Machine Learning Lab 2

Theodor Emanuelsson (theem089), Lisa Goldschmidtböing (lisgo269), Christoforos Spyretos (chrsp415)

2021-12-02

Assignmnent 1: Explict Regularization

Reading & Splitting the data.

```
data <- read.csv("files/tecator.csv")

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

1.

Fitting the linear regression model.

The probabilistic model of linear regression is given by $y = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + ... + b_{100}x_{100} + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma^2)$ and $\sigma^2 \geq 0$.

```
train <- train[, -c(1, 103, 104)]
test <- test[, -c(1, 103, 104)]

lrm <- lm(Fat ~ ., data = train)
# summary(lrm)

train_error <- mean((train$Fat - predict(lrm))^2)
train_error
```

```
## [1] 0.005709117
```

```
test_error <- mean((test$Fat - predict(lrm, test))^2)
test_error
```

```
## [1] 722.4294
```

The model fits the training data well, but as the MSE of the test data is relatively high, it might have problems dealing with new data and is overfitting to the training data.

2.

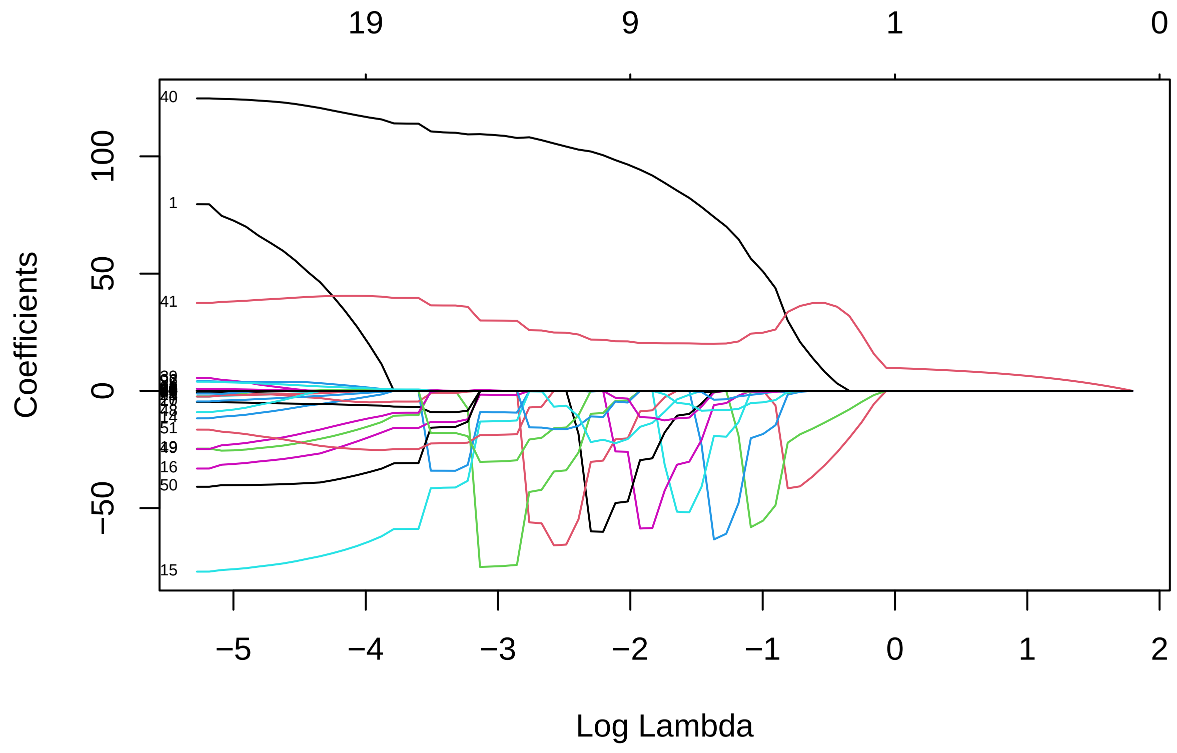
Report the cost function.

The cost function of the Lasso regression that it should be optimised is given by $\hat{\boldsymbol{\theta}} = \underset{\theta}{\operatorname{argmin}} \frac{1}{n}||\boldsymbol{X\theta} - \boldsymbol{y}||_2^2 + \lambda||\boldsymbol{\theta}||$

3.

Fitting the lasso regression model.

```r
library(glmnet)

X <- train[, -101]
Y <- data.frame(train[, 101])

lasso <- glmnet(as.matrix(X), as.matrix(Y), alpha = 1)

# summary(lasso) log(lasso$lambda)

plot(lasso, xvar = "lambda", label = TRUE)
```



```r
for (i in lasso$lambda) {
    nparam <- sum(coef(lasso, s = i)[, 1] != 0)
    if (nparam == 4) {
        lambda <- i
        break
    }
}
cat("Optimal lambda:", lambda)
```

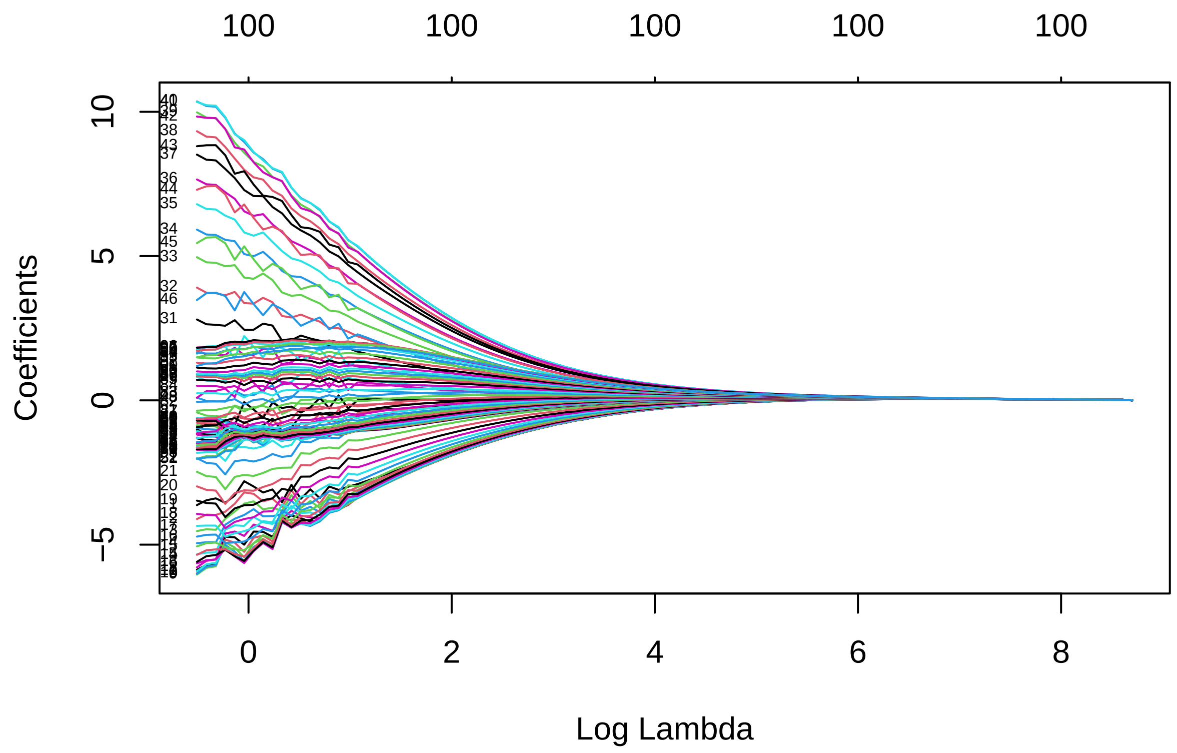
```
## Optimal lambda: 0.8530452
```

The numbers above the plot represent the number of features that have coefficients that are not zero; thus, the larger value of lambda has, the fewer features with an impact on the model has. In order to select a model with only three features, several lambdas could be chosen, and we choose the smallest of these options, which is equal to 0.8530452.

4.

Fitting the ridge regression model.

```
ridge <- glmnet(as.matrix(X), as.matrix(Y), alpha = 0)

plot(ridge, xvar = "lambda", label = TRUE)
```



In ridge regression, when the value of lambda is increased, the features are never typically 0. It is not possible to reduce the number of features with coefficients with impact. Thus, it is better to use the lasso regression to select a model with specific features or reduce the number of features.

5.

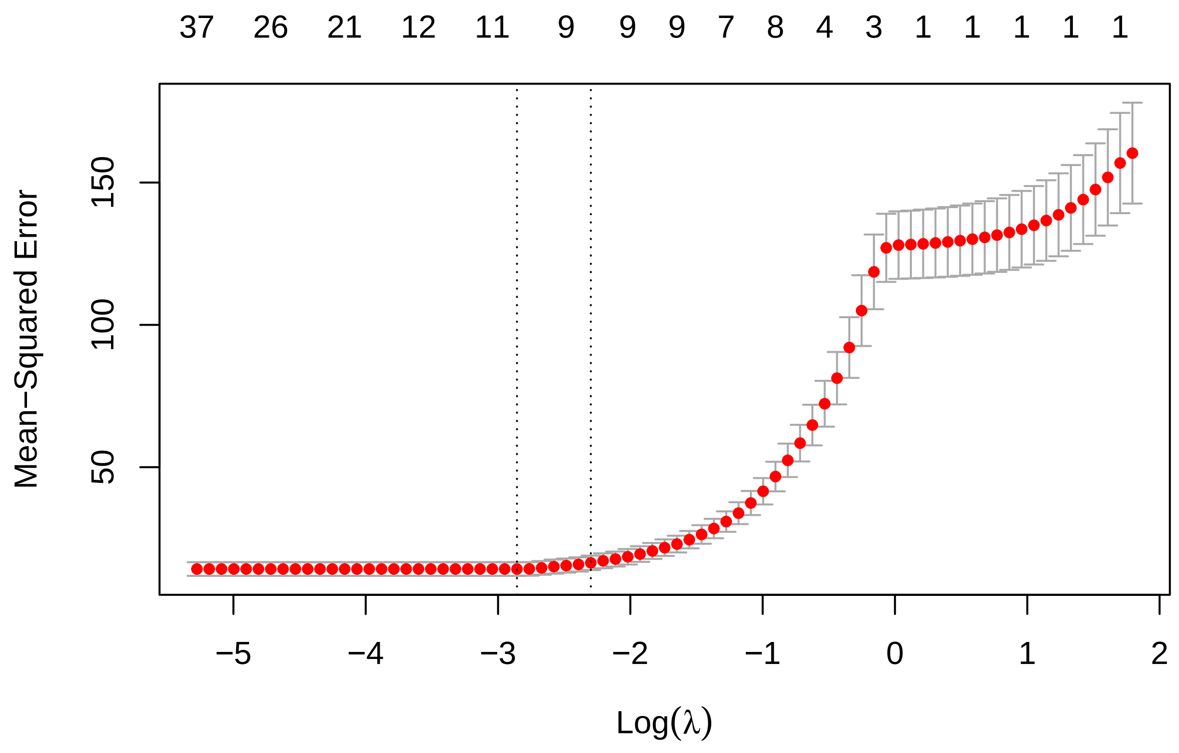
```r
cv_lasso <- cv.glmnet(as.matrix(X), as.matrix(Y), alpha = 1)
cat("Optimal lambda:", cv_lasso$lambda.min)
```

```
## Optimal lambda: 0.05744535
```

```r
sum(coef(cv_lasso, s = cv_lasso$lambda.min)[, 1] != 0) - 1  #substract intercept
```

```
## [1] 8
```

```r
plot(cv_lasso)
```

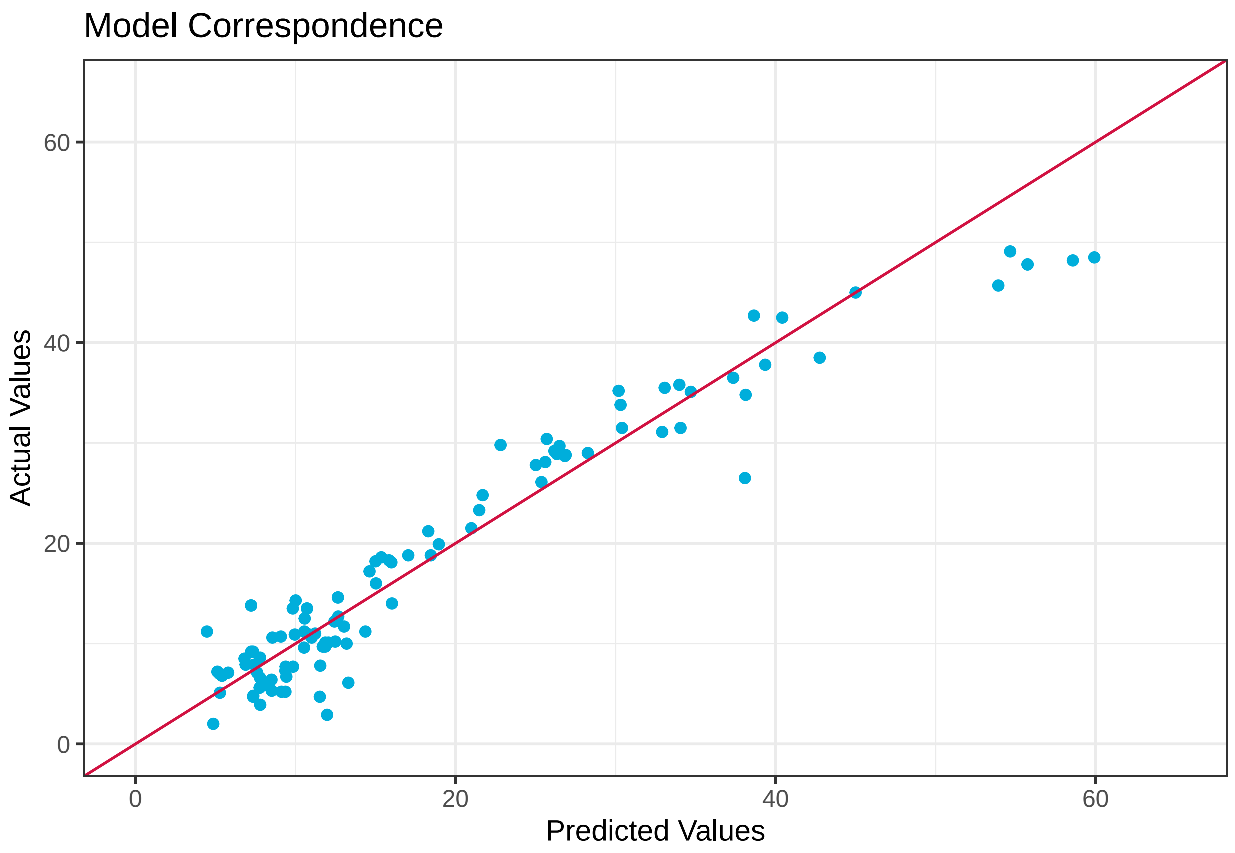


The confidence bounce is larger when the log(λ) is larger; thus, there is a confidence limit for each point, allowing how much better or worse each lambda is.

The optimal value for $log(\lambda)$ equals approximately -2.8, but the plot does not illustrate that this value is a statistical significant better prediction than the log(λ) $= -4$. This can be seen by the fact that the confidence intervals overlap.

Choosing this optimal value for $log(\lambda)$, we get eight variables and the intercept that impact the model.

```r
optimal.lambda <- predict(cv_lasso, as.matrix(test[, -101]),
    s = cv_lasso$lambda.min)

optimal_df <- data.frame(Actual = test$Fat, Predicted = optimal.lambda)

library(ggplot2)

my_scatterplot <- ggplot(optimal_df, aes(x = s1, y = Actual)) +
    geom_point(color = "#00aedb") + lims(x = c(0, 65), y = c(0,
    65)) + geom_abline(color = "#d11141") + labs(title = "Model Correspondence") +
    xlab("Predicted Values") + ylab("Actual Values") + theme_bw()

my_scatterplot
```



The plot illustrates that the model predictions corresponding to optimal lambda are relatively "good". Most observations are placed around the bisector, which is shown as the red line, but there is a prominent cluster of outliers when Fat is between 40 and 50, but predictions are between 50 and 60, which shows that for particular high values, the model is overestimating in all cases.

Assignment 2: Decision trees and logistic regression for bank marketing

1.

Import the data to R, remove variable "duration" and divide into training/validation/test as 40/30/30.

We start with reading in the data set. Additionally, we change all character variables to factor variables and split the dataset into training, validation and test dataset. We noticed that the dataset description and the real dataset are inconsistent. For example, the description says that we should have the weekday (Monday, Tuesday,...), but in our dataset, we have the day of the month (1,..31). We are going to work with the dataset we have but keep some possible interpretation issues in mind.

```r
bank <- read.csv2("files/bank-full.csv", header = TRUE)

# remove duration
df <- bank[, -12]

# changing target to factor
df$y <- factor(df$y, levels = c("yes", "no"))

# most other variables are supposed to be factor variables
# as well
for (i in c(2:5, 7:11, 15)) {
    df[, i] <- as.factor(df[, i])
}

## split like specified in the lecture slides
n = dim(df)[1]
set.seed(12345)
id = sample(1:n, floor(n * 0.4))
train = df[id, ]

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

id3 = setdiff(id1, id2)
test = df[id3, ]
```

2.

Fit decision trees to the training data so that you change the default settings one by one and report the misclassification rates for the training and validation data.

a.

Decision Tree with default settings

```r
# define misclassification function (return in percent)
misclass = function(X, X1) {
    return(round(1 - sum(diag(table(X, X1)))/length(X), 4))
}

tree1 <- tree(y ~ ., data = train)
summary(tree1)
```

```
##
## Classification tree:
## tree(formula = y ~ ., data = train)
## Variables actually used in tree construction:
## [1] "poutcome" "month"    "contact"  "day"
## Number of terminal nodes:  6
## Residual mean deviance:  0.5993 = 10830 / 18080
## Misclassification error rate: 0.1048 = 1896 / 18084
```

b.

Decision Tree with smallest allowed node size equal to 7000

```r
tree2 <- tree(y ~ ., data = train, control = tree.control(nobs = nrow(train),
    minsize = 7000))
summary(tree2)
```

```
##
## Classification tree:
## tree(formula = y ~ ., data = train, control = tree.control(nobs = nrow(train),
##     minsize = 7000))
## Variables actually used in tree construction:
## [1] "poutcome" "month"    "contact"
## Number of terminal nodes:  5
## Residual mean deviance:  0.6097 = 11020 / 18080
## Misclassification error rate: 0.1048 = 1896 / 18084
```

c.

Decision trees minimum deviance to 0.0005

```r
tree3 <- tree(y ~ ., data = train, control = tree.control(nobs = nrow(train),
    mindev = 5e-04))
summary(tree3)
```

```
##
## Classification tree:
## tree(formula = y ~ ., data = train, control = tree.control(nobs = nrow(train),
##     mindev = 5e-04))
## Variables actually used in tree construction:
##  [1] "poutcome"  "day"       "pdays"     "job"       "month"     "balance"
##  [7] "age"       "previous"  "marital"   "campaign"  "contact"   "housing"
## [13] "education" "loan"
## Number of terminal nodes:  150
## Residual mean deviance:  0.4991 = 8951 / 17930
## Misclassification error rate: 0.0913 = 1651 / 18084
```

```
train1 <- predict(tree1, type = "class")
train2 <- predict(tree2, type = "class")
train3 <- predict(tree3, type = "class")

valid1 <- predict(tree1, newdata = valid, type = "class")
valid2 <- predict(tree2, newdata = valid, type = "class")
valid3 <- predict(tree3, newdata = valid, type = "class")

errors_df <- data.frame(`Training Error` = c(misclass(train$y,
    train1), misclass(train$y, train2), misclass(train$y, train3)),
    `Validation Error` = c(misclass(valid$y, valid1), misclass(valid$y,
        valid2), misclass(valid$y, valid3)))

row.names(errors_df) <- c("Tree A", "Tree B", "Tree C")
errors_df
```

```
##        Training.Error Validation.Error
## Tree A         0.1048           0.1093
## Tree B         0.1048           0.1093
## Tree C         0.0912           0.1149
```

Which model is the best one among these three?

If one looks at the misclassification rates of the training data set, the last tree performs best. Nevertheless, taking the misclassification rates of the validation dataset into account, the last tree performs worst. This could show us that the last tree might overfit a bit as it is better on the training dataset. There is no difference visible between the first two trees. The second tree has one splitless; we would prefer that one because it is simpler and still achieves the same results.

Report how changing the deviance and node size affected the size of the trees and explain why.

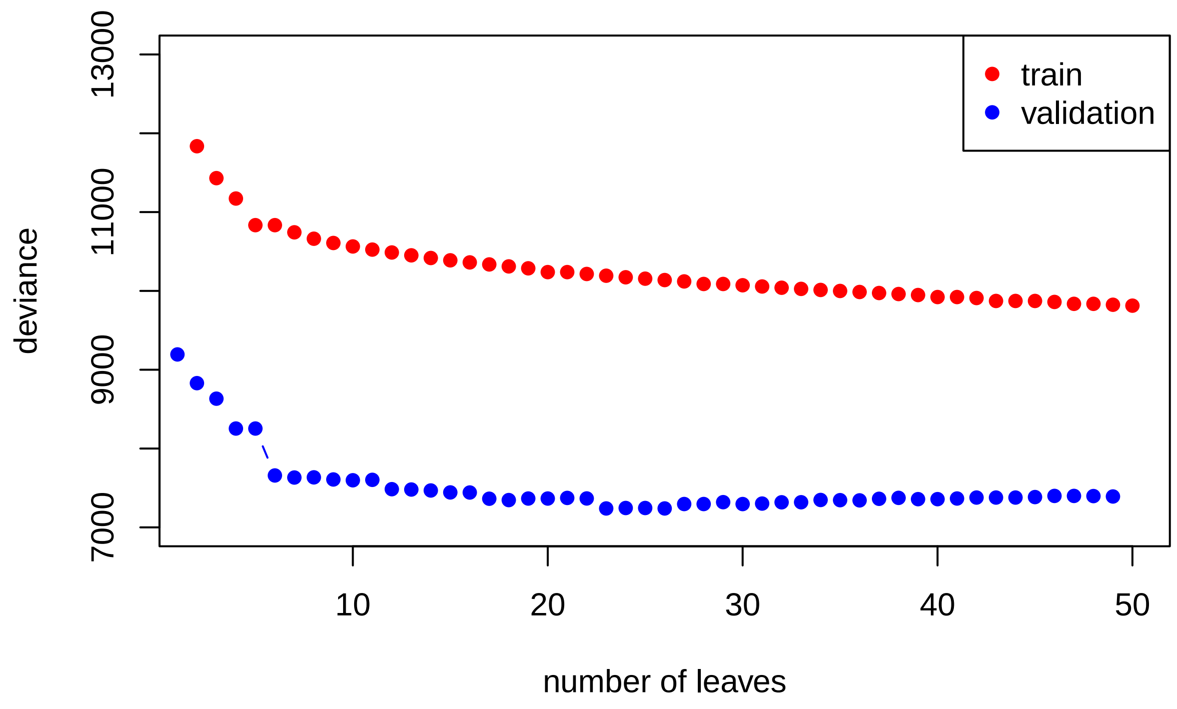
Changing the minimal node size to 7000 (default value is 10) reduces the tree a little bit (one split less than before) while setting the minimum deviance to 0.0005 (default value is 0.01) builds a noticeable bigger tree. As we have a bigger value for the minimal node size, it makes sense that the tree stops growing earlier as the requirement is not fulfilled anymore afterwards, while reducing the minimum deviance lets the tree grow bigger. The value specified for the deviance shows how big the value for a specific node that is supposed to be split has to be compared to the root node for allowing another split. So the smaller this value is, the more likely it is that we can have another split, and hence the tree grows more significant with a smaller value.

3.

Use training and validation sets to choose the optimal tree depth in the model 2c: study the trees up to 50 leaves. Present a graph of the dependence of deviances for the training and the validation data on the number of leaves and interpret this graph in terms of bias-variance trade-off.

In the summary of the third tree, one can see that it has 150 leaves at the moment, so now we will prune this tree down to the number of leaves between 2 and 50.

```
n <- 50
trainScore = rep(0, n)
validScore = rep(0, n)

for (i in 2:n) {
    prunedTree = prune.tree(tree3, best = i)
    pred = predict(prunedTree, newdata = valid, type = "tree")
    trainScore[i] = deviance(prunedTree)
    validScore[i] = deviance(pred)
}

plot(2:n, trainScore[2:n], type = "b", col = "red", pch = 16,
    xlab = "number of leaves", ylab = "deviance", ylim = c(7000,
        13000))
lines(validScore[2:n], type = "b", pch = 16, col = "blue")
legend("topright", legend = c("train", "validation"), pch = 16,
    col = c("red", "blue"))
```



The graph shows that the deviance is higher for very small numbers of leaves and decreases for the training dataset up to 50 trees. For the validation dataset, it decreases as well. However, it increases a tiny bit again after around 25 leaf nodes, which might lead to the conclusion that having more than 25 leaves might lead to an overfit to the training data.

Report the optimal amount of leaves and which variables seem to be most important for decision making in this tree. Interpret the information provided by the tree structure (not everything but most important findings).

The optimal amount of leaves seems to be 23 as it reduces the deviance of the validation dataset to a minimum and is the smallest value with this deviance. The simpler the model, the better, so we pick 23 and not 25, for example.

```r
prunedTree_opt <- prune.tree(tree3, best = which.min(validScore[-1]))  #23 nodes
summary(prunedTree_opt)
```

```
##
## Classification tree:
## snip.tree(tree = tree3, nodes = c(231L, 472L, 929L, 458L, 112L,
## 12L, 117L, 465L, 5L, 459L, 13L, 4L, 228L, 237L, 928L, 113L, 233L,
## 473L, 31L))
## Variables actually used in tree construction:
## [1] "poutcome" "day"      "month"    "contact"  "housing"  "pdays"    "job"
## [8] "campaign" "age"
## Number of terminal nodes:  23
## Residual mean deviance:  0.5644 = 10190 / 18060
## Misclassification error rate: 0.1027 = 1857 / 18084
```

In order to analyze which variables are needed the most for building the tree, we have built a table showing all nodes where a cut is made and a second table where we can see how often each variable has been used in the first table.

```r
results <- data.frame(variable = prunedTree_opt$frame$var, split_left = prunedTree_opt$frame$splits[,
    1], split_right = prunedTree_opt$frame$splits[, 2])

results[which(results$variable != "<leaf>"), ]
```

```
##    variable           split_left           split_right
## 1  poutcome                   :c                  :abd
## 2       day          :ciklnuvw12 :abdefghjmopqrstxyz034
## 5     month                :chkl             :abdefgij
## 6       day  :bcdfijkopqruvwy012        :aeghlmnstxz34
## 9   contact                  :ab                    :c
## 10    month                 :adg                :befij
## 11      day  :ahjklmorsuvwxyz023          :bcdefginpqt1
## 12      day              :jkruw2          :ahlmosvxyz03
## 15  housing                   :a                    :b
## 16    month                  :ag                    :d
## 18      day                  :pq               :bcdefi
## 21    month                   :g                   :ad
## 24      day :abcdefijlmnoptuwyz0          :ghkqrsvx1234
## 25    pdays                 <382                  >382
## 26      job               :afikl              :bcdeghj
## 27      day             :bclmoyz0           :adefijnptuw
## 28 campaign                 <1.5                  >1.5
## 34      age                <61.5                 >61.5
## 35    pdays                 <385                  >385
## 36      age                <26.5                 >26.5
## 41    month               :abdej                  :fgi
## 42      day         :aeghikmnrsw4             :dfjqtux01
```

```
sort(table(results[which(results$variable != "<leaf>"), "variable"]),
    decreasing = TRUE)[1:9]
```

```
##
##       day     month       age     pdays       job   housing   contact campaign
##         8         5         2         2         1         1         1        1
## poutcome
##         1
```

One can see that the variable day has been used eight times, and the variable month has been used five times within the tree building process. So both variables seem to be quite significant and helpful for splitting the data into the two desired groups. Nevertheless, one must also keep in mind at which point in the tree the respective variables are used as the more critical splits are done initially. Considering this, especially the variable poutcome has to be named, which is used in the first step and hence especially important for the splitting process. Other variables used for the splitting are age, pdays, job, housing, contact and campaign. All other variables are not used in our optimal tree.

4.

Estimate the confusion matrix, accuracy and F1 score for the test data by using the optimal model from step 3.

```
pred_test <- predict(prunedTree_opt, newdata = test, type = "class")
conf_matrix <- table(test$y, pred_test)
knitr::kable(conf_matrix, caption = "Test data (rows=real values, cols=prediction)")
```

Table 1: Test data (rows=real values, cols=prediction)

	yes	no
yes	330	1255
no	196	11783

```
paste("Accuracy for test data:", round((conf_matrix[1, 1] + conf_matrix[2,
    2])/sum(conf_matrix) * 100, 2), "%")
```

```
## [1] "Accuracy for test data: 89.3 %"
```

```
prec <- conf_matrix[1, 1]/(conf_matrix[2, 1] + conf_matrix[1,
    1])
recal <- conf_matrix[1, 1]/(conf_matrix[1, 2] + conf_matrix[1,
    1])
f1 <- 2 * prec * recal/(prec + recal)  # not the best value
paste("F_1 score for test data:", round(f1, 4))
```

```
## [1] "F_1 score for test data: 0.3126"
```

Comment whether the model has a good predictive power and which of the measures (accuracy or F1-score) should be preferred here.

Considering the accuracy of around 90%, the model seems to perform exceptionally well but not great. Looking at the confusion matrix, one can see that there are many more wrong classified cases to "no" that is "yes" in the real dataset than the other way around. If one thinks about the underlying problem, that would be a bad trade-off for the bank as they would have noticeable less "yes" with the prediction and would lose all the potential people that could have a deposit with them as obviously their desire is the receive as many deposits as possible. As we have a pretty imbalanced problem here, the F_1 score might, in general, be the better choice compared to the accuracy. However, the value for f_1 is quite bad; with the problem we just described, we should think about different weights for the different types of mistakes as one is a lot worse than the other: It might be okay to call a few people you do not get money from in the end while it might be pretty bad not to call the people potentially bringing lots of money with them. Hence, in the next part, this is given more weight.

5.

Perform a decision tree classification of the test data with a weighted loss matrix (0,5,1,0) and report the confusion matrix for the test data.

```r
pred_loss <- as.data.frame(predict(prunedTree_opt, newdata = test,
    type = "vector"))

pred_loss$ratio <- factor(ifelse(pred_loss$no/pred_loss$yes >=
    5, "no", "yes"), levels = c("yes", "no"))

conf_matrix_loss <- table(test$y, pred_loss$ratio)
knitr::kable(conf_matrix_loss, caption = "Test data (rows=real values, cols=prediction)")
```

Table 2: Test data (rows=real values, cols=prediction)

	yes	no
yes	792	793
no	938	11041

Compare the results with the results from step 4 and discuss how the rates have changed and why.

Compared to the previous confusion matrix, a few changes can be observed. As we were punishing the false negatives more now, the number of false negatives has reduced noticeably because they were weighted a lot more (before 1255, now 793). On the other hand, false positives have increased quite a lot (before 196, now 938). Additionally, we have many more true positives (before 330, now 792) than before and only a few true negatives less than before (before 11783, now 11041).

Suppose one thinks about that in the context of the problem that would mean that there are a lot fewer people of the ones that will have the deposit with them that they "miss" because the prediction would not have recommended to call them. To achieve that, the number of people they would call after the prediction and be unsuccessful in the real world would increase quite a lot, and can be seen as the cost they need to pay to avoid losing the other people. Nevertheless, another positive effect is that many more people who will leave the deposit have also been predicted, so the actual positive rate is a lot higher.

```r
paste("Accuracy for test data:", round((conf_matrix_loss[1, 1] +
    conf_matrix_loss[2, 2])/sum(conf_matrix_loss) * 100, 2),
    "%")
```

```
## [1] "Accuracy for test data: 87.24 %"
```

```r
prec <- conf_matrix_loss[1, 1]/(conf_matrix_loss[2, 1] + conf_matrix_loss[1,
    1])
recal <- conf_matrix_loss[1, 1]/(conf_matrix_loss[1, 2] + conf_matrix_loss[1,
    1])

f1 <- 2 * prec * recal/(prec + recal)  # not the best value
paste("F_1 score for test data:", round(f1, 4))
```

```
## [1] "F_1 score for test data: 0.4778"
```

Comparing the F_1 score and accuracy, one can see that the accuracy is a bit smaller for this model, but the F_1 score increased a bit and is therefore better. As we said earlier the F_1 score is probably the better choice in this imbalanced problem. Together with the argumentation concerning the context of the problem above, we can be happy with this improvement and prefer this model to the previous one.

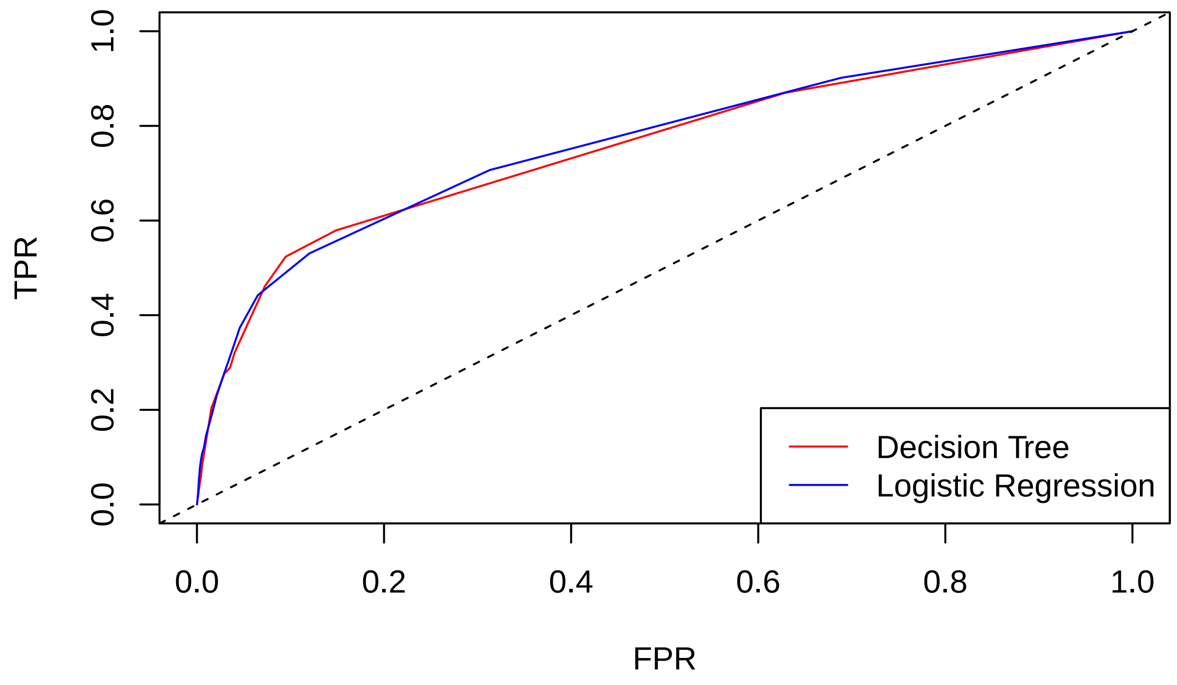
6.

Use the optimal tree and a logistic regression model to classify the test data by using different thresholds and plot the corresponding ROC curves.

As it is not completely clear whether the optimal tree is the one from step 3 or step 5, we decided to stick to the optimal tree from step 3 as it has the highest accuracy. One could have also picked the one from step 5 as it is more suitable for our problem.

```r
mySeq <- seq(0.05, 0.95, by = 0.05)
TPR <- numeric(length = length(mySeq))
FPR <- numeric(length = length(mySeq))

# for the first four sequences we can not use the following
# loop as everything is classified to the 'no' group, so we
# cn just set TPR and FPR to 0

TPR[1:4] <- 0
FPR[1:4] <- 0

# for the rest we do it step by step
for (i in 5:length(mySeq)) {
    p <- as.factor(ifelse(predict(prunedTree_opt, newdata = test,
        type = "vector")[, 2] > mySeq[i], "1", "0"))

    conf <- table(test$y, p)

    TPR[i] <- conf[1, 1]/(conf[1, 1] + conf[1, 2])
    FPR[i] <- conf[2, 1]/(conf[2, 1] + conf[2, 2])
}

# connecting to 1 in the end as it would look weird if they
# end in different positions (default roc curve would do
# that as well)
TPR[length(mySeq) + 1] <- 1
FPR[length(mySeq) + 1] <- 1
```

```r
# logistic regression
myGlm <- glm(y ~ ., data = train, family = "binomial")

TPR_l <- numeric(length = length(mySeq) + 1)
FPR_l <- numeric(length = length(mySeq) + 1)

for (i in 1:length(mySeq)) {
    p <- as.factor(ifelse(predict(myGlm, newdata = test, type = "response") >
        mySeq[i], "1", "0"))

    conf <- table(test$y, p)

    TPR_l[i] <- conf[1, 1]/(conf[1, 1] + conf[1, 2])
    FPR_l[i] <- conf[2, 1]/(conf[2, 1] + conf[2, 2])
}

TPR_l[length(mySeq) + 1] <- 1
FPR_l[length(mySeq) + 1] <- 1

plot(FPR, TPR, type = "l", col = "red", ylim = c(0, 1), xlim = c(0,
    1), ylab = "TPR", xlab = "FPR")
lines(FPR_l, TPR_l, col = "blue")
abline(a = 0, b = 1, lty = 2, col = "black")
legend("bottomright", legend = c("Decision Tree", "Logistic Regression"),
    col = c("red", "blue"), lty = 1)
```



Conclusion?

Both curves for the Decision Tree and the Logistic Regression are pretty similar, and both of them are pretty bad. A good ROC curve would be close to the top of the plot, and both of them are closer to the angle bisector.

Why could the precision-recall curve be a better option here?

As we have already mentioned earlier, we have an imbalanced problem here. The precision-recall curve is usually more informative for imbalanced problems as the FPR in the roc-curve is prone to the significant size difference between the two classes.

Assignment 3: Principal components and implicit regularization

The *communities* dataset has 1994 observations on 101 variables. It combines socio-economic data from multiple sources in the United States between 1990 and 1995.

```r
df <- read.csv("files/communities.csv")
ViolentCrimesPerPop <- df$ViolentCrimesPerPop
resp_ind <- which(colnames(df) == "ViolentCrimesPerPop")
```

1.

Scale all variables except of ViolentCrimesPerPop and implement PCA by using `eigen()`

```r
data_scaled <- scale(df[, -resp_ind])
covar_mat <- var(data_scaled)
eigen_decomp <- eigen(covar_mat)
eigen_vec <- eigen_decomp$vectors
eigen_val <- eigen_decomp$values
```

How many features are needed to obtain at least 95% of variance in the data? What is the proportion of variance explained by each of the first two principal components

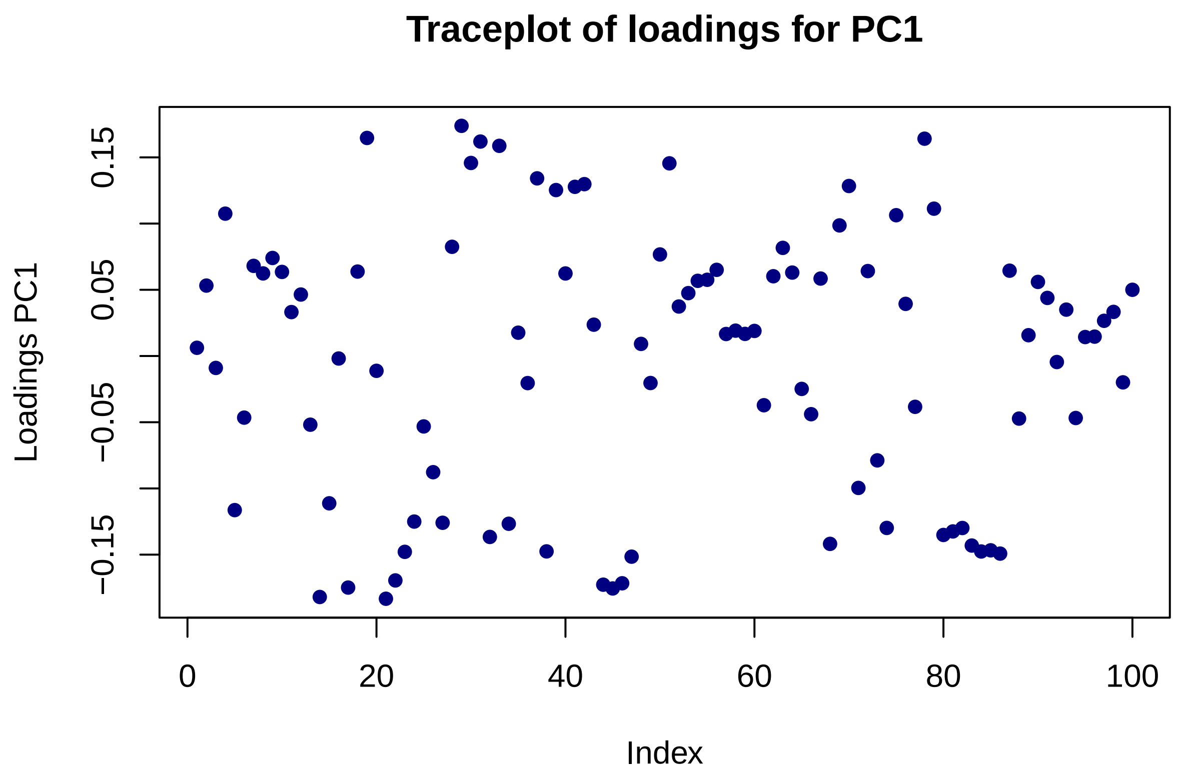
We need at least 35 components to account for 95% of the variance present in the original scaled data. The first component explains 25.02% of the variance, and the second explains 16.94% of the variance, giving them a total of 41.95% of variance explained together.

```r
which(round(cumsum(eigen_val/sum(eigen_val) * 100), 2) >= 95)[1]
sum(eigen_val[1:2])
```

2.

Repeat PCA by using `princomp()` and make a score plot of the first principal components

```r
fit <- princomp(data_scaled)
scores <- as.data.frame(fit$scores)
plot(fit$loadings[, 1], main = "Traceplot of loadings for PC1",
    ylab = "Loadings PC1", pch = 16, col = "navy")
```



Do many features have a notable contribution to the first component? Report which 5 features contribute mostly by the absolute value. Does there features have anything in common? May the have a logical relationship to the crime level?

There are quite a lot of variables that contribute some information to the first principal component. There are, for instance, 40 variables that have a loading ≥ 0.1. The 5 variables that contribute the most can be seen below:

```r
PC1_load <- sort(abs(fit$loadings[, 1]), decreasing = T)
sum(PC1_load >= 0.1)  # 40
head(PC1_load, 5)
```

They are the following:

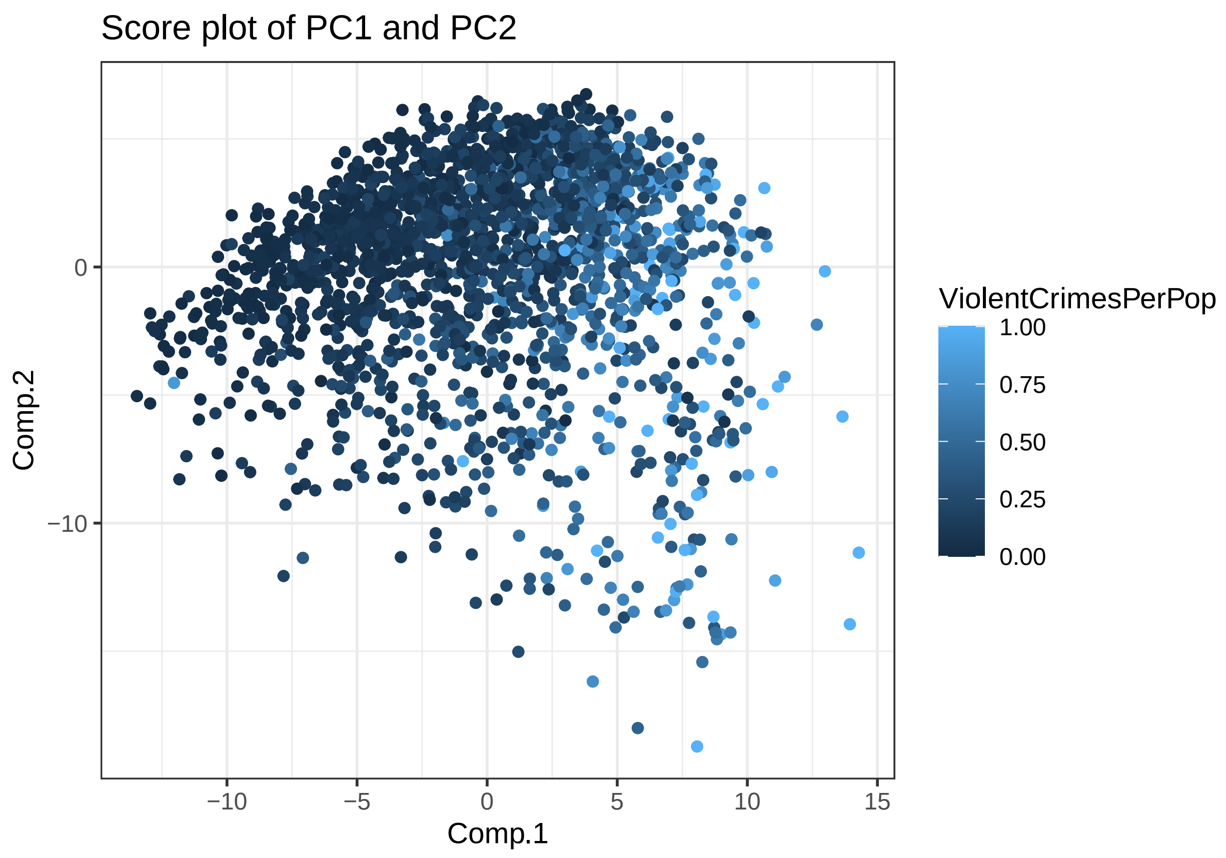
- medFamInc: median family income (differs from household income for non-family households) (**Loading: -0.183**)
- medIncome: median household income (**Loading: -0.182**)
- PctKids2Par: percentage of kids in family housing with two parents (**Loading: -0.176**)
- pctWInvInc: percentage of households with investment / rent income in 1989 (**Loading: -0.175**)
- PctPopUnderPov: percentage of people under the poverty level (**Loading: 0.174**)

These variables are related to household economic situations and poverty. Therefore, they can be seen as indicators of socio-economic levels in the different communities. The percentage of people under the poverty line has a logical relationship with increased levels of crime. Median family income and median household income are highly correlated and contain the same information (collinear). They negatively correlate with the principal component, meaning that higher median income results in a lower PCA score. The percentage of kids in family housing with two parents can be seen as an indicator of socio-economic levels in a community. In general, two parents can contribute more to the family economy. The percentage of households with investment income in 1989 is a direct indicator of the economic situation in the community.

Provide a plot of the PC scores in the coordinates (PC1, PC2) in which the color of points is given by ViolentCrimesPerPop. Analyse.

It is possible to identify a trend in ViolentCrimesPerPop given the two principal components. Higher values on PC1 has a moderately strong positive relationship with ViolentCrimesPerPop. PC2 seem to have a negative relationship with ViolentCrimesPerPop, but it seems weaker than the first component. Finally, an interesting outlier has a low score on PC1 and a high value on ViolentCrimesPerPop.

```
ggplot(data = scores, aes(x = Comp.1, y = Comp.2, color = ViolentCrimesPerPop)) +
    geom_point() + labs(title = "Score plot of PC1 and PC2") +
    theme_bw()
```



3.

Scale the original data and split into training and test (50/50) and estimate a linear regression model.

In terms of predictions, the training and testing error is relatively low. It seems that using all of these variables, it is possible to create a model that can reasonably accurately predict ViolentCrimesPerPop. There are, however, many predictors in the model that are unnecessary since they cannot be significantly differentiated from 0. There is most likely multicollinearity in the model and low interpretability of the feature's effects. From a prediction perspective, the model can be deemed adequate. Having a higher MSE for the test data than for the training data shows a tendency to overfitting.

```r
# Split
n <- nrow(df)
set.seed(12345)
id <- sample(1:n, floor(n * 0.5))
df_train <- df[id, ]  # Training data
df_test <- df[-id, ]  # Test data

# Scale test data by mean and std of training data
mean_train <- numeric(ncol(df_train))
sd_train <- numeric(ncol(df_train))

for (i in 1:ncol(df_train)) {
    mean_train[i] <- mean(df_train[, i])
    sd_train[i] <- sd(df_train[, i])
}

df_train <- data.frame(scale(df_train))

df_test_temp <- data.frame(matrix(nrow = nrow(df_test), ncol = ncol(df_test)))
for (i in 1:ncol(df_train)) {
    df_test_temp[, i] <- (df_test[, i] - mean_train[i])/sd_train[i]
}
df_test <- as.data.frame(df_test_temp)
colnames(df_test) <- colnames(df_train)

# Linear regression model
fit_lm <- caret::train(ViolentCrimesPerPop ~ ., data = df_train,
    method = "lm")
# Errors
training_error <- mean((df_train[, 101] - predict(fit_lm))^2)
test_error <- mean((df_test[, 101] - predict(fit_lm, df_test))^2)

cat("Training Error: ", training_error, "\nTest Error: ", test_error)
```

```
## Training Error:  0.2752071
## Test Error:  0.4248011
```

4.

Implement a function that depends on parameter vector θ, and represents the cost function for linear regression without intercept on the training data

```r
# Calculate cost given theta
cost_fun <- function(theta, data) {
    X <- as.matrix(data[, -101])
    y <- data[, 101]
    n <- length(y)
    stopifnot(length(theta) == ncol(X), length(y) == nrow(X))
    cost <- (1/n) * sum((X %*% theta - y)^2)
    return(cost)
}
```

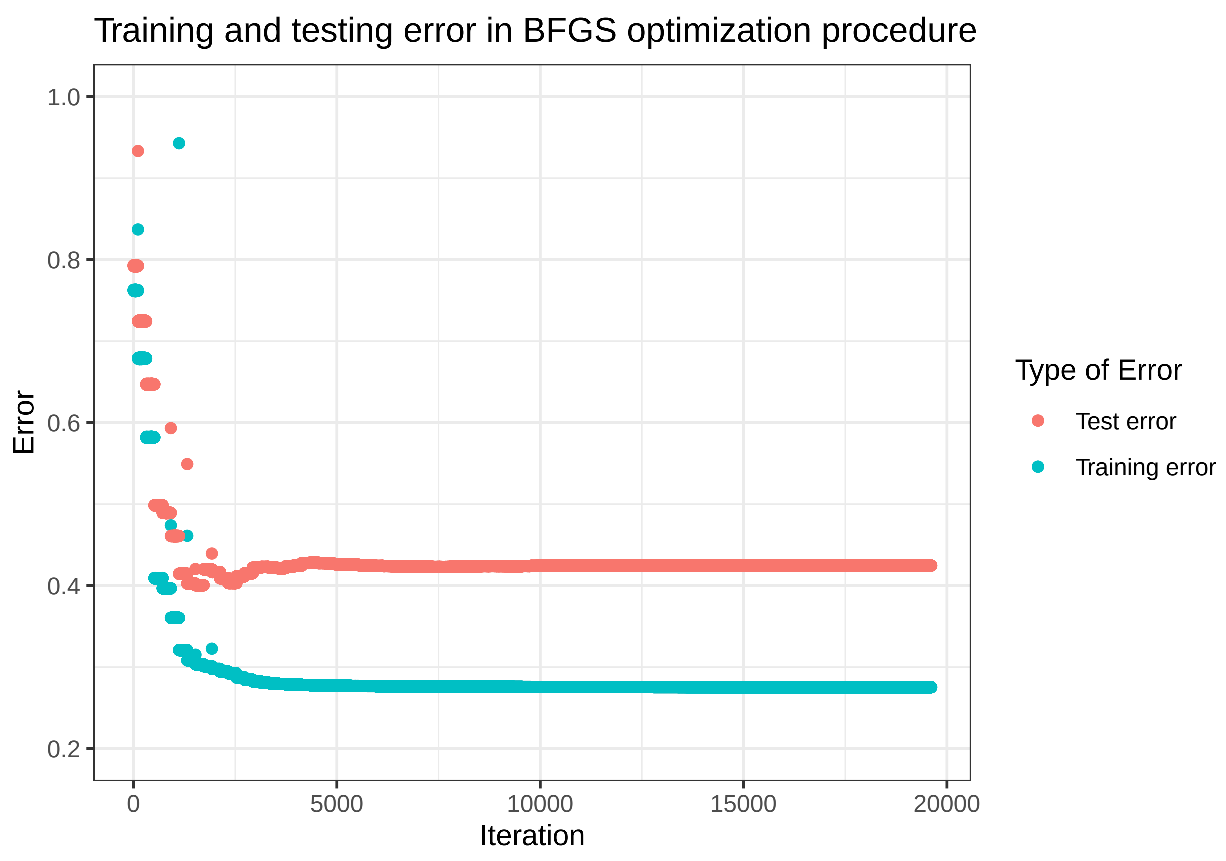
Use BFGS optimization to optimize the cost and compute training and test errors for every iteration

```
# Create new environment to store iterations
parameter_env <- new.env()
# parameter_env$iteration <- list() Create lists in new env
# to store data
parameter_env$cost_train <- list()
parameter_env$cost_test <- list()

# Wrapper function
cost.lm <- function(theta, training_data, testing_data) {
    n <- length(parameter_env[["cost_train"]])
    # parameter_env[['iteration']][[n+1]] <- theta
    cost_train <- cost_fun(theta, data = training_data)  # Call cost on training
    parameter_env[["cost_train"]][[n + 1]] <- cost_train  # store cost train
    cost_test <- cost_fun(theta, data = testing_data)  # call cost on testing
    parameter_env[["cost_test"]][[n + 1]] <- cost_test  # store cost test
    return(cost_train)
}
# Optimize train cost function through optimizing the
# wrapper function
opt <- optim(rep(0, 100), fn = cost.lm, training_data = df_train,
    testing_data = df_test, gr = NULL, method = "BFGS")
# Gather errors and discard first 500 iter
train_errors <- as.numeric(do.call(rbind, parameter_env[["cost_train"]]))[-c(1:500)]
test_errors <- as.numeric(do.call(rbind, parameter_env[["cost_test"]]))[-c(1:500)]
```

Present a plot showing the dependence of both errors on the iteration number

The following plot shows the dependence of training and testing error in iterations in the BFGS procedure above. The y-axis has been limited to show only values between [0.2, 1] to better display the increase in test error after many iterations. As shown in the plot, implicit regularization seems to be a good idea for this problem, as the testing error increases after about 2000 iterations. Finding the minimum cost function for the training data leads to overfitting the training data. By stopping early, it is possible to achieve a model better at generalizing to the test data.

```
# Plot
ggplot() + geom_point(aes(x = 1:length(train_errors), y = train_errors,
    color = "Training error")) + geom_point(aes(x = 1:length(test_errors),
    y = test_errors, color = "Test error")) + theme_bw() + labs(title = "Training and testing error in I
    color = "Type of Error", y = "Error", x = "Iteration") +
    ylim(c(0.2, 1))
```



Find the optimal iteration number according to early stopping criterion

```r
# Find optimal set of parameters
optim_ind <- which.min(test_errors)
optimal_train <- train_errors[optim_ind]
optimal_test <- test_errors[optim_ind]

optim_errors <- data.frame(`Training Error` = c(training_error,
    optimal_train), `Test Error` = c(test_error, optimal_test))
row.names(optim_errors) <- c("Fully optimized model", "Early stopping model")
optim_errors
```

```
##                         Training.Error Test.Error
## Fully optimized model        0.2752071  0.4248011
## Early stopping model         0.3032999  0.4002329
```

```r
cat("Optimal iteration is", optim_ind + 500)
```

```
## Optimal iteration is 2182
```

The model when optimization of $\underset{\theta}{\operatorname{argmin}} \frac{1}{n}||\boldsymbol{X\theta} - \boldsymbol{y}||_2^2$ is stopped early is better than the model when the optimization is run until convergence. The fact that the test error increases by the number of iterations are an indication of overfitting to the training data. The differences are, however, fairly low in this example. Still, implicit regularization here generates a model that is better at generalizing to new data.

Statement of Contribution

We first worked on all assignments ourselves, and everyone finished all of them. Christoforos was mainly concentrating on assignment 1 and wrote down the interpretations there. Lisa focused most on assignment 2 and wrote the interpretation for these tasks while Theodor was doing the same for assignment 3. After everyone finished, we discussed our results and adjusted a few things.

Code

```r
library(formatR)
library(ggplot2)
library(caret)
library(tree)
library(rpart)  # for loss matrix in 2.5

knitr::opts_chunk$set(echo = TRUE)
knitr::opts_chunk$set(tidy.opts = list(width.cutoff = 60), tidy = TRUE)
data <- read.csv("files/tecator.csv")

n = dim(data)[1]
set.seed(12345)
id = sample(1:n, floor(n * 0.5))
train = data[id, ]
test = data[-id, ]
train <- train[, -c(1, 103, 104)]
test <- test[, -c(1, 103, 104)]

lrm <- lm(Fat ~ ., data = train)
# summary(lrm)

train_error <- mean((train$Fat - predict(lrm))^2)
train_error
test_error <- mean((test$Fat - predict(lrm, test))^2)
test_error
library(glmnet)

X <- train[, -101]
Y <- data.frame(train[, 101])

lasso <- glmnet(as.matrix(X), as.matrix(Y), alpha = 1)

# summary(lasso) log(lasso$lambda)

plot(lasso, xvar = "lambda", label = TRUE)


for (i in lasso$lambda) {
    nparam <- sum(coef(lasso, s = i)[, 1] != 0)
    if (nparam == 4) {
        lambda <- i
        break
    }
}
cat("Optimal lambda:", lambda)
ridge <- glmnet(as.matrix(X), as.matrix(Y), alpha = 0)

plot(ridge, xvar = "lambda", label = TRUE)
cv_lasso <- cv.glmnet(as.matrix(X), as.matrix(Y), alpha = 1)
cat("Optimal lambda:", cv_lasso$lambda.min)

sum(coef(cv_lasso, s = cv_lasso$lambda.min)[, 1] != 0) - 1  #substract intercept

plot(cv_lasso)
optimal.lambda <- predict(cv_lasso, as.matrix(test[, -101]),
    s = cv_lasso$lambda.min)

optimal_df <- data.frame(Actual = test$Fat, Predicted = optimal.lambda)

library(ggplot2)

my_scatterplot <- ggplot(optimal_df, aes(x = s1, y = Actual)) +
    geom_point(color = "#00aedb") + lims(x = c(0, 65), y = c(0,
    65)) + geom_abline(color = "#d11141") + labs(title = "Model Correspondence") +
    xlab("Predicted Values") + ylab("Actual Values") + theme_bw()

my_scatterplot
bank <- read.csv2("files/bank-full.csv", header = TRUE)

# remove duration
df <- bank[, -12]

# changing target to factor
df$y <- factor(df$y, levels = c("yes", "no"))

# most other variables are supposed to be factor variables
# as well
for (i in c(2:5, 7:11, 15)) {
    df[, i] <- as.factor(df[, i])
}

## split like specified in the lecture slides
n = dim(df)[1]
set.seed(12345)
id = sample(1:n, floor(n * 0.4))
train = df[id, ]

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

id3 = setdiff(id1, id2)
test = df[id3, ]
# define misclassification function (return in percent)
```

```r
misclass = function(X, X1) {
    return(round(1 - sum(diag(table(X, X1)))/length(X), 4))
}

tree1 <- tree(y ~ ., data = train)
summary(tree1)
tree2 <- tree(y ~ ., data = train, control = tree.control(nobs = nrow(train),
    minsize = 7000))
summary(tree2)
tree3 <- tree(y ~ ., data = train, control = tree.control(nobs = nrow(train),
    mindev = 5e-04))
summary(tree3)

train1 <- predict(tree1, type = "class")
train2 <- predict(tree2, type = "class")
train3 <- predict(tree3, type = "class")

valid1 <- predict(tree1, newdata = valid, type = "class")
valid2 <- predict(tree2, newdata = valid, type = "class")
valid3 <- predict(tree3, newdata = valid, type = "class")

errors_df <- data.frame(`Training Error` = c(misclass(train$y,
    train1), misclass(train$y, train2), misclass(train$y, train3)),
    `Validation Error` = c(misclass(valid$y, valid1), misclass(valid$y,
        valid2), misclass(valid$y, valid3)))

row.names(errors_df) <- c("Tree A", "Tree B", "Tree C")
errors_df
n <- 50
trainScore = rep(0, n)
validScore = rep(0, n)

for (i in 2:n) {
    prunedTree = prune.tree(tree3, best = i)
    pred = predict(prunedTree, newdata = valid, type = "tree")
    trainScore[i] = deviance(prunedTree)
    validScore[i] = deviance(pred)
}

plot(2:n, trainScore[2:n], type = "b", col = "red", pch = 16,
    xlab = "number of leaves", ylab = "deviance", ylim = c(7000,
        13000))
lines(validScore[2:n], type = "b", pch = 16, col = "blue")
legend("topright", legend = c("train", "validation"), pch = 16,
    col = c("red", "blue"))
prunedTree_opt <- prune.tree(tree3, best = which.min(validScore[-1]))  #23 nodes
summary(prunedTree_opt)
results <- data.frame(variable = prunedTree_opt$frame$var, split_left = prunedTree_opt$frame$splits[,
    1], split_right = prunedTree_opt$frame$splits[, 2])

results[which(results$variable != "<leaf>"), ]

sort(table(results[which(results$variable != "<leaf>"), "variable"]),
    decreasing = TRUE)[1:9]
pred_test <- predict(prunedTree_opt, newdata = test, type = "class")
conf_matrix <- table(test$y, pred_test)
knitr::kable(conf_matrix, caption = "Test data (rows=real values, cols=prediction)")

paste("Accuracy for test data:", round((conf_matrix[1, 1] + conf_matrix[2,
    2])/sum(conf_matrix) * 100, 2), "%")

prec <- conf_matrix[1, 1]/(conf_matrix[2, 1] + conf_matrix[1,
    1])
recal <- conf_matrix[1, 1]/(conf_matrix[1, 2] + conf_matrix[1,
    1])
f1 <- 2 * prec * recal/(prec + recal)  # not the best value
paste("F_1 score for test data:", round(f1, 4))
pred_loss <- as.data.frame(predict(prunedTree_opt, newdata = test,
    type = "vector"))

pred_loss$ratio <- factor(ifelse(pred_loss$no/pred_loss$yes >=
    5, "no", "yes"), levels = c("yes", "no"))

conf_matrix_loss <- table(test$y, pred_loss$ratio)
knitr::kable(conf_matrix_loss, caption = "Test data (rows=real values, cols=prediction)")
paste("Accuracy for test data:", round((conf_matrix_loss[1, 1] +
    conf_matrix_loss[2, 2])/sum(conf_matrix_loss) * 100, 2),
    "%")

prec <- conf_matrix_loss[1, 1]/(conf_matrix_loss[2, 1] + conf_matrix_loss[1,
    1])
recal <- conf_matrix_loss[1, 1]/(conf_matrix_loss[1, 2] + conf_matrix_loss[1,
    1])

f1 <- 2 * prec * recal/(prec + recal)  # not the best value
paste("F_1 score for test data:", round(f1, 4))
mySeq <- seq(0.05, 0.95, by = 0.05)
TPR <- numeric(length = length(mySeq))
FPR <- numeric(length = length(mySeq))

# for the first four sequences we can not use the following
# loop as everything is classified to the 'no' group, so we
# cn just set TPR and FPR to 0

TPR[1:4] <- 0
FPR[1:4] <- 0

# for the rest we do it step by step
for (i in 5:length(mySeq)) {
    p <- as.factor(ifelse(predict(prunedTree_opt, newdata = test,
        type = "vector")[, 2] > mySeq[i], "1", "0"))

    conf <- table(test$y, p)

    TPR[i] <- conf[1, 1]/(conf[1, 1] + conf[1, 2])
    FPR[i] <- conf[2, 1]/(conf[2, 1] + conf[2, 2])
```

```r
}

# connecting to 1 in the end as it would look weird if they
# end in different positions (default roc curve would do
# that as well)
TPR[length(mySeq) + 1] <- 1
FPR[length(mySeq) + 1] <- 1

# logistic regression
myGlm <- glm(y ~ ., data = train, family = "binomial")

TPR_l <- numeric(length = length(mySeq) + 1)
FPR_l <- numeric(length = length(mySeq) + 1)

for (i in 1:length(mySeq)) {
    p <- as.factor(ifelse(predict(myGlm, newdata = test, type = "response") >
        mySeq[i], "1", "0"))

    conf <- table(test$y, p)

    TPR_l[i] <- conf[1, 1]/(conf[1, 1] + conf[1, 2])
    FPR_l[i] <- conf[2, 1]/(conf[2, 1] + conf[2, 2])
}

TPR_l[length(mySeq) + 1] <- 1
FPR_l[length(mySeq) + 1] <- 1

plot(FPR, TPR, type = "l", col = "red", ylim = c(0, 1), xlim = c(0,
    1), ylab = "TPR", xlab = "FPR")
lines(FPR_l, TPR_l, col = "blue")
abline(a = 0, b = 1, lty = 2, col = "black")
legend("bottomright", legend = c("Decision Tree", "Logistic Regression"),
    col = c("red", "blue"), lty = 1)
df <- read.csv("files/communities.csv")
ViolentCrimesPerPop <- df$ViolentCrimesPerPop
resp_ind <- which(colnames(df) == "ViolentCrimesPerPop")
data_scaled <- scale(df[, -resp_ind])
covar_mat <- var(data_scaled)
eigen_decomp <- eigen(covar_mat)
eigen_vec <- eigen_decomp$vectors
eigen_val <- eigen_decomp$values
which(round(cumsum(eigen_val/sum(eigen_val) * 100), 2) >= 95)[1]
sum(eigen_val[1:2])
fit <- princomp(data_scaled)
scores <- as.data.frame(fit$scores)
plot(fit$loadings[, 1], main = "Traceplot of loadings for PC1",
    ylab = "Loadings PC1", pch = 16, col = "navy")
PC1_load <- sort(abs(fit$loadings[, 1]), decreasing = T)
sum(PC1_load >= 0.1)  # 40
head(PC1_load, 5)
ggplot(data = scores, aes(x = Comp.1, y = Comp.2, color = ViolentCrimesPerPop)) +
    geom_point() + labs(title = "Score plot of PC1 and PC2") +
    theme_bw()
```

```r
# Split
n <- nrow(df)
set.seed(12345)
id <- sample(1:n, floor(n * 0.5))
df_train <- df[id, ]  # Training data
df_test <- df[-id, ]  # Test data

# Scale test data by mean and std of training data
mean_train <- numeric(ncol(df_train))
sd_train <- numeric(ncol(df_train))

for (i in 1:ncol(df_train)) {
    mean_train[i] <- mean(df_train[, i])
    sd_train[i] <- sd(df_train[, i])
}

df_train <- data.frame(scale(df_train))

df_test_temp <- data.frame(matrix(nrow = nrow(df_test), ncol = ncol(df_test)))
for (i in 1:ncol(df_train)) {
    df_test_temp[, i] <- (df_test[, i] - mean_train[i])/sd_train[i]
}
df_test <- as.data.frame(df_test_temp)
colnames(df_test) <- colnames(df_train)

# Linear regression model
fit_lm <- caret::train(ViolentCrimesPerPop ~ ., data = df_train,
    method = "lm")
# Errors
training_error <- mean((df_train[, 101] - predict(fit_lm))^2)
test_error <- mean((df_test[, 101] - predict(fit_lm, df_test))^2)

cat("Training Error: ", training_error, "\nTest Error: ", test_error)
# Calculate cost given theta
cost_fun <- function(theta, data) {
    X <- as.matrix(data[, -101])
    y <- data[, 101]
    n <- length(y)
    stopifnot(length(theta) == ncol(X), length(y) == nrow(X))
    cost <- (1/n) * sum((X %*% theta - y)^2)
    return(cost)
}
# Create new environment to store iterations
parameter_env <- new.env()
# parameter_env$iteration <- list() Create lists in new env
# to store data
parameter_env$cost_train <- list()
parameter_env$cost_test <- list()

# Wrapper function
cost.lm <- function(theta, training_data, testing_data) {
    n <- length(parameter_env[["cost_train"]])
    # parameter_env[['iteration']][[n+1]] <- theta
    cost_train <- cost_fun(theta, data = training_data)  # Call cost on training
    parameter_env[["cost_train"]][[n + 1]] <- cost_train  # store cost train
    cost_test <- cost_fun(theta, data = testing_data)  # call cost on testing
    parameter_env[["cost_test"]][[n + 1]] <- cost_test  # store cost test
    return(cost_train)
}
# Optimize train cost function through optimizing the
# wrapper function
opt <- optim(rep(0, 100), fn = cost.lm, training_data = df_train,
    testing_data = df_test, gr = NULL, method = "BFGS")
# Gather errors and discard first 500 iter
train_errors <- as.numeric(do.call(rbind, parameter_env[["cost_train"]]))[-c(1:500)]
test_errors <- as.numeric(do.call(rbind, parameter_env[["cost_test"]]))[-c(1:500)]
# Plot
ggplot() + geom_point(aes(x = 1:length(train_errors), y = train_errors,
    color = "Training error")) + geom_point(aes(x = 1:length(test_errors),
    y = test_errors, color = "Test error")) + theme_bw() + labs(title = "Training and testing error in 
    color = "Type of Error", y = "Error", x = "Iteration") +
    ylim(c(0.2, 1))
# Find optimal set of parameters
optim_ind <- which.min(test_errors)
optimal_train <- train_errors[optim_ind]
optimal_test <- test_errors[optim_ind]

optim_errors <- data.frame(`Training Error` = c(training_error,
    optimal_train), `Test Error` = c(test_error, optimal_test))
row.names(optim_errors) <- c("Fully optimized model", "Early stopping model")
optim_errors

cat("Optimal iteration is", optim_ind + 500)
```