Machine Learning Lab 02

Maximilian Pfundstein (maxpf364)

2018-12-02

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
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

resident	property	age	other	housing	existcr	job	depends	telephon	foreign	good_bad
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] "duration" "history"  "marital"  "existcr"  "amount"   "purpose"
##  [7] "savings"  "resident" "age"      "other"
## Number of terminal nodes:  22
## Residual mean deviance:  0.7423 = 277.6 / 374
## Misclassification error rate: 0.1869 = 74 / 396
```

The confusion matrix looks as follows:

	bad	good
bad	49	56
good	43	152

Therefore the error rate is:

```
## [1] 0.33
```

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" "amount"   "marital"  "duration" "resident"
## [13] "job"      "installp" "purpose"  "employed" "housing"
## Number of terminal nodes:  53
## Residual mean deviance:  0.9468 = 324.7 / 343
## Misclassification error rate: 0.2247 = 89 / 396
```

The confusion matrix looks as follows:

	bad	good
bad	25	43
good	67	165

Therefore the error rate is:

```
## [1] 0.3666667
```

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

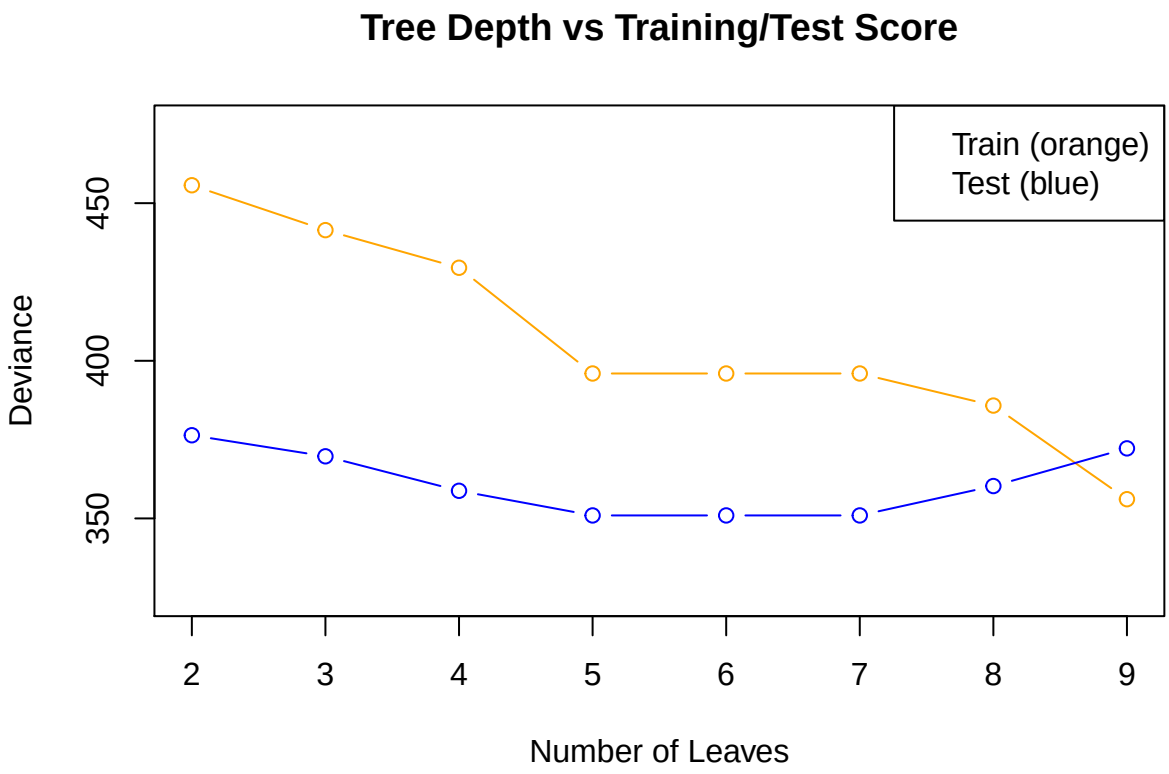
The best tree is the tree with index `5` and a test score of `350.952`.

```
## [1] 5
```

```
## [1] 350.952
```

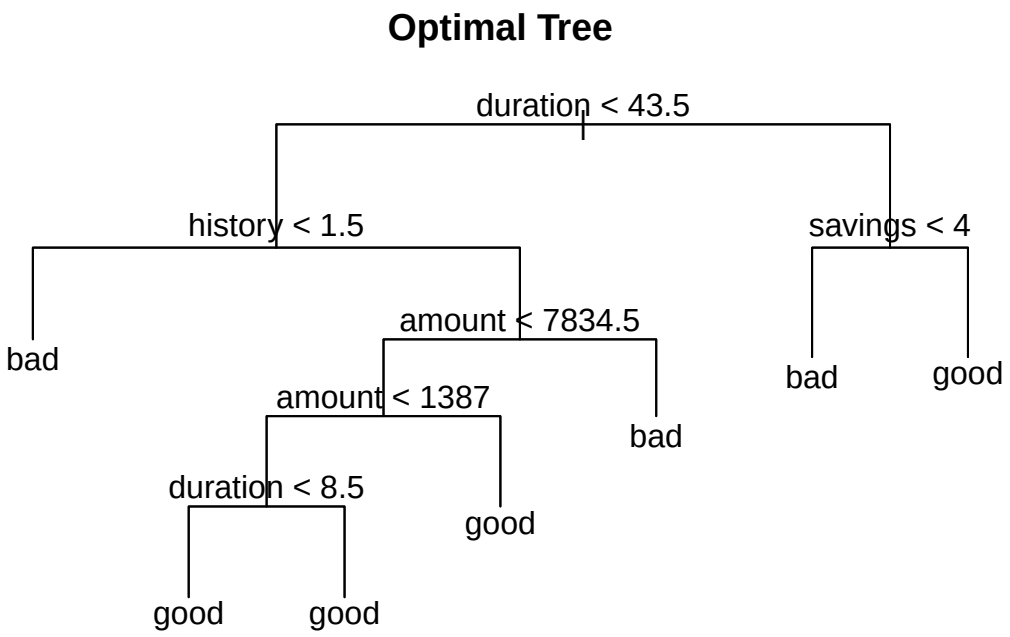
1.3.2 Dependency of Deviances

The following plots shows the Tree Depth vs the Training Score. The orange line indicates the training and the blue line the test score.



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 duration is smaller than `43.5`. This means this is the feature where the tree evaluated the most influence on the prediction. We can see that the right side only has one more variable which is savings. On the left side the tree splits further, starting with the history as the second most important feature after deciding the duration. As the tree has more leaves to teh left side we can see, that splitting this data further makes more than than it would've been on teh right sind.

1.3.5 Estimate of the Missclassification Rate

```
##
## Classification tree:
## snip.tree(tree = decisionTree_deviance, nodes = c(6L, 11L, 41L,
## 21L, 4L))
## Variables actually used in tree construction:
## [1] "duration" "history"  "amount"   "savings"
## Number of terminal nodes:  7
## Residual mean deviance:  1.018 = 396 / 389
## Misclassification error rate: 0.2323 = 92 / 396
```

	bad	good
bad	25	43
good	67	165

```
## [1] 0.2633333
```

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	62	55
good	52	231

```
## [1] 0.2675
```

Data for Naive Bayes on test:

	bad	good
bad	50	45
good	42	163

```
## [1] 0.29
```

1.4.3 Comparison with Step 3

We can see that the misqualification rate for the optimized decision tree with `0.2633333` is better than the Naive Bayes approach with a rate of `0.29`. 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

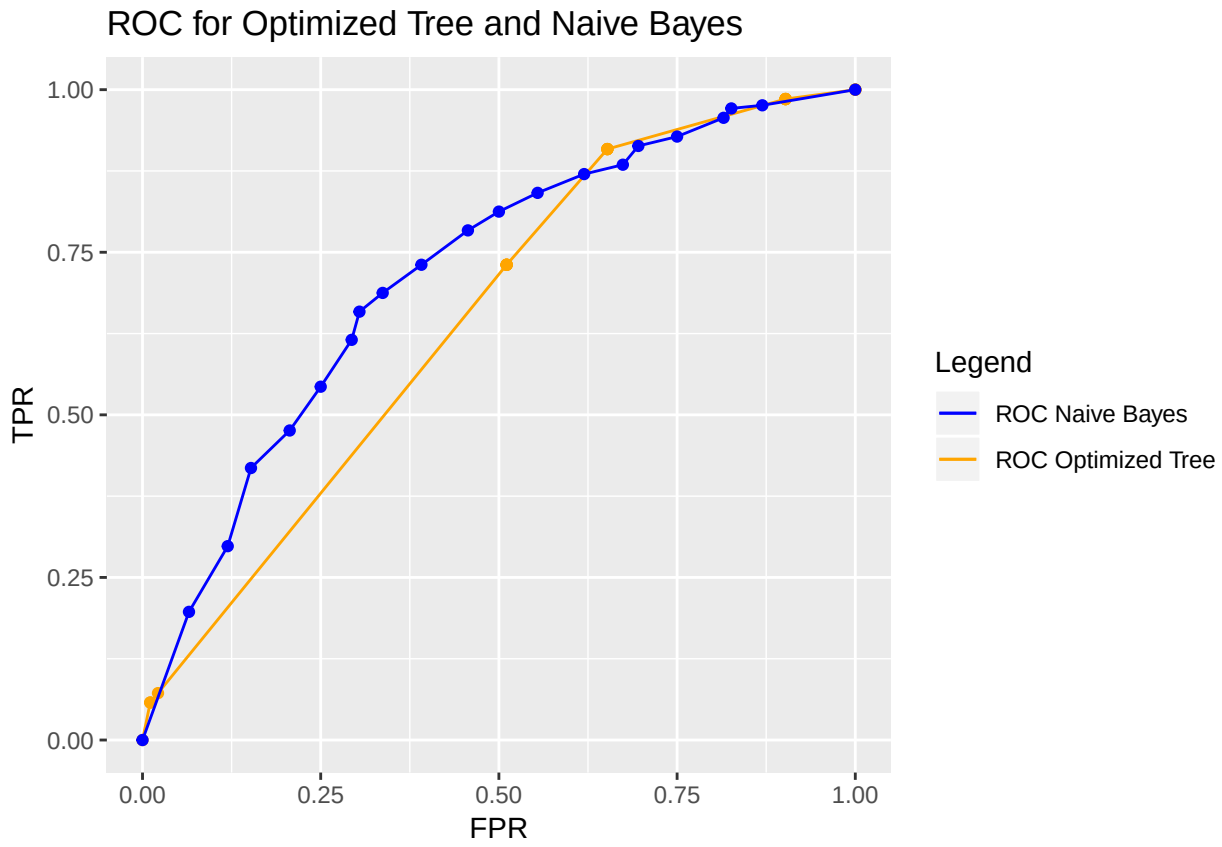
Task: Compute the TPR and FPR values for the two models.

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	0.8695652	0.9759615
1.0000000	1.0000000	0.8260870	0.9711538
1.0000000	1.0000000	0.8152174	0.9567308
0.9021739	0.9855769	0.7500000	0.9278846
0.9021739	0.9855769	0.6956522	0.9134615
0.9021739	0.9855769	0.6739130	0.8846154
0.9021739	0.9855769	0.6195652	0.8701923
0.9021739	0.9855769	0.5543478	0.8413462
0.6521739	0.9086538	0.5000000	0.8125000
0.6521739	0.9086538	0.4565217	0.7836538
0.6521739	0.9086538	0.3913043	0.7307692
0.6521739	0.9086538	0.3369565	0.6875000
0.5108696	0.7307692	0.3043478	0.6586538
0.5108696	0.7307692	0.2934783	0.6153846
0.5108696	0.7307692	0.2500000	0.5432692
0.5108696	0.7307692	0.2065217	0.4759615
0.0217391	0.0721154	0.1521739	0.4182692
0.0108696	0.0576923	0.1195652	0.2980769
0.0108696	0.0576923	0.0652174	0.1971154
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	93	272
TRUE	21	14

```
## [1] 0.7325
```

Confusion Matrix for Test:

	bad	good
FALSE	78	202
TRUE	14	6

```
## [1] 0.72
```

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. We have a really low false positive rate (with just 14 out of 286 being missclassified) in contrast to a high false negative rate (21 out of 35 being missclassified).

2 Assignment 3: Uncertainty Estimation

2.1 Import and Plot `State.csv`

Task:

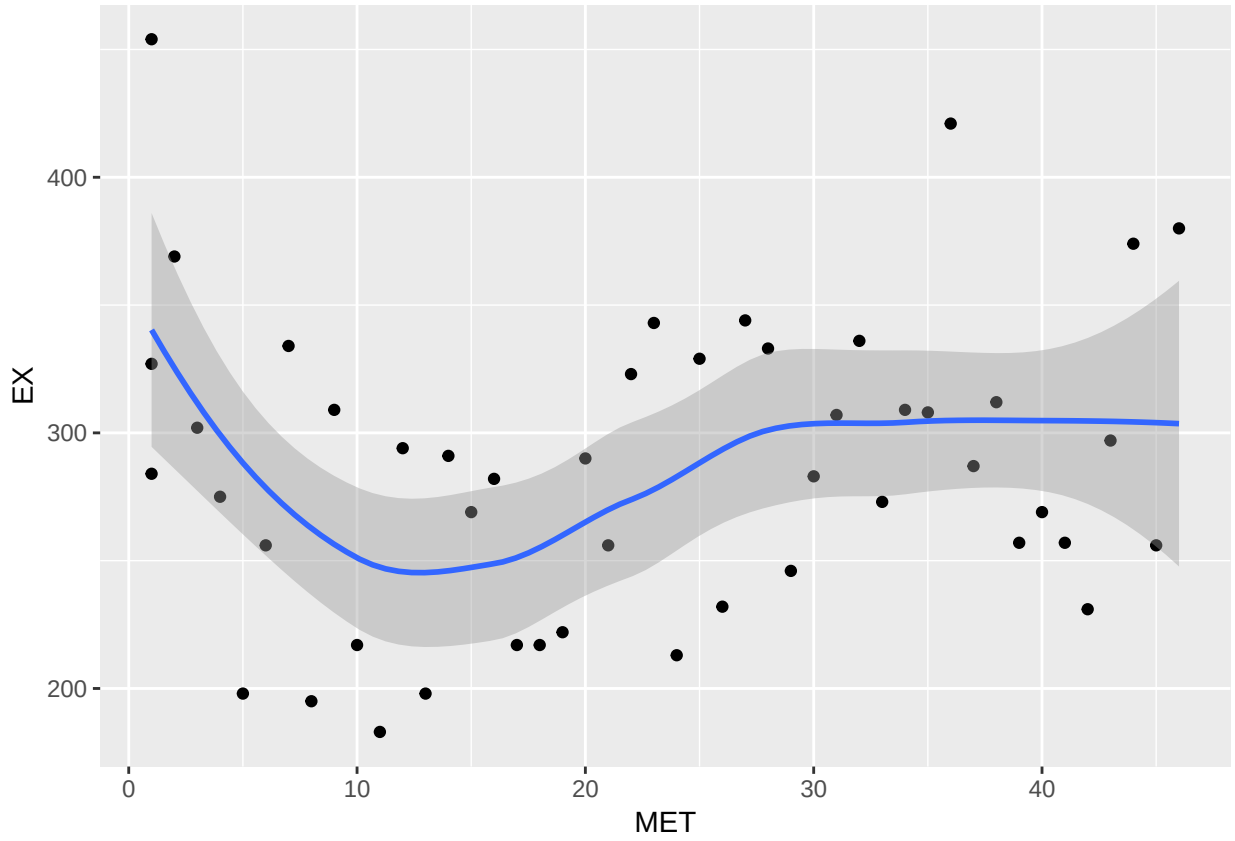
- Reorder your data with respect to the increase of MET and plot EX versus MET.
- Discuss what kind of model can be appropriate here.

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:



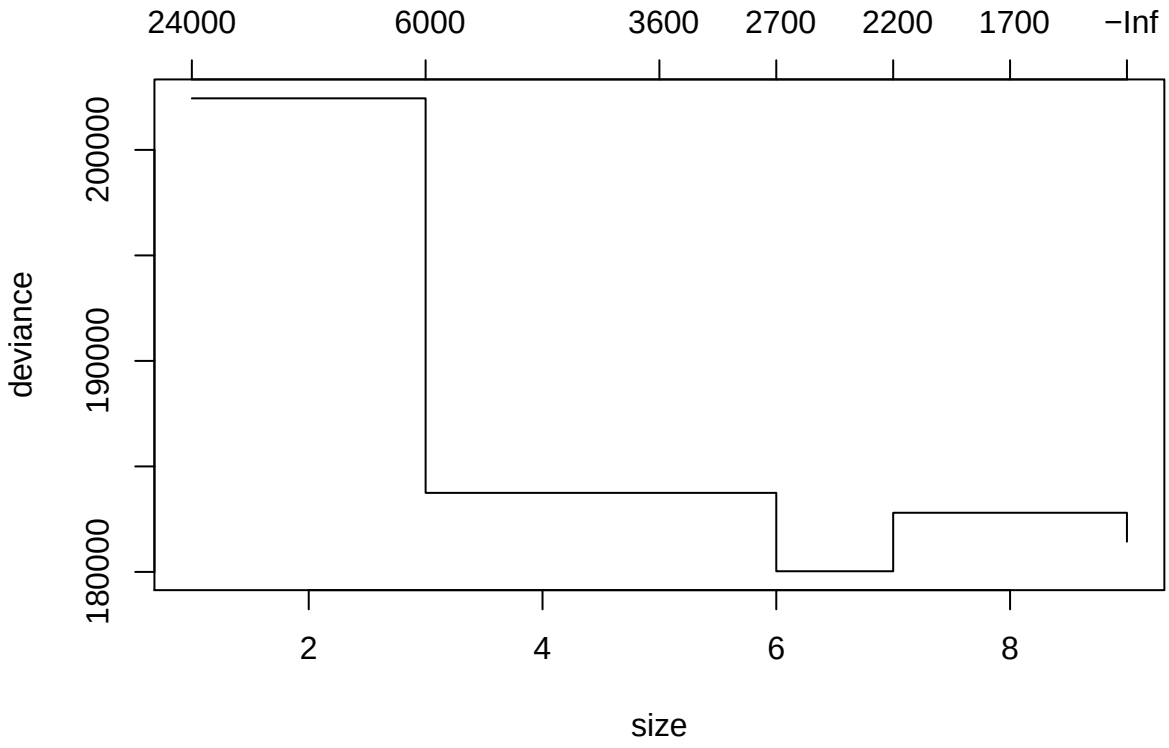
At a first glance this data looks messy, but taking a second look gives the impression that the data has different kind of levels. Therefor a Regression Tree would probably be a good model.

2.2 Regression Tree Model

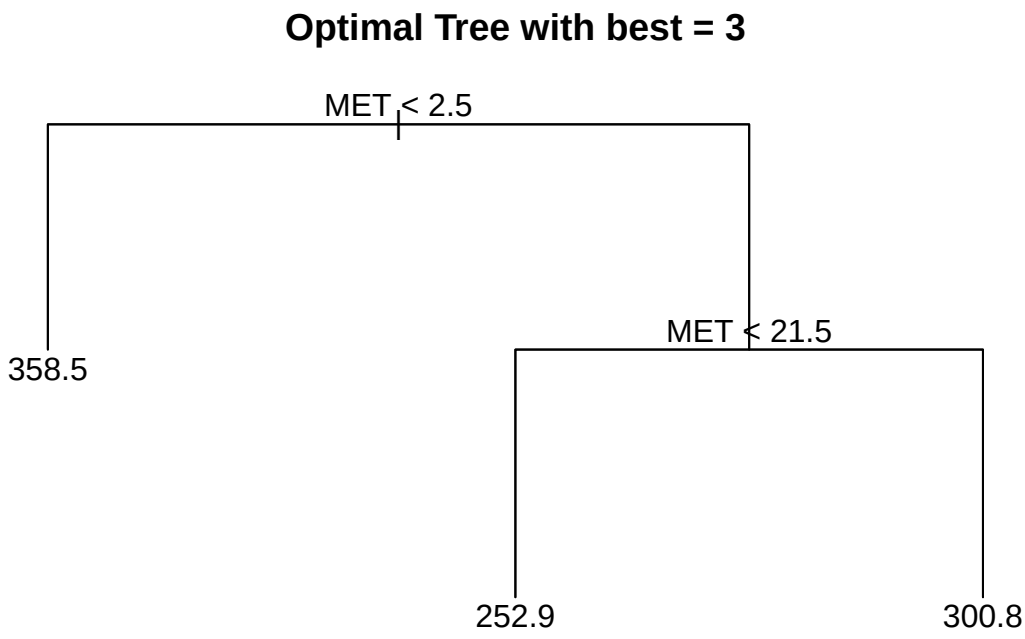
Task:

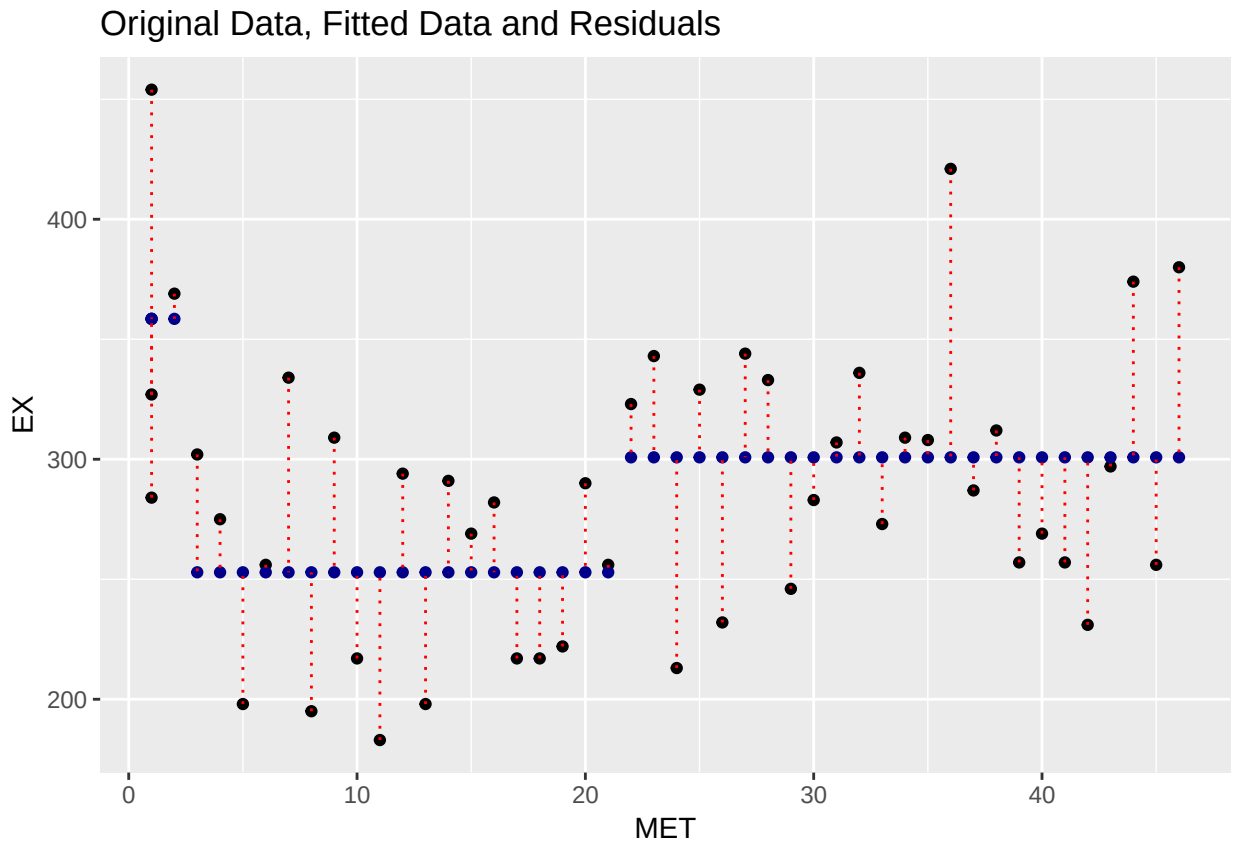
- Report the selected tree.
- Plot the original and the fitted data and histogram of residuals.
- Comment on the distribution of the residuals and the quality of the fit.

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

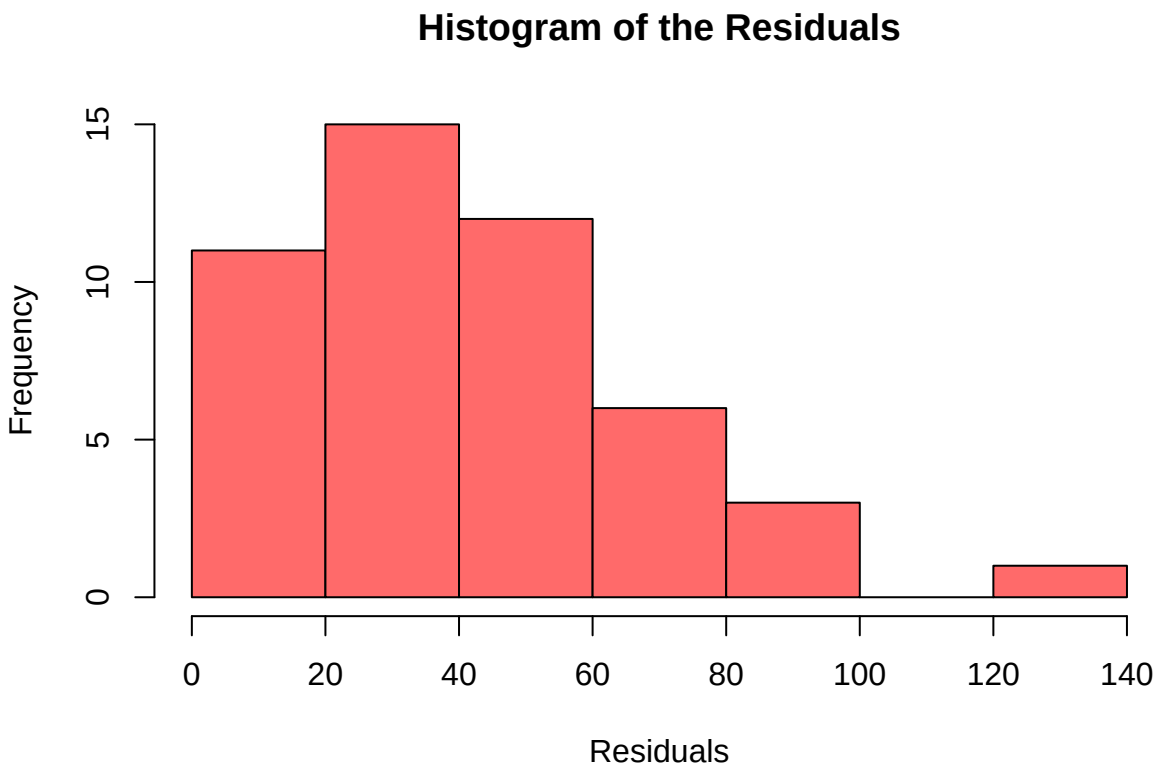


We see that 3 or 4 would work for the size of the tree. For the following we will declare `best = 3`. Let's prune our best tree and have a look at it.


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 residusls.

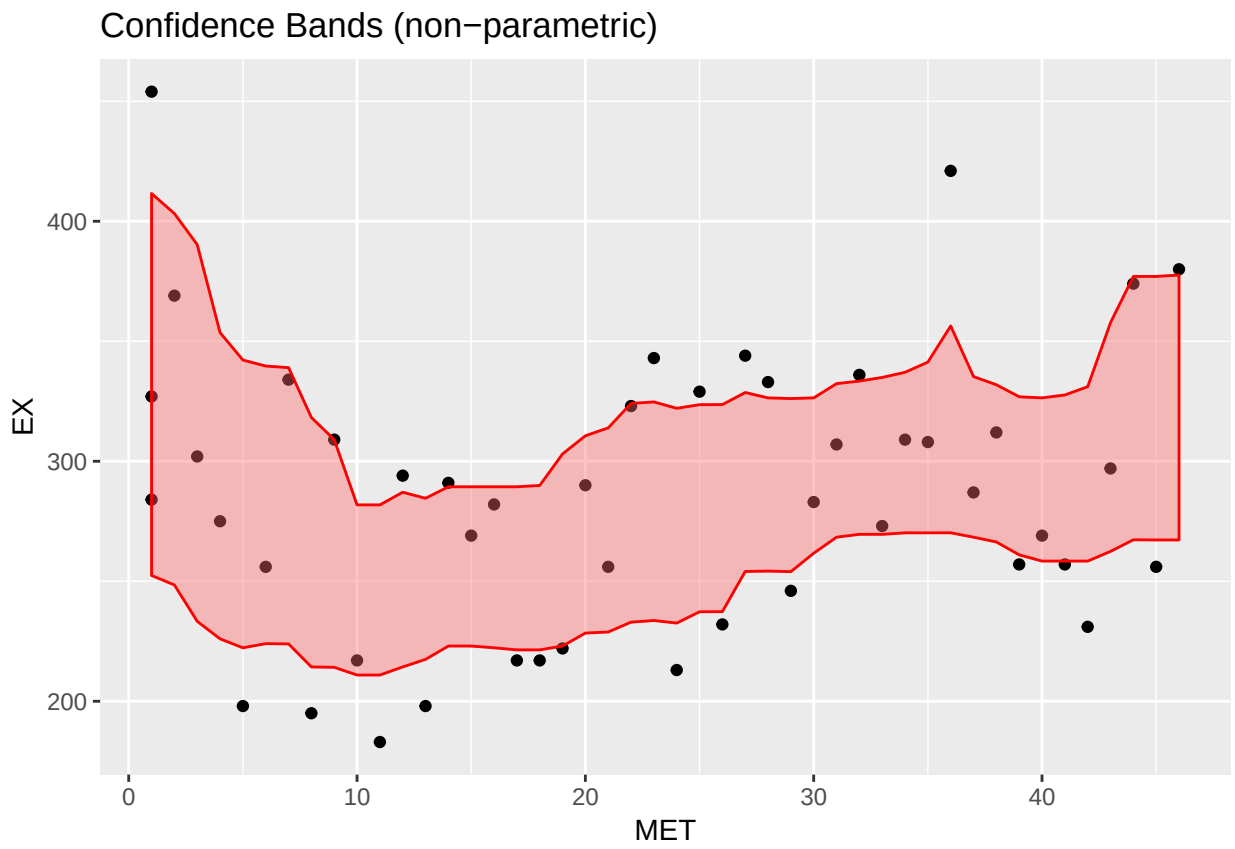


The histogramm looks like a Chi-Squared distribution with `k ~ 3`.

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.

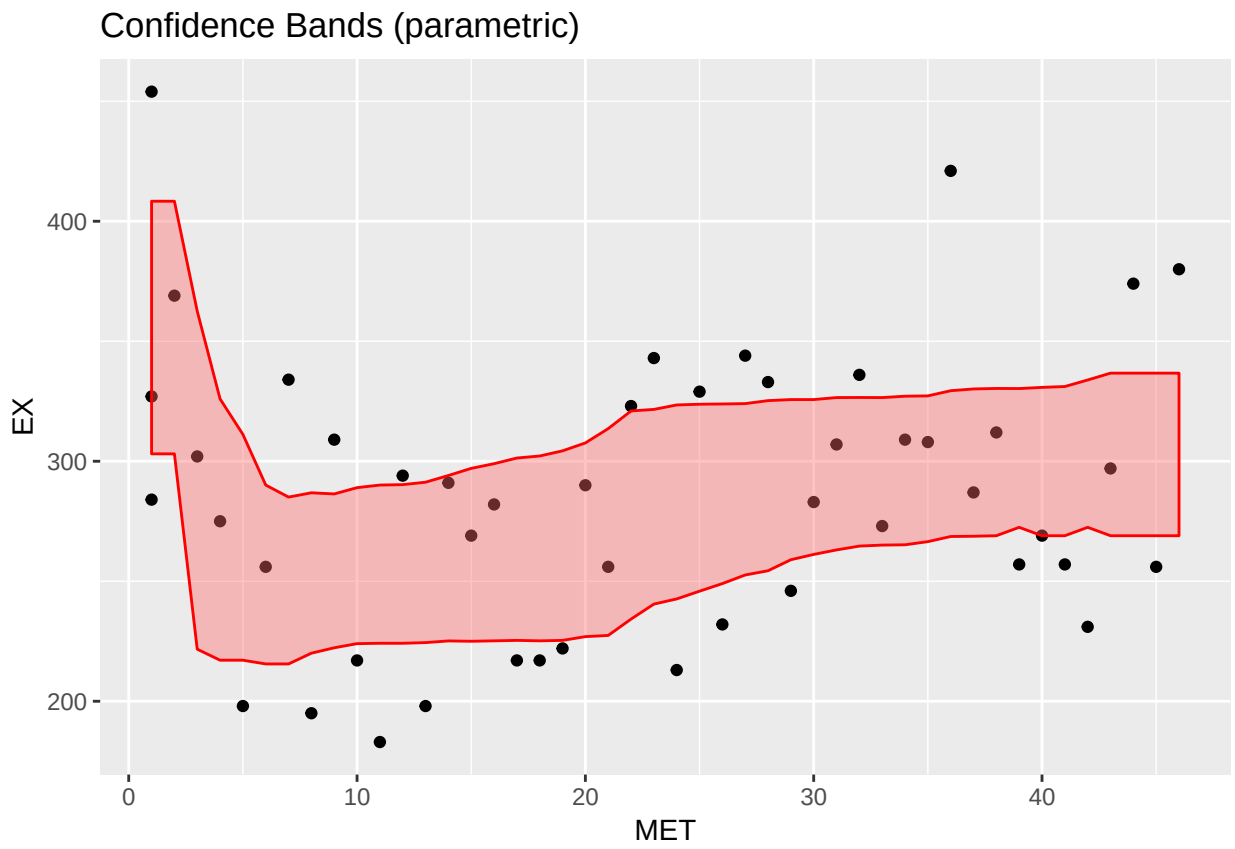


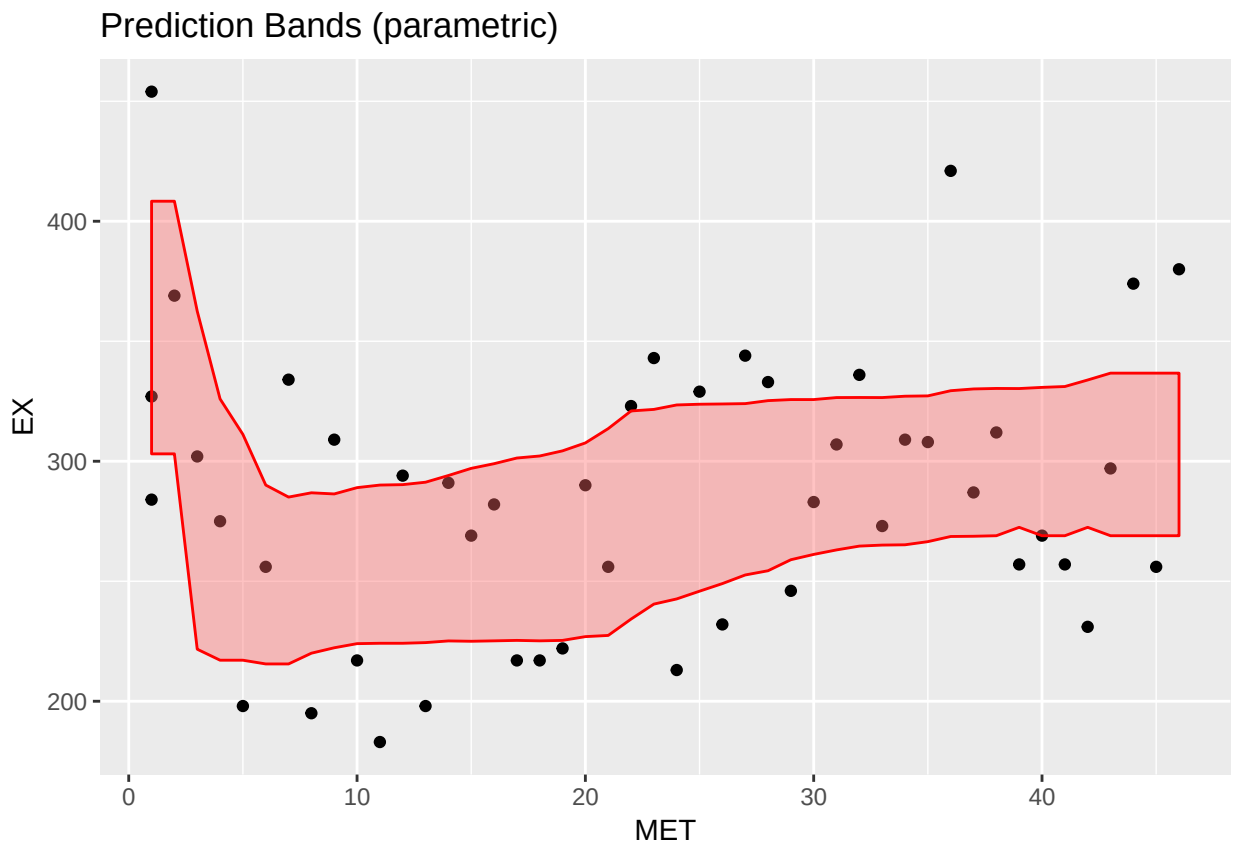
The prediction interval is neither really smooth or bumby, it's somwhere in between. **TODO**.

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: **TODO**

2.5 Conclusions

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

Answer: **TODO**

3 Assignemnt 4: Principal Components

```
set.seed(12345)
```

4 Appendix: Source Code

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

```
library(boot)

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")


n=dim(creditscoring)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.4))
train=creditscoring[id,]

id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.3))

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)
```

```r
summary(decisionTree_gini)
#plot(decisionTree_gini)


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


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


# Taken from the slides
trainScore = rep(0, 9)
testScore = rep(0, 9)

for(i in 2:9) {
  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 the trim the first index
optimalTreeIdx = which.min(testScore[-1]) + 1
optimalTreeScore = min(testScore[-1])

print(optimalTreeIdx)
print(optimalTreeScore)


plot(2:9, trainScore[2:9], type = "b", col = "orange", ylim = c(325,475),
     main = "Tree Depth vs Training/Test Score", ylab = "Deviance",
     xlab = "Number of Leaves")
points(2:9, testScore[2:9], 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))
```

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


naiveBayesModel = naiveBayes(good_bad ~ ., data = train)
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") +

  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[,1]/prediction_bayes_train_raw[,2] > 10
confusion_matrix_bayes_test = table(prediction_bayes_test_raw[,1]/prediction_bayes_test_raw[,2] > 10, t

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)
print(error_bayes_train_raw)


kable(confusion_matrix_bayes_test)
print(error_bayes_test_raw)


set.seed(12345)

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,
             abs(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="indianred1", main = "Histogram of the Residuals", xlab = "Residuals")


# 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 envolope
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 = "indianred1", color = "red") +
    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") +
```

```r
  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 = "indianred1", color = "red") +
    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(res2)
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 = "indianred1", color = "red") +
    labs(title = "Prediction Bands (parametric)", y = "EX",
       x = "MET", color = "Legend")
```

```r
set.seed(12345)
```