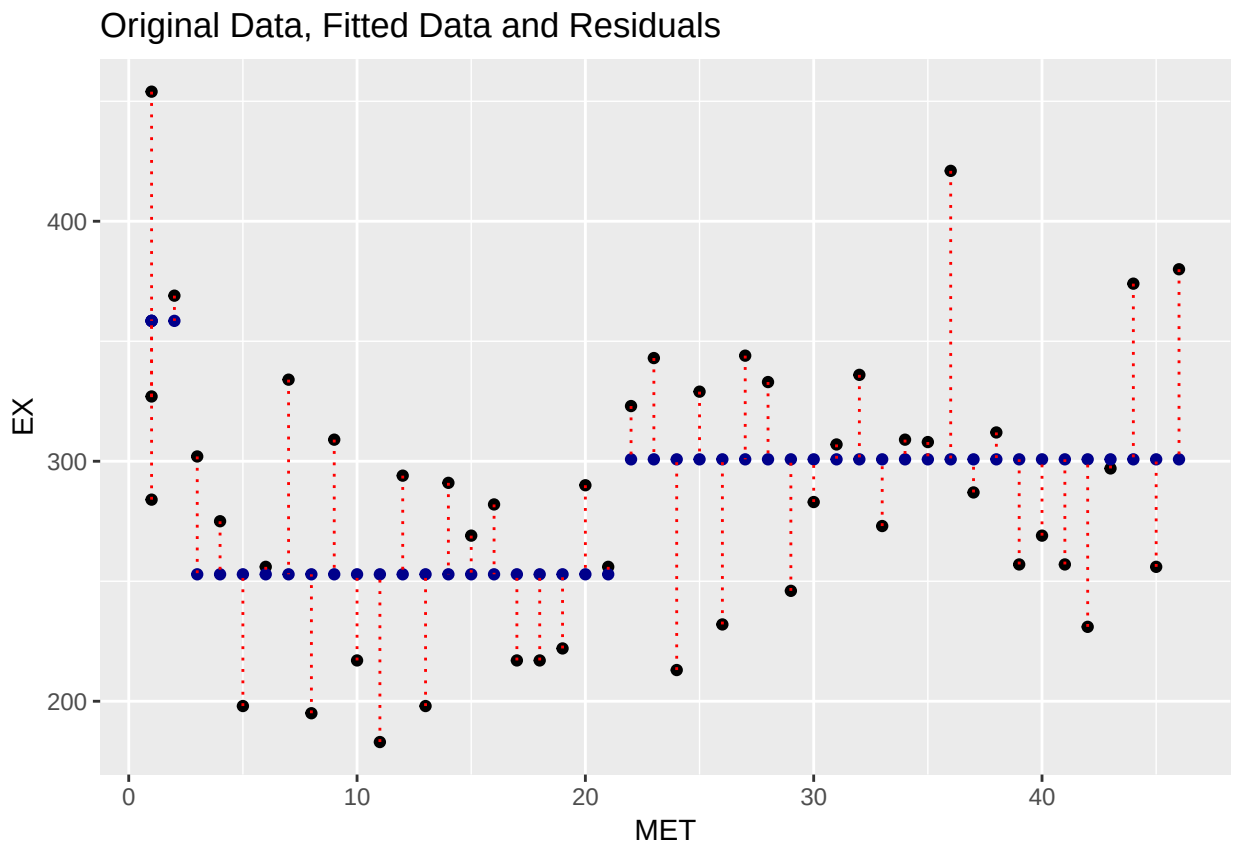
- Report the selected tree.
- Plot the original and the fitted data and histogram of residuals.
- Comment on the distribution of the residuals and the quality of the fit.

Let's create a Regression Tree Model and use Cross Validation to see which sizes is the best.

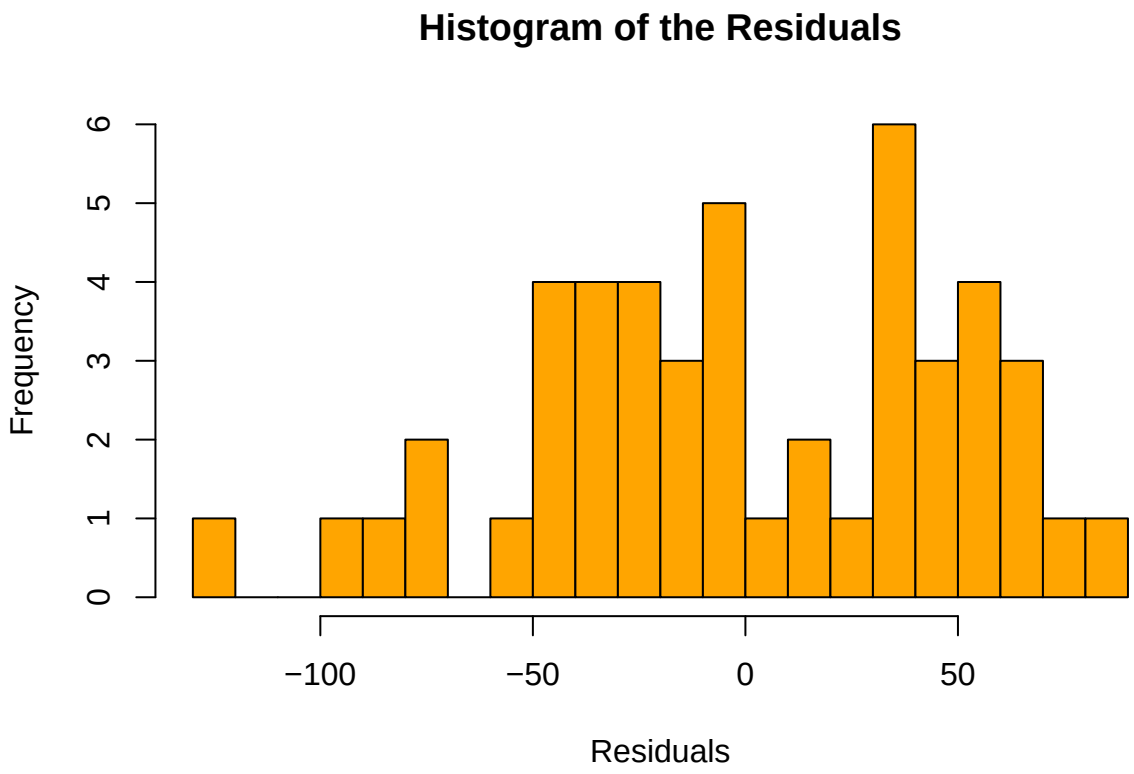


We see that 3 or 4 would work for the size of the tree. For the following we will declare `best = 3`. Let's prune our best tree and have a look at it.


Optimal Tree with best = 3
MET < 2.5
358.5
MET < 21.5
252.9
300.8



And here we have the histrogram of the residuals.

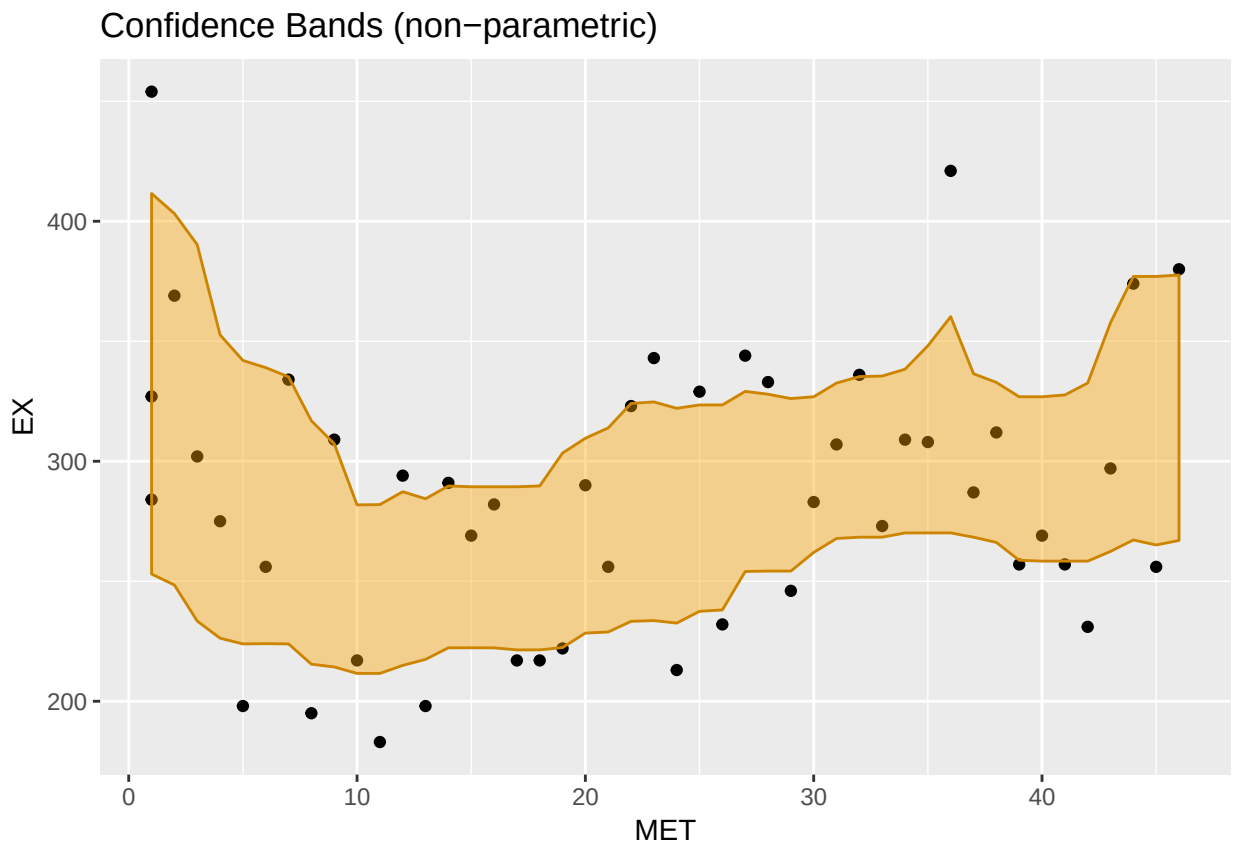


The histogramm looks like a Chi-Squared distribution with `k ~ 3`. As our tree has only three levels of outcome, we have only "three lines" in our plot (blue dots). They only vary in length and their EX value. As we only have those three levels, the quality of this fit is actually not that bet (as it's somehow capturing teh trend of the data). If if were better, it'd probably overfit which means that it will perform worse on new data (let's dave if the amount of levels is way higher than three).

2.3 Confidence Bands (non-parametric)

Task:

- Compute and plot the 95% confidence bands for the regression tree model from step 2 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.

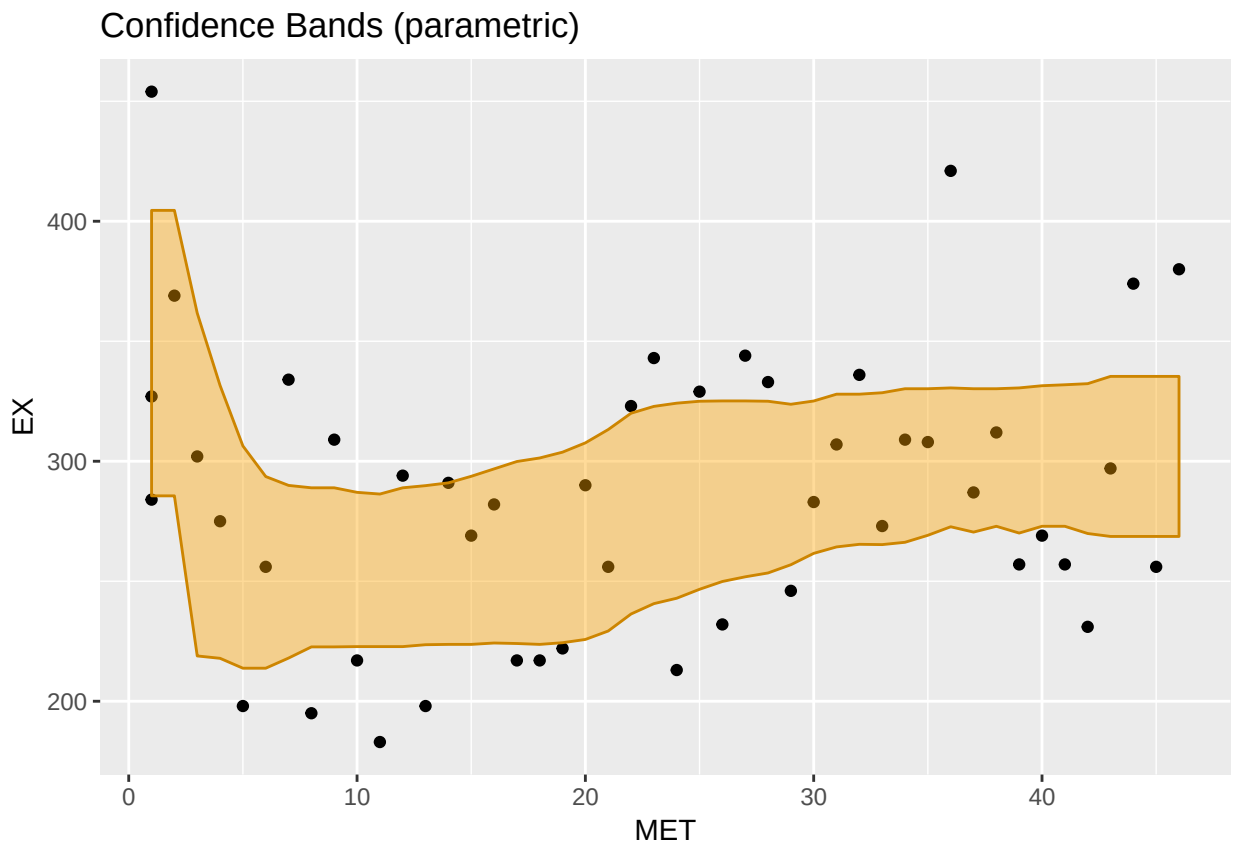


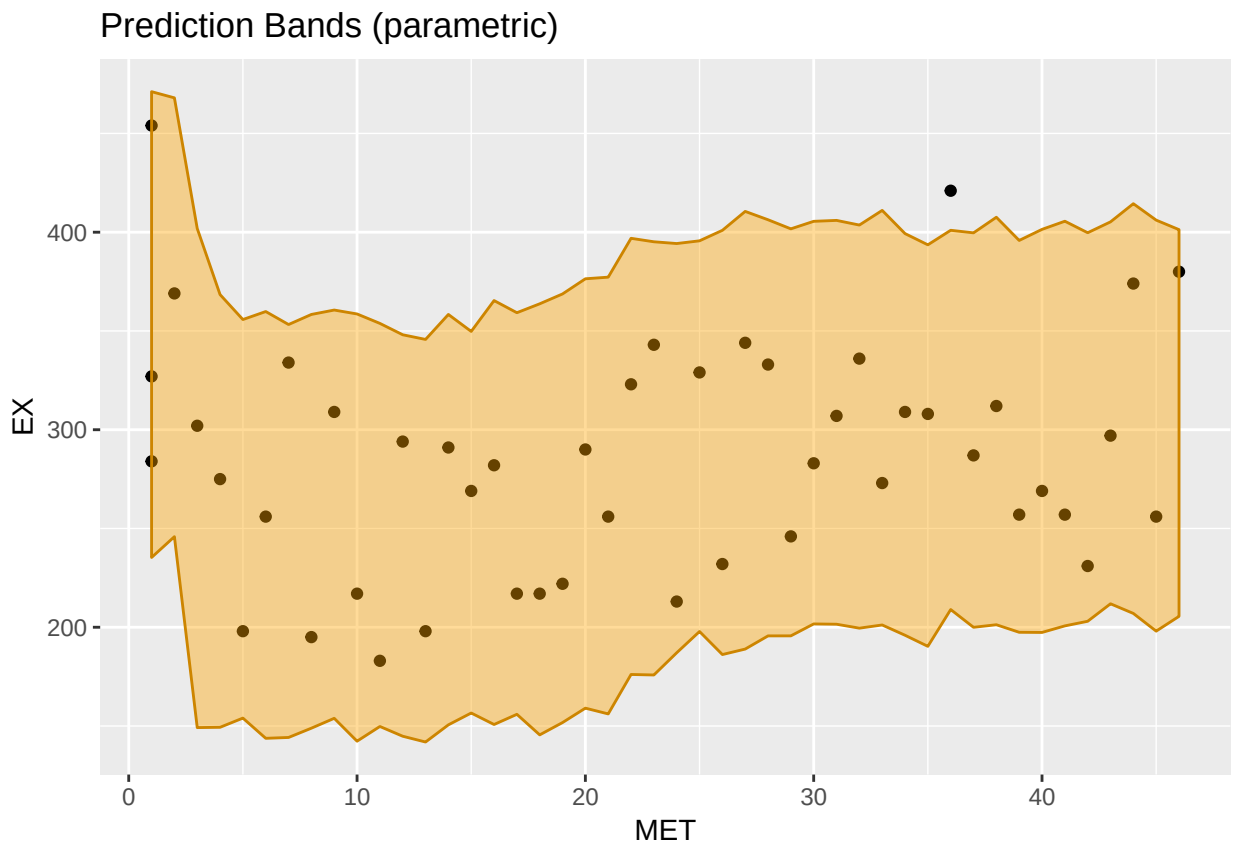
The confidence bands seem bumpy. This rises from the fact that our tree model has basically just three levels (compare residuals plot) and as we average this over all of the samples, we get this result. The results from our regression model from step 2 seem reliable as it's following the general trend of our data.

2.4 Confidence Bands (parametric)

Task:

- Compute and plot the 95% confidence and prediction bands the regression tree model from step 2 by using a parametric bootstrap.
- 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?


Confidence Bands (parametric)
EX
400
300
200
0
10
20
30
40
MET



Comment: The confidence intervall is shallow. It is showing us, for which original data points we have a confidence of 95% that we classified is correctly, which makes sense, as we used the tree generated previously as a statistic. For the prediction band we can see that basically just one data point is not inside of the band. We have therefor 1 out of 48 datapoints outside of our prediction band, which is around 2.1%. I'd have expected it to be two or three, but I think one is still feasible.

2.5 Conclusions

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

Answer: It makes more sense to use a parametric bootstrap if the distribution is known. Looking at the histrogram we cannot really estimate from which kind of distributions this one is coming from as we have only few data points. Thus a non-parametric bootstrap would make more sense here. If we had more data points, this estimation could change.

3 Assignemnt 4: Principal Components

3.1 Principal Component Analysis

Task:

- 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?

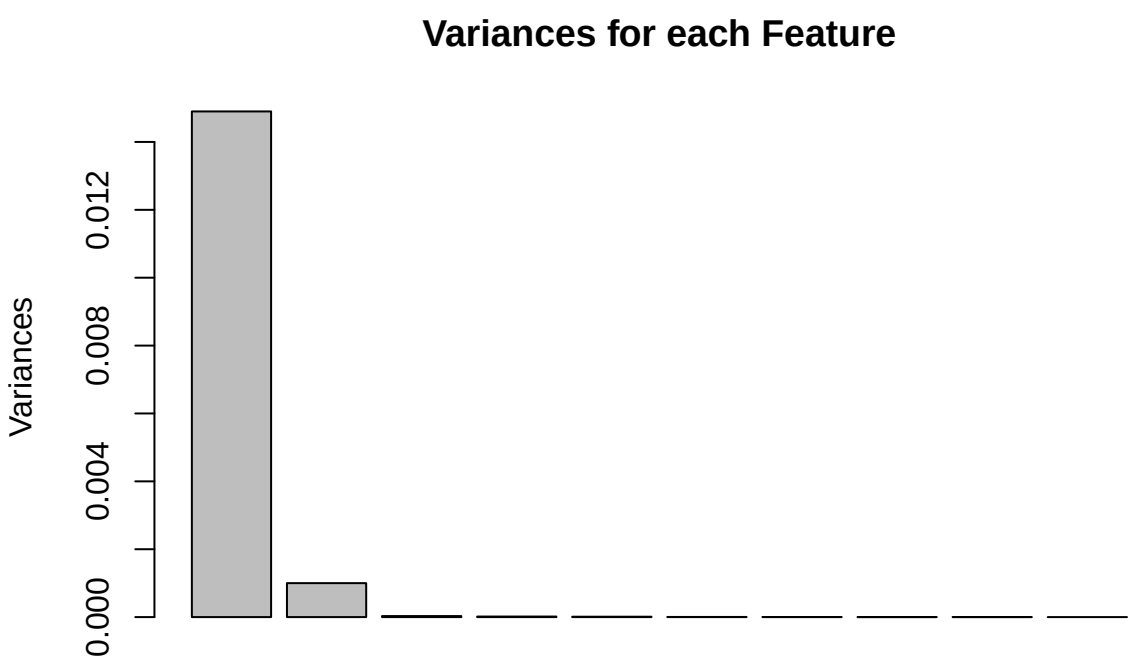
Let's import the data and have a look at it.

Table 12: NIRspectra.csv

	EX	ECAB	MET	GROW	YOUNG	OLD	WEST	STATE
3	327	87	1	3,7	28,5	11,2	0	VT
41	284	93,9	1	13,3	30,7	8,7	1	ID
42	454	125,8	1	13,7	29,1	7,8	1	WY
30	369	93,4	2	2,9	30,2	9,3	1	ND
31	302	88,2	3	4,6	28,9	10,5	1	SD
2	275	94,3	4	14,7	26,4	11,2	0	NH

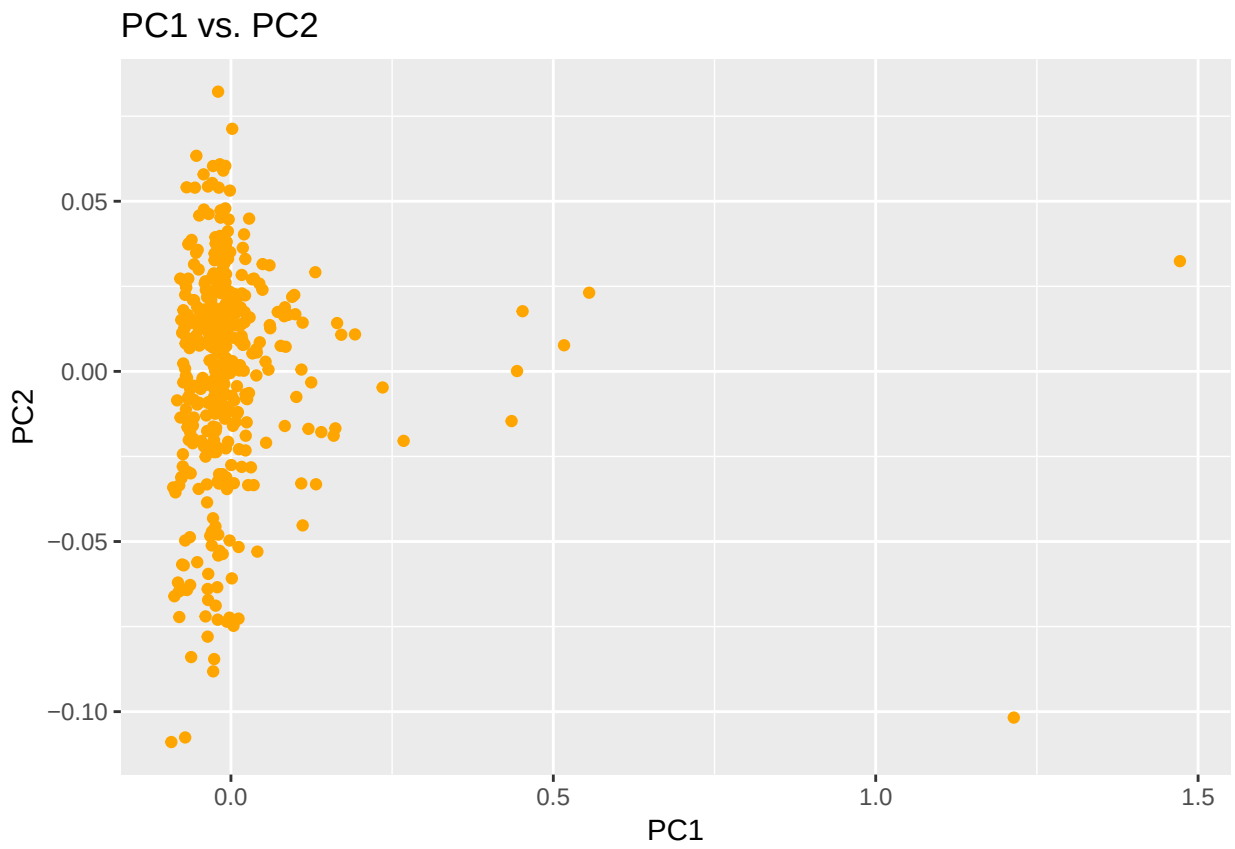
Table 13: Variance for each Feature

x
93.332
6.263
0.185
0.101
0.068
0.025



As we can see from the plot and the table, the first feature is responsible for `93.332%` of the variance and the second feature for `6.262%`. Together those two make of of `99.594%` of the variance.

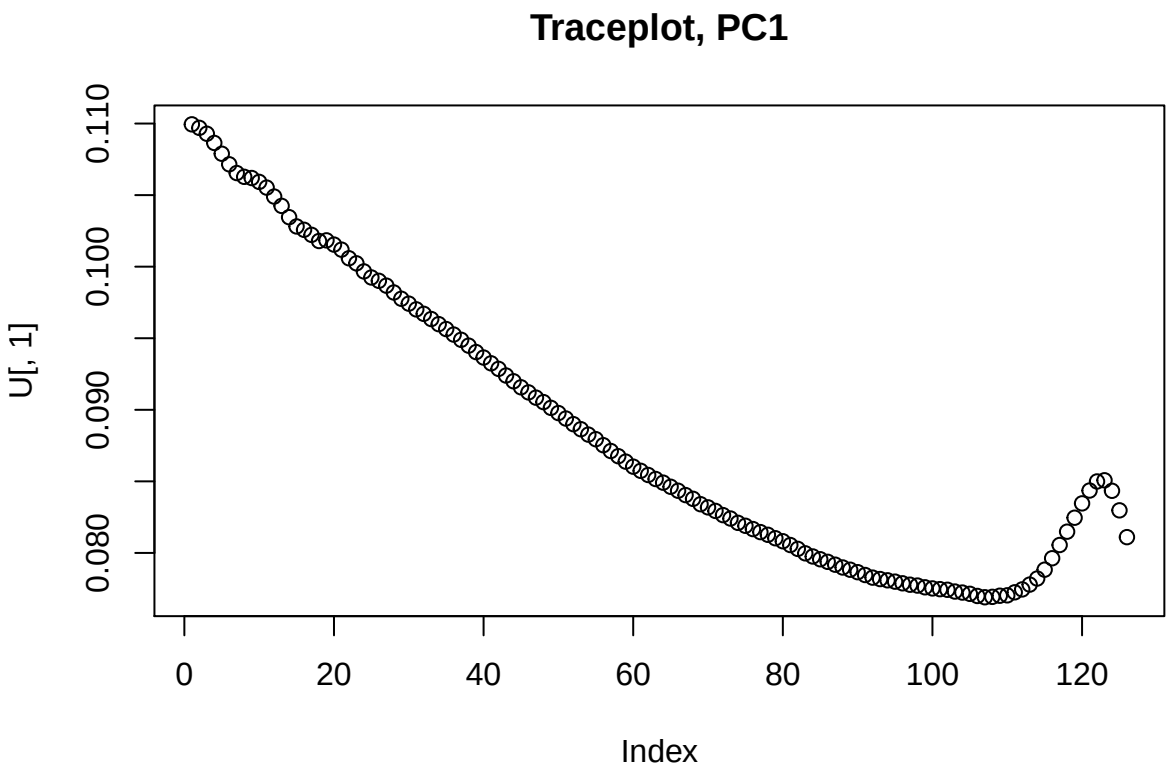
Let's have a look at the plot for PC1 and PC2.

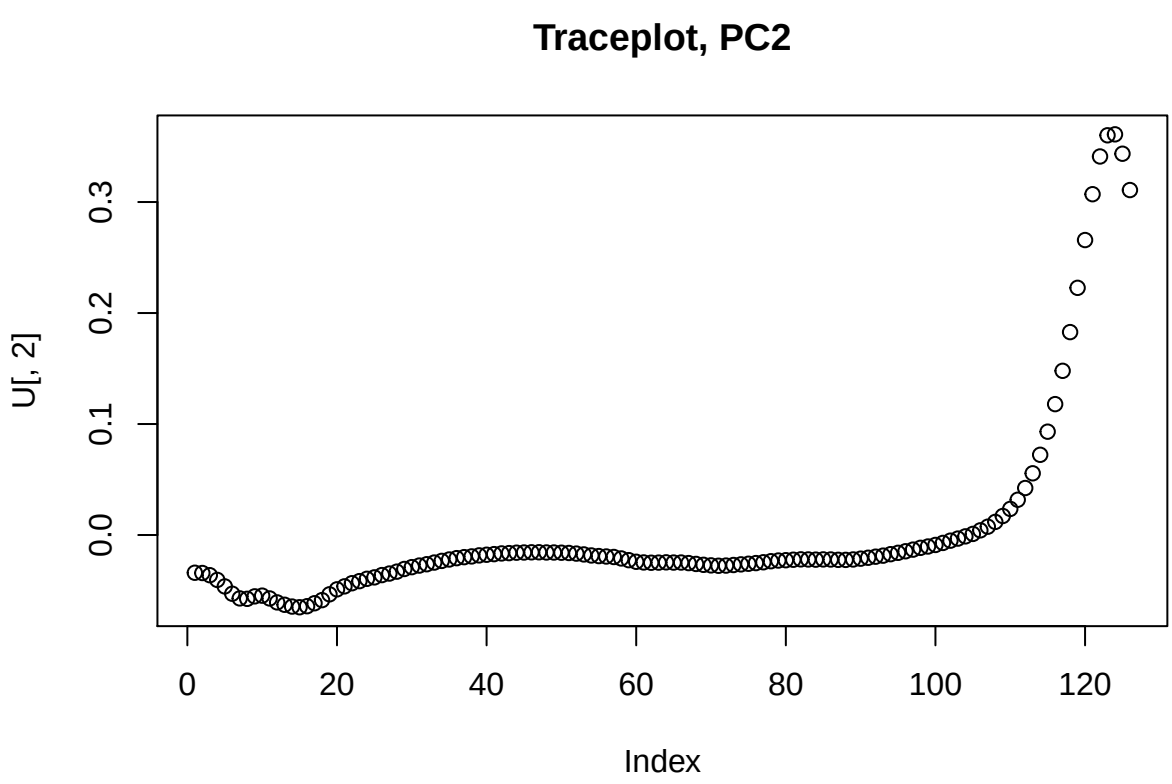


We can see that there are some datapoints which are far away from the others, but just two of them are really far away, their PC1 values are greater than `1.0`, even nearly around `1.5` for one.

3.2 Trace Plots

Task: 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?

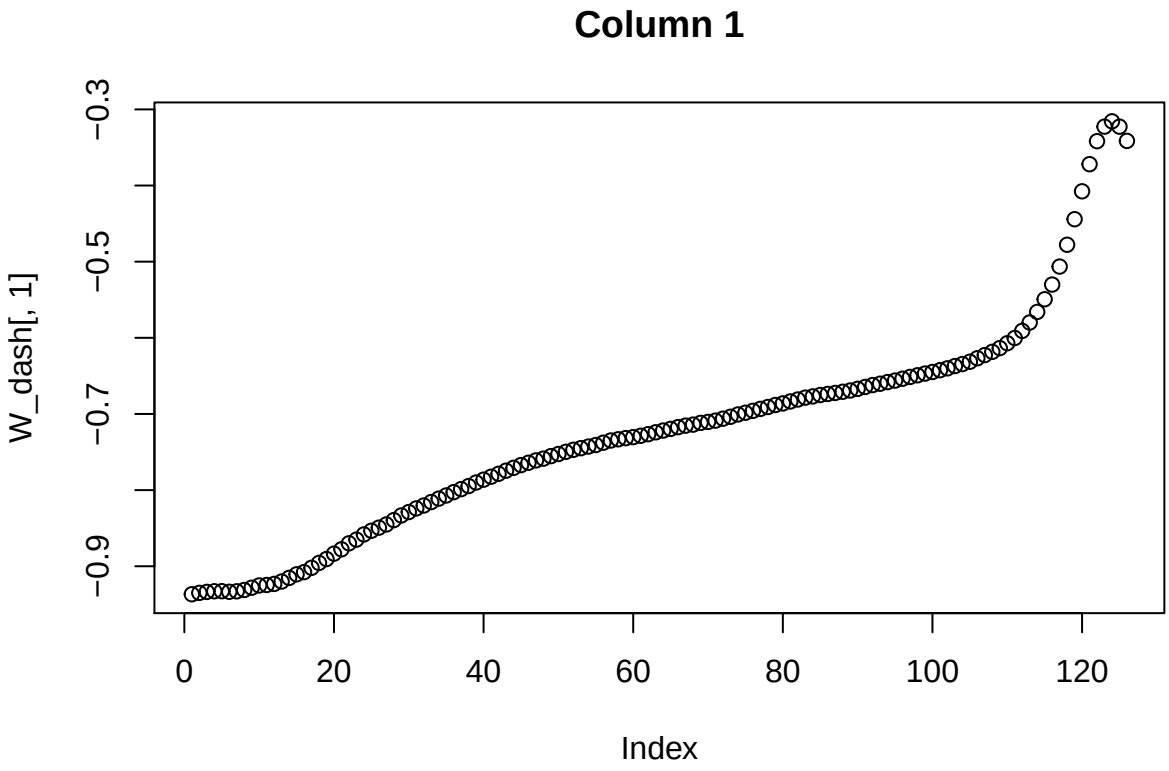

Traceplot, PC1
U[, 1]
0.080
0.090
0.100
0.110
0
20
40
60
80
100
120
Index

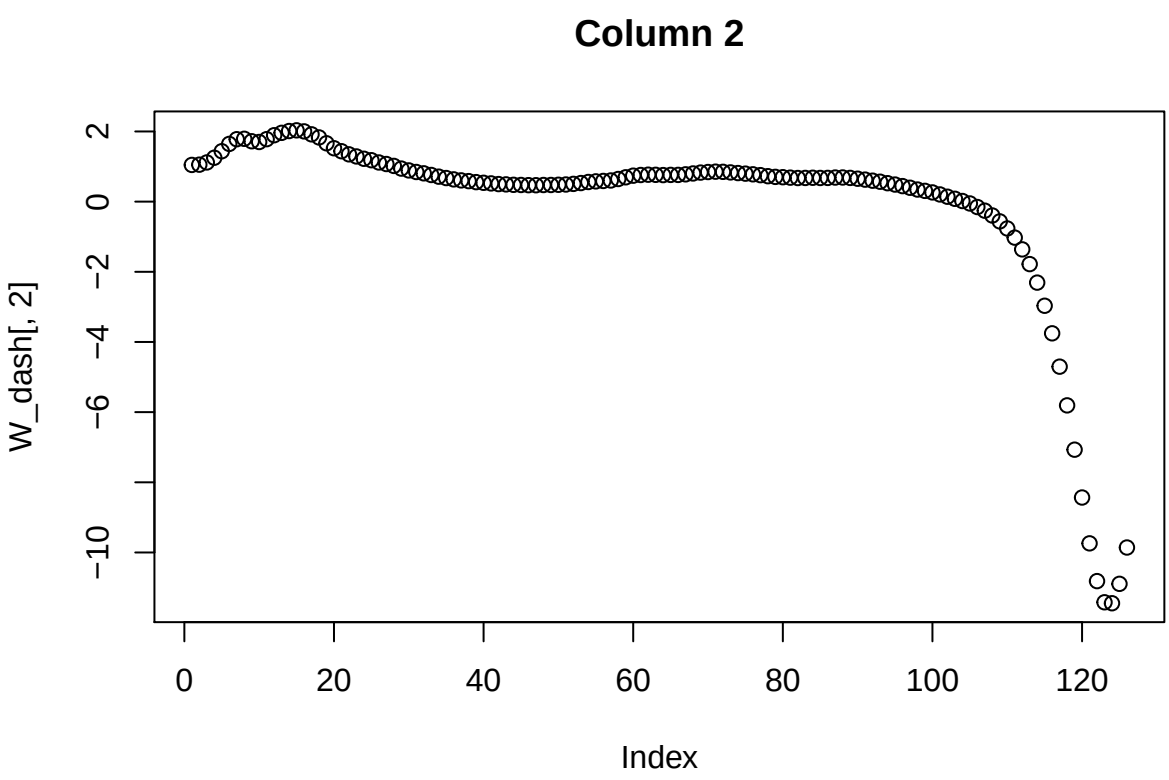


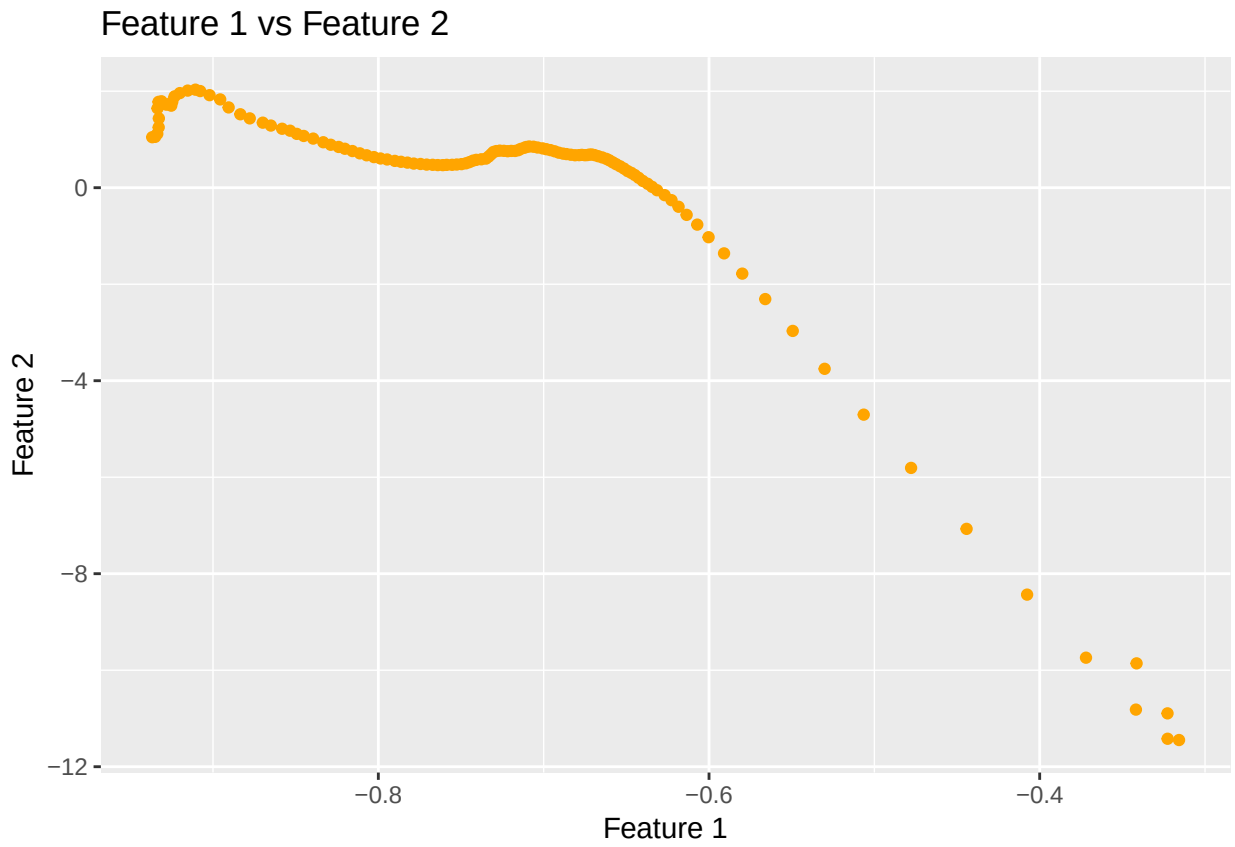
The latter one seems to be explained by fewer features as more coordinates are around zero.

3.3 ICA

- Perform Independent Component Analysis with the number of components selected in step 1.
- Compute W' = K * W
- 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'?
- Make a plot of the scores of the first two latent features and compare it with the score plot from step 1.


Column 1
W_dash[, 1]
−0.3
−0.5
−0.7
−0.9
0
20
40
60
80
100
120
Index


Column 2
W_dash[, 2]
2
0
−2
−4
−6
−10
0
20
40
60
80
100
120
Index



Conclusion: W' is the direct projection from the feature space into new space found by ICA. K is the pre-whitning matrix and W is our estimated un-mixing matrix. We can see for component two, that a similar basis is found, but inverted.

4 Appendix: Source Code

```
knitr::opts_chunk$set(echo = TRUE)
library(knitr)
library(ggplot2)
library(readxl)
library(tree)
library(e1071)
library(boot)
library(fastICA)

################################################################################
# Assignment 2: Analysis of Credit Scoring
################################################################################

set.seed(12345)
creditscoring = read_excel("./creditscoring.xls")
creditscoring$good_bad = as.factor(creditscoring$good_bad)
kable(head(creditscoring[,(ncol(creditscoring)-10):ncol(creditscoring)]),
      caption = "creditscoring.xls")
```

```r
n=dim(creditscoring)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=creditscoring[id,]

id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.25))

valid=creditscoring[id2,]
id3=setdiff(id1,id2)
test=creditscoring[id3,]


# Create the models
decisionTree_deviance = tree(good_bad ~ ., data = train, split = "deviance")
decisionTree_gini = tree(good_bad ~ ., data = train, split = "gini")


# Prediction
prediction_deviance_train =
  predict(decisionTree_deviance, newdata = train, type = "class")
prediction_deviance_test =
  predict(decisionTree_deviance, newdata = test, type = "class")

predictiona_gini_train =
  predict(decisionTree_gini, newdata = train, type = "class")
prediction_gini_test =
  predict(decisionTree_gini, newdata = test, type = "class")


summary(decisionTree_deviance)
#plot(decisionTree_deviance)


confusion_matrix_deviance = table(prediction_deviance_test, test$good_bad)
kable(confusion_matrix_deviance)


error_rate_deviance =
  1 - sum(diag(confusion_matrix_deviance)/sum(confusion_matrix_deviance))
print(error_rate_deviance)


summary(decisionTree_gini)
#plot(decisionTree_gini)


confusion_matrix_gini = table(prediction_gini_test, test$good_bad)
kable(confusion_matrix_gini)
```

```r
error_rate_gini =
  1 - sum(diag(confusion_matrix_gini)/sum(confusion_matrix_gini))
print(error_rate_gini)


# Taken from the slides
trainScore = rep(0, 15)
testScore = rep(0, 15)

for(i in 2:15) {
  prunedTree = prune.tree(decisionTree_deviance, best = i)
  pred = predict(prunedTree, newdata = valid, type = "tree")
  trainScore[i] = deviance(prunedTree)
  testScore[i] = deviance(pred)
}


## Add one as we trim the first index
optimalTreeIdx = which.min(testScore[-1]) + 1
optimalTreeScore = min(testScore[-1])

print(optimalTreeIdx)
print(optimalTreeScore)


plot(2:15, trainScore[2:15], type = "b", col = "orange", ylim = c(250,650),
     main = "Tree Depth vs Training/Test Score", ylab = "Deviance",
     xlab = "Number of Leaves")
points(2:15, testScore[2:15], type = "b", col = "blue")
legend("topright", legend = c("Train (orange)", "Test (blue)"))


optimalTree = prune.tree(decisionTree_deviance, best = optimalTreeIdx)
plot(optimalTree)
text(optimalTree, pretty = 1)
title("Optimal Tree")


prediction_optimalTree_test =
  predict(optimalTree, newdata = test, type = "class")

confusion_matrix_optimalTree = table(prediction_optimalTree_test, test$good_bad)

error_optimalTree =
  1 - sum(diag(confusion_matrix_optimalTree)/sum(confusion_matrix_optimalTree))


summary(optimalTree)
kable(confusion_matrix_gini)
print(error_optimalTree)


naiveBayesModel = naiveBayes(good_bad ~ ., data = train)
```

```r
summary(naiveBayesModel)


# Prediction
prediction_bayes_train =
  predict(naiveBayesModel, newdata = train, type = "class")
prediction_bayes_test =
  predict(naiveBayesModel, newdata = test, type = "class")

confusion_matrix_bayes_train = table(prediction_bayes_train, train$good_bad)
confusion_matrix_bayes_test = table(prediction_bayes_test, test$good_bad)

error_bayes_train = 1 - sum(diag(confusion_matrix_bayes_train)/
                            sum(confusion_matrix_bayes_train))
error_bayes_test =  1 - sum(diag(confusion_matrix_bayes_test)/
                            sum(confusion_matrix_bayes_test))


kable(confusion_matrix_bayes_train)
print(error_bayes_train)


kable(confusion_matrix_bayes_test)
print(error_bayes_test)


# prediction optimal tree
prediction_optimalTree_test_p =
  predict(optimalTree, newdata = test, type = "vector")
# prediction naive bayes
prediction_bayes_test_p =
  predict(naiveBayesModel, newdata = test, type = "raw")

pi = seq(from = 0.00, to = 1.0, by = 0.05)
fprs_tree = c()
tprs_tree = c()
fprs_bayes = c()
tprs_bayes = c()

for (i in pi) {
  current_tree_pi_confusion =
    table(test$good_bad, factor(prediction_optimalTree_test_p[,2] > i,
                                lev=c(TRUE, FALSE)))
  current_bayes_pi_confusion =
    table(test$good_bad, factor(prediction_bayes_test_p[,2] > i,
                                lev=c(TRUE, FALSE)))

  # FPR = FP / N-
  # TPR = TP / N+
  fprs_tree = c(fprs_tree, current_tree_pi_confusion[1,1]/
                  sum(current_tree_pi_confusion[1,]))
  tprs_tree = c(tprs_tree, current_tree_pi_confusion[2,1]/
                   sum(current_tree_pi_confusion[2,]))
```

```
  fprs_bayes = c(fprs_bayes, current_bayes_pi_confusion[1,1]/
                   sum(current_bayes_pi_confusion[1,]))
  tprs_bayes = c(tprs_bayes, current_bayes_pi_confusion[2,1]/
                   sum(current_bayes_pi_confusion[2,]))
}

roc_values = data.frame(fprs_tree, tprs_tree, fprs_bayes, tprs_bayes)


kable(roc_values)


ggplot(roc_values) +
  geom_line(aes(x = fprs_tree, y = tprs_tree,
                colour = "ROC Optimized Tree")) +
  geom_point(aes(x = fprs_tree, y = tprs_tree), colour = "orange") +

  geom_line(aes(x = fprs_bayes, y = tprs_bayes,
                colour = "ROC Naive Bayes")) +
  geom_point(aes(x = fprs_bayes, y = tprs_bayes), colour = "blue") +

  geom_abline(slope=1, intercept=0, linetype="dotted") +

  labs(title = "ROC for Optimized Tree and Naive Bayes", y = "TPR",
       x = "FPR", color = "Legend") +
  scale_color_manual(values = c("blue", "orange"))


L = matrix(c(0, 10, 1, 0), nrow = 2)
colnames(L) = c("Predicted", "Predicted")
rownames(L) = c("good", "bad")
kable(L)


# Prediction
prediction_bayes_train_raw =
  predict(naiveBayesModel, newdata = train, type = "raw")
prediction_bayes_test_raw =
  predict(naiveBayesModel, newdata = test, type = "raw")

confusion_matrix_bayes_train =
  table(prediction_bayes_train_raw[,2]/
          prediction_bayes_train_raw[,1] > 10, train$good_bad)
confusion_matrix_bayes_test =
  table(prediction_bayes_test_raw[,2]/
          prediction_bayes_test_raw[,1] > 10, test$good_bad)

error_bayes_train_raw = 1 - sum(diag(confusion_matrix_bayes_train)/
                                  sum(confusion_matrix_bayes_train))
error_bayes_test_raw =  1 - sum(diag(confusion_matrix_bayes_test)/
                                  sum(confusion_matrix_bayes_test))

kable(confusion_matrix_bayes_train)
```

```r
print(error_bayes_train_raw)


kable(confusion_matrix_bayes_test)
print(error_bayes_test_raw)


################################################################################
# Assignment 3: Uncertainty Estimation
################################################################################

statedata = read.csv("./State.csv", sep = ";")
kable(head(statedata),  caption = "State.csv")


statedata$MET = as.numeric(statedata$MET)
statedata = statedata[order(statedata$MET),]
ggplot(statedata, aes(x = MET, y = EX)) +  geom_point() + geom_smooth()


# Create the model
reg_tree = tree(EX ~ MET, data = statedata, control =
                  tree.control(nobs = nrow(statedata), minsize = 8))

# Use cross validation
cross_val_reg_tree = cv.tree(reg_tree)

# Plot the deviance of the sizes
plot(cross_val_reg_tree)


# Let's create the pruned the with best set to 3 and get it's prediction
pruned_tree = prune.tree(reg_tree, best = 3)
pruned_tree_prediction = predict(pruned_tree, newdata = statedata, type = "vector")

# We create a data.frame to save our values to make it easier to plot the data
pruned_tree_plot_dataframe =
  data.frame(statedata$MET, statedata$EX, pruned_tree_prediction,
             pruned_tree_prediction-statedata$EX)
names(pruned_tree_plot_dataframe) = c("met", "orignal_ex", "predicted_ex", "residual")

# Lets first plot the pruned tree
plot(pruned_tree)
text(pruned_tree, pretty = 1)
title("Optimal Tree with best = 3")

# Lets create a plot with the real and predictes values and highlight the
# residuals
ggplot(pruned_tree_plot_dataframe) +
  geom_point(aes(x = pruned_tree_plot_dataframe$met,
                 y = pruned_tree_plot_dataframe$orignal_ex),
             color = "black") +
  geom_point(aes(x = pruned_tree_plot_dataframe$met,
```

```
                y = pruned_tree_plot_dataframe$predicted_ex),
            color = "darkblue") +
  geom_segment(mapping=aes(x=pruned_tree_plot_dataframe$met,
                           y=pruned_tree_plot_dataframe$orignal_ex,
                           xend=pruned_tree_plot_dataframe$met,
                           yend=pruned_tree_plot_dataframe$predicted_ex),
               color = "red", linetype = "dotted") +
  labs(title = "Original Data, Fitted Data and Residuals", y = "EX",
       x = "MET", color = "Legend")


hist(pruned_tree_plot_dataframe$residual,
     col="orange", main = "Histogram of the Residuals", xlab = "Residuals", breaks = 20)


# We take the function given from the slides and adjust to the tree
# computing bootstrap samples
f_non_p_bootstrap = function(data, ind) {

  # First take the subsample
  data1 = data[ind,]

  # Now create a tree with the same hyperparameters from that subsample
  tree_model = tree(EX ~ MET, data = data1,
                    control = tree.control(nrow(data),minsize = 8))
  tree_model_pruned = prune.tree(tree_model, best = 3)

  # Use that model to predict on the real data
  prediction = predict(tree_model_pruned, newdata = data)
  return(prediction)
}

# Lets create the Bootstrap (again taken from slides)
res = boot(statedata, f_non_p_bootstrap, R = 1000)

# Confidence Bands using envelope
ci_non_p_bootstrap = envelope(res)
ci_non_p_bootstrap_df = as.data.frame(t(ci_non_p_bootstrap$point))
names(ci_non_p_bootstrap_df) = c("upper_bound", "lower_bound")
pruned_tree_plot_dataframe =
  data.frame(pruned_tree_plot_dataframe, ci_non_p_bootstrap_df)

# Plot the data
ggplot(pruned_tree_plot_dataframe) +
  geom_point(aes(x = pruned_tree_plot_dataframe$met,
                 y = pruned_tree_plot_dataframe$orignal_ex),
             color = "black") +
  geom_ribbon(aes(x = pruned_tree_plot_dataframe$met,
                  ymin = ci_non_p_bootstrap_df$lower_bound,
                  ymax = ci_non_p_bootstrap_df$upper_bound),
              alpha = 0.4, fill = "orange", color = "orange3") +
    labs(title = "Confidence Bands (non-parametric)", y = "EX",
       x = "MET", color = "Legend")
```

```r
# Again we take the sample from the slides and adjust it to our needs
# 1) Compute value mle
# 2) Write function ran.gen that depends on data and mle and which generates
# new data
# 3) Write function statistic that depend on data which will be generated by
# ran.gen and should return the estimator

## 1)
mle = pruned_tree

## 2)
rng = function(data, mle) {
  data1 = data.frame(EX=data$EX, MET=data$MET)
  n = length(data$EX)
  #generate new Price
  # summary needed to access the residuals
  data1$EX = rnorm(n, predict(mle, newdata=data1), sd(summary(mle)$residuals))
  return(data1)
}

## 3) f_non_p_bootstrap + distribution N
f_p_bootstrap = function(data) {

  # The index is not needed any more as we don't take a sub-sample

  # Now create a tree with the same hyperparameters from that subsample
  tree_model = tree(EX ~ MET, data = data,
                    control = tree.control(nrow(data), minsize = 8))
  tree_model_pruned = prune.tree(tree_model, best = 3)

  # Use that model to predict on the real data
  prediction = predict(tree_model_pruned, newdata = data)

  return(prediction)
}

# Bootstrap
res2 = boot(statedata,
            statistic = f_p_bootstrap, R=1000, mle=mle,
            ran.gen=rng, sim="parametric")

# Confidence Bands using envolope
ci_p_bootstrap = envelope(res2)
ci_p_bootstrap_df = as.data.frame(t(ci_p_bootstrap$point))
names(ci_p_bootstrap_df) = c("upper_bound", "lower_bound")
pruned_tree_plot_dataframe_p =
  data.frame(pruned_tree_plot_dataframe, ci_p_bootstrap_df)

# Plot the data
ggplot(pruned_tree_plot_dataframe_p) +
  geom_point(aes(x = pruned_tree_plot_dataframe_p$met,
                 y = pruned_tree_plot_dataframe_p$orignal_ex),
```

```
            color = "black") +
  geom_ribbon(aes(x = pruned_tree_plot_dataframe_p$met,
                  ymin = ci_p_bootstrap_df$lower_bound,
                  ymax = ci_p_bootstrap_df$upper_bound),
              alpha = 0.4, fill = "orange", color = "orange3") +
    labs(title = "Confidence Bands (parametric)", y = "EX",
       x = "MET", color = "Legend")


# Prediction Bands

# from slides
f_p_bootstrap_pb = function(data) {

  # The index is not needed any more as we don't take a sub-sample

  # Now create a tree with the same hyperparameters from that subsample
  tree_model = tree(EX ~ MET, data = data,
                    control = tree.control(nrow(data), minsize = 8))
  tree_model_pruned = prune.tree(tree_model, best = 3)

  # Use that model to predict on the real data
  prediction = predict(tree_model_pruned, newdata = data)

  # Add the rnrom to the prediction
  prediction_normal = rnorm(nrow(data), prediction, sd(summary(mle)$residual))

  return(prediction_normal)
}

# Bootstrap
res3 = boot(statedata, statistic = f_p_bootstrap_pb,
            R=1000, mle=mle, ran.gen=rng, sim="parametric")

# Confidence Bands using envolope
pb_p_bootstrap = envelope(res3)
pb_p_bootstrap_df = as.data.frame(t(pb_p_bootstrap$point))
names(pb_p_bootstrap_df) = c("upper_bound", "lower_bound")
pruned_tree_plot_dataframe_p_pb =
  data.frame(pruned_tree_plot_dataframe, pb_p_bootstrap_df)

# Plot the data
ggplot(pruned_tree_plot_dataframe_p_pb) +
  geom_point(aes(x = pruned_tree_plot_dataframe_p_pb$met,
                 y = pruned_tree_plot_dataframe_p_pb$orignal_ex),
             color = "black") +
  geom_ribbon(aes(x = pruned_tree_plot_dataframe_p_pb$met,
                  ymin = pb_p_bootstrap_df$lower_bound,
                  ymax = pb_p_bootstrap_df$upper_bound), alpha = 0.4,
              fill = "orange", color = "orange3") +
    labs(title = "Prediction Bands (parametric)", y = "EX",
       x = "MET", color = "Legend")
```

```
##############################################################################
# Assignment 4: Principal Components
##############################################################################

set.seed(12345)
nir_spectra = read.csv2("./NIRspectra.csv")
kable(head(statedata),  caption = "NIRspectra.csv")


# Copy to not modify the original dataset
nir_spectra_copy = nir_spectra
nir_spectra_copy$Viscosity = c()

# PCA
res = prcomp(nir_spectra_copy)

# Eigenvalues
lambda = res$sdev^2

# Proportion of variation
kable(head(sprintf("%2.3f",lambda/sum(lambda)*100)), caption = "Variance for each Feature")

# Plot
screeplot(res, main = "Variances for each Feature")


ggplot(as.data.frame(res$x)) +
  geom_point(aes(x = res$x[,1],
                 y = res$x[,2]),
             color = "orange") +
    labs(title = "PC1 vs. PC2", y = "PC2",
       x = "PC1", color = "Legend")


U = res$rotation
plot(U[,1], main = "Traceplot, PC1")
plot(U[,2], main = "Traceplot, PC2")


set.seed(12345)

ica_model = fastICA(X = nir_spectra_copy, n.comp = 2, alg.typ = "parallel",
                    fun = "logcosh", alpha = 1, method = "R", row.norm = FALSE,
                    maxit = 200, tol = 0.0001, verbose = FALSE)

W_dash = ica_model$K %*% ica_model$W

plot(W_dash[,1], main= "Column 1")
plot(W_dash[,2], main= "Column 2")

ggplot(as.data.frame(W_dash)) +
  geom_point(aes(x = W_dash[,1],
                 y = W_dash[,2]),
```

```
            color = "orange") +
  labs(title = "Feature 1 vs Feature 2", y = "Feature 2",
     x = "Feature 1", color = "Legend")
```