lab2

2022-11-22

```
## [1] "Test error"
```

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

```
## [1] "Train error"
```

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

Comment on the quality of fit and prediction and therefore on the quality of model

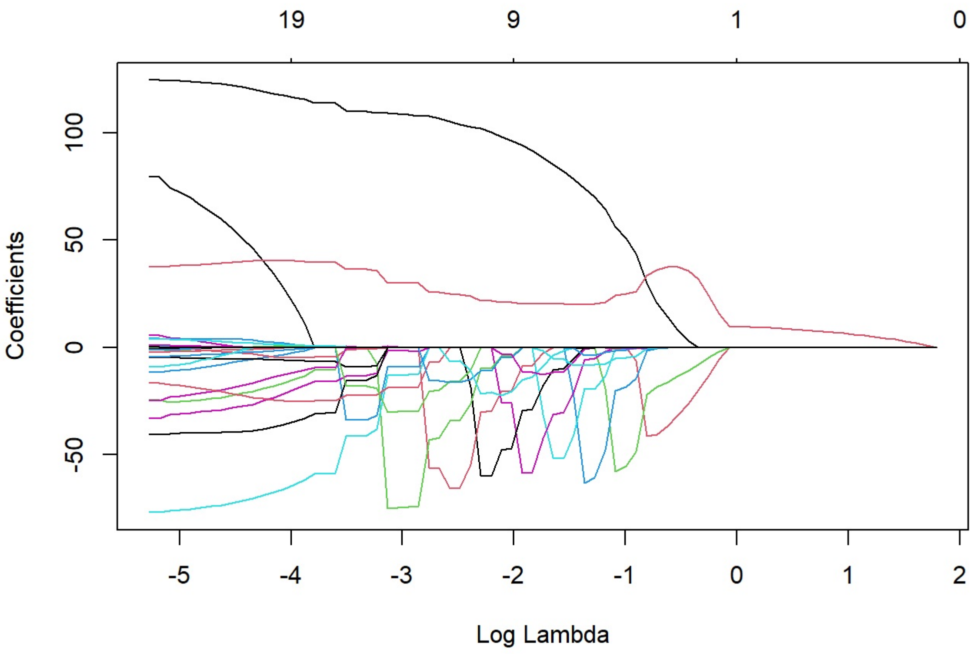
If we calculate the mean square error of the training prediction vs the test prediction we see that we get extremely high error from test compare to training. This means that the model is very specific for the data that it is trained on. Which means that the quality of the model is not great for predicting other than particularity the training data. Overfitting.

Exercise 2 L1 (also called lasso) regression model

L1 Cost function looks like this from the course elitterature:

Exercise 2
This is the cost function for L1 - regularization. The extra penalty term is $\lambda||\theta||_1$ for lasso regression.

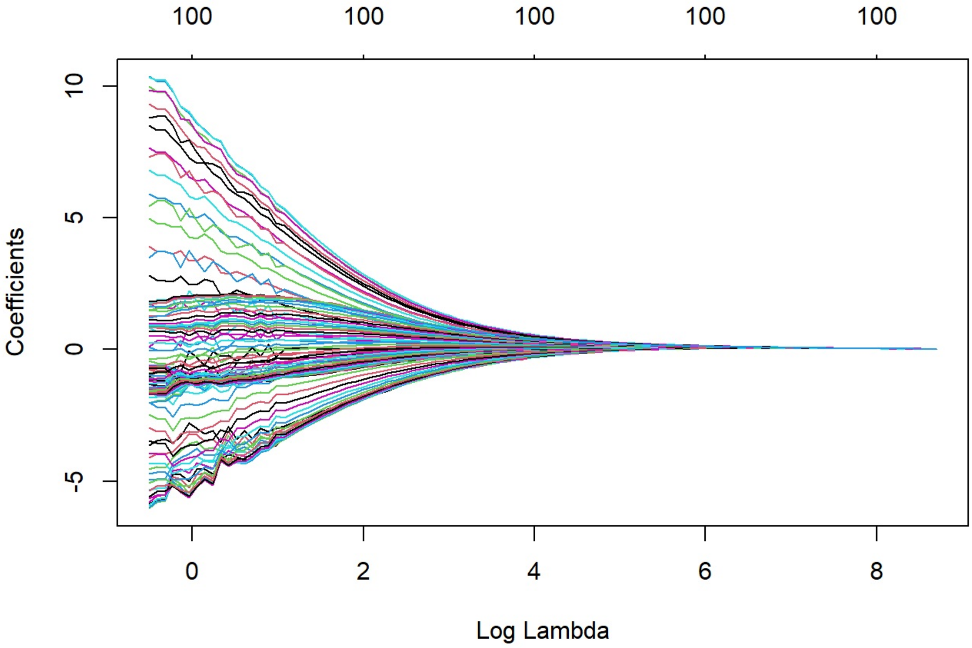
$$\theta_{hat} = arg\ min_{\theta} 1/n * ||X\theta - y||_2^2 + \lambda||\theta||_1$$



What value of the penalty factor can be chosen if we want to select model with only three features?

From my interpretation, to select a model with 3 feature you need to pick a log-lambda value of -0.5.

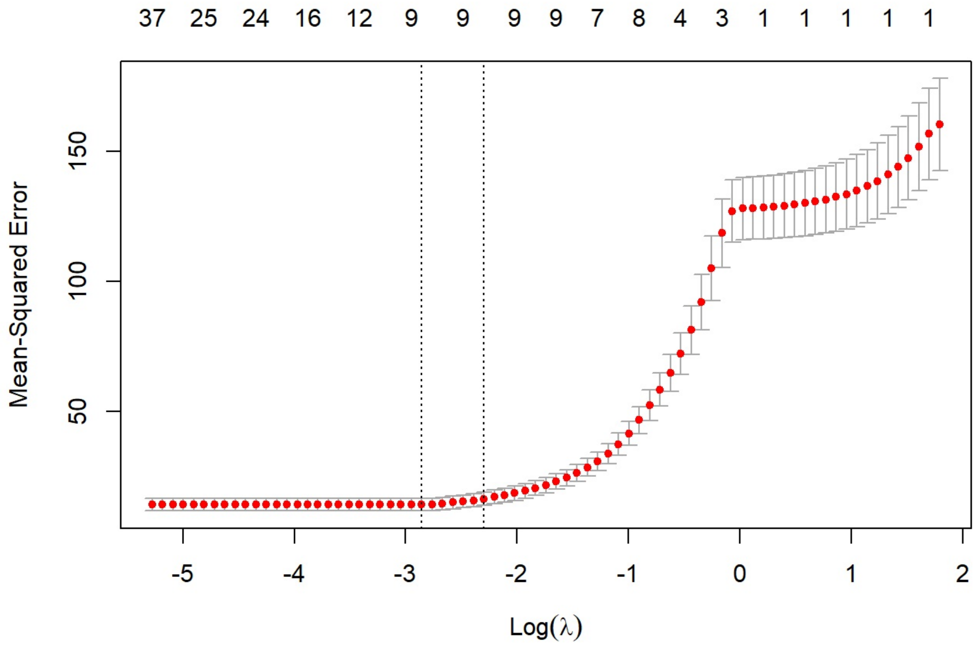
Exercise 4

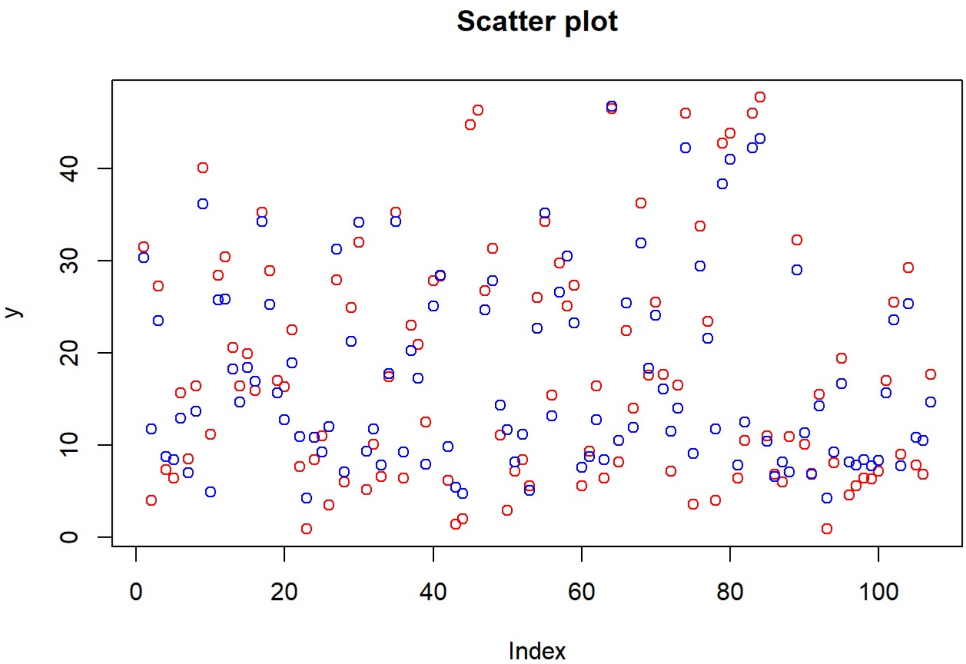


You can not gather much information from ridge regression plot.

In ridge regression plot we can clearly see that the coefficients are penalized in a way that makes the least effective in the estimation shrink faster. In the lasso regression plot we cannot make any conclusions about the coefficients in the same way. However, we can draw conclusions about the features being used depending on the log-lambda values.

Exercise 5





The CV score increases as lambda increases, where around 0 the increase slows down.

We interpret that the lambda_min is the optimal lambda and uses a model of 9 features.

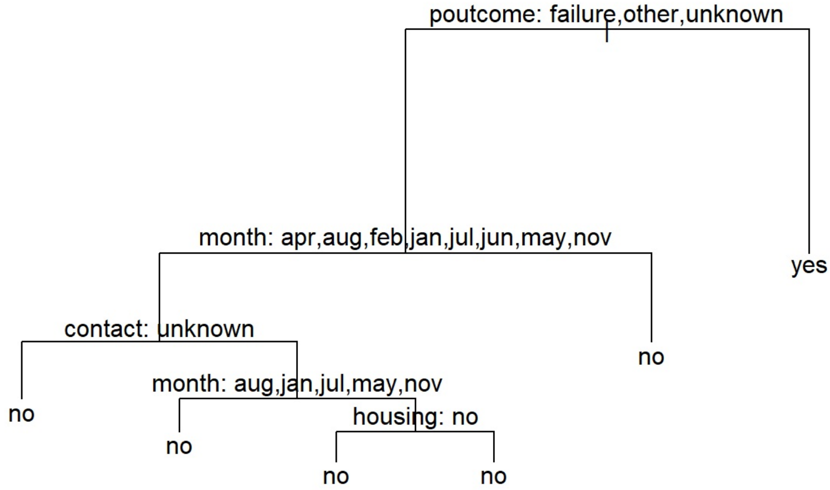
It would seem that they have the same amount of CV-score but the optmal lambda uses less model features.Therefor it would not generate significantly better predictions then lambda = -4.

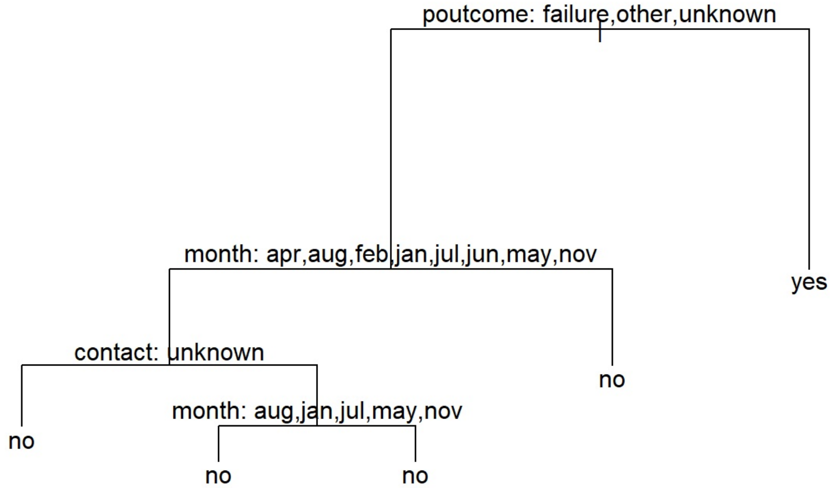
Using the scatter plot we conclude that the predictions are very good where they predict almost in the same place as the original data.

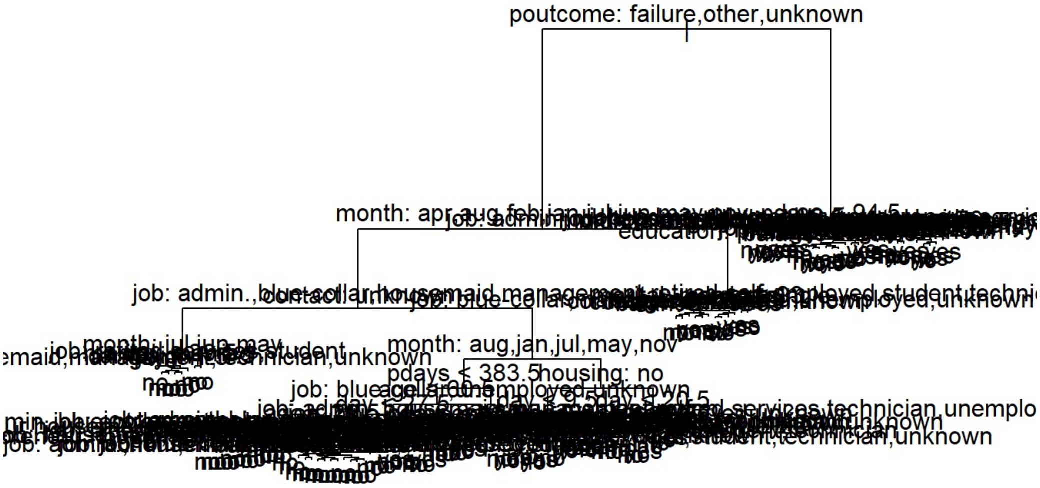
Task 2.

Exercise 1

Exercise 2


poutcome: failure,other,unknown
month: apr,aug,feb,jan,jul,jun,may,nov
yes
contact: unknown
no
month: aug,jan,jul,may,nov
housing: no
no
no
no
no


poutcome: failure,other,unknown
month: apr,aug,feb,jan,jul,jun,may,nov
yes
contact: unknown
no
month: aug,jan,jul,may,nov
no
no
no


poutcome: failure,other,unknown
month: aug,jan,jul,may,nov
pdays < 383.5
housing: no

Misclassification rate

```
## [1] "1) Training and validation"
```

```
## [1] 0.1048441
```

```
## [1] 0.1092679
```

```
## [1] "2) Training and validation"
```

```
## [1] 0.1048441
```

```
## [1] 0.1092679
```

```
## [1] "3) Training and validation"
```

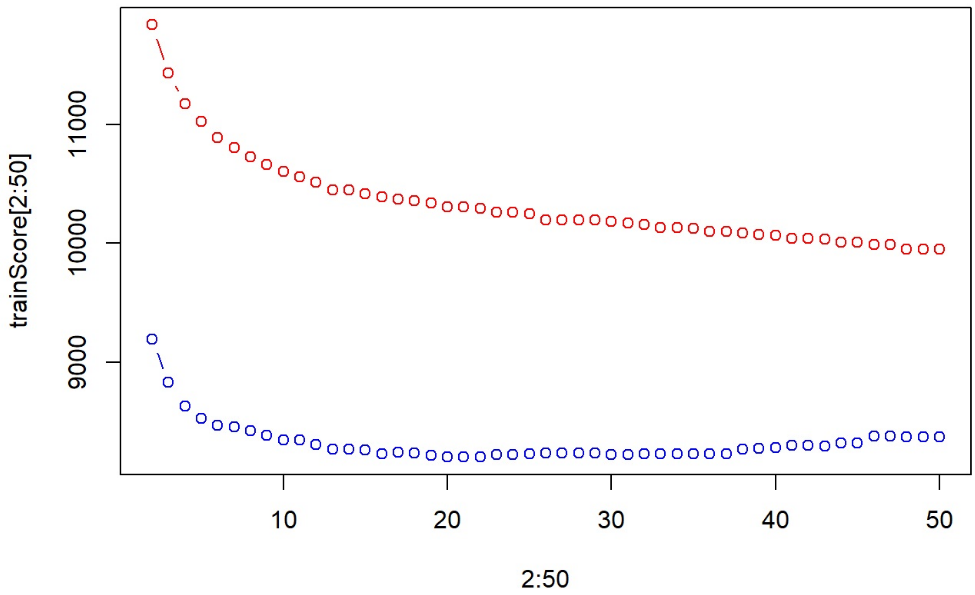
```
## [1] 0.09400575
```

```
## [1] 0.1119221
```

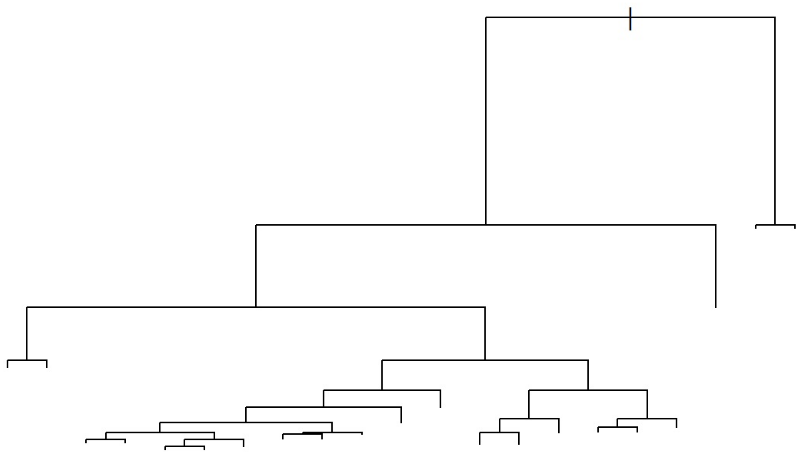
The best model among these three seems to be c, which changed the deviance. It lowered the misclassification rate for the training data, but increased it for the validation data. Not sure why.

Changing deviance resulted in a much larger tree, probably because more values were allowed to be included even though the deviance was low (this is my guess). Setting minsize to 7000 made the tree smaller, as it didn't expand the last node, "housing", compared to the original tree.

Exercise 3




```
## [1] 21
```



index 20-22, should represent 17-19 leaves. Looks like there's no difference on the data between 20-22, so all of these options should be equally good. Information provided by tree structure? Not sure at all.

Exercise 4

Accuracy = .8959 Recall = .9055 F1 = .9437

The accuracy is close to 90%, so the predictive power of the model seems to be quite good. The F1 score is usually preferred when data is unbalanced (for instance, when the quantity of examples in one class outnumbers the ones from the other class). I think F1 is better in this regard.

Exercise 5

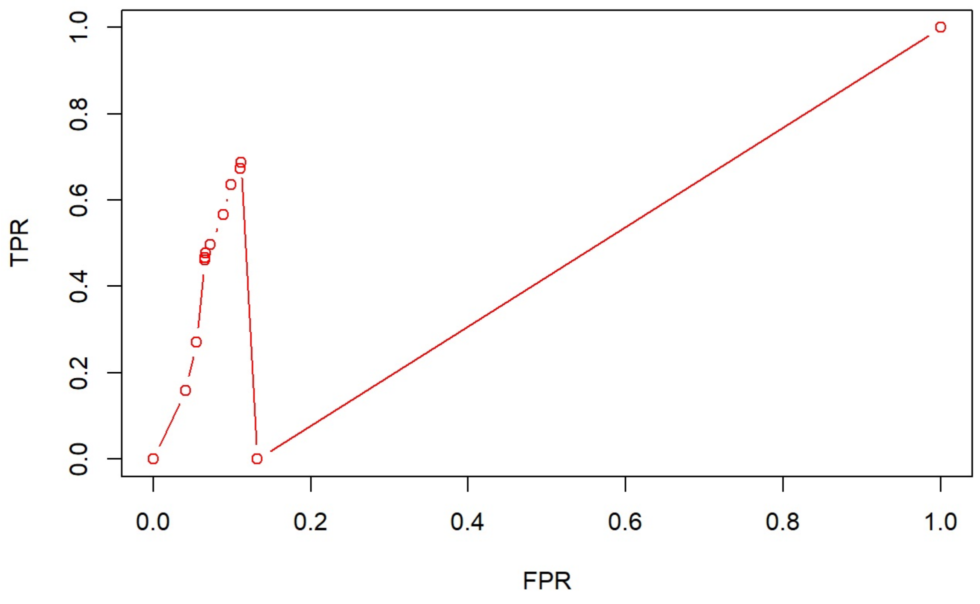
Accuracy = .8869

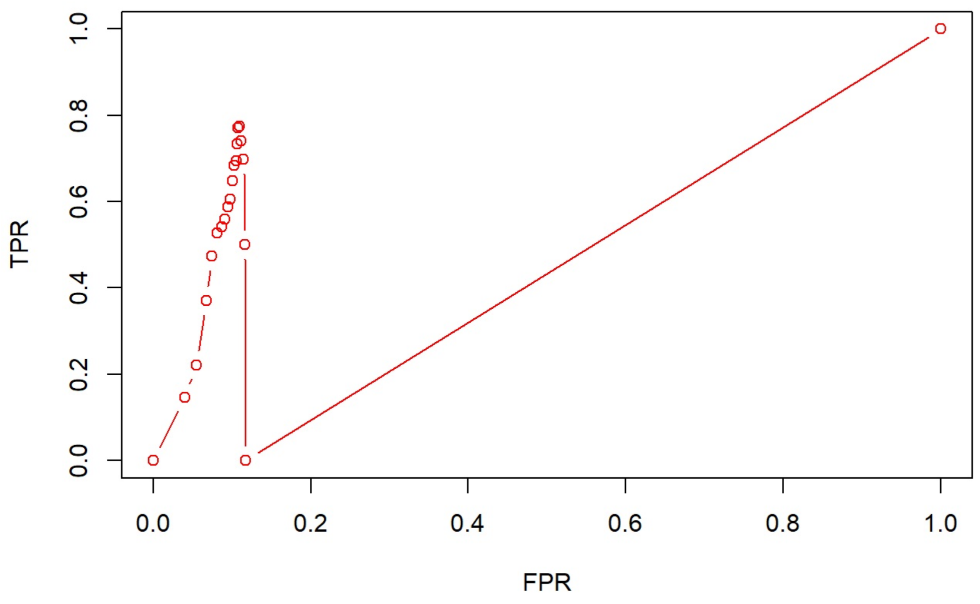
Recall = .88930

F1 = .93959

All of the values has lowered in comparison to those in exercise 4. This is because we used a loss matrix which in turn might have made it harder to predict the values.

Exercise 6





This curve seems to be pretty bad, mainly because the FPR/TPR values does not come close enough to 1 (mainly FPR). Therefore the prediction of the curve fails after FPR~.012

We tried this with the logistic regression as well, which gave the same result.

The precision-recall curve might be a better option if there are more data points available, which would make the plot more accurate.

TASK 3

Exercise 1

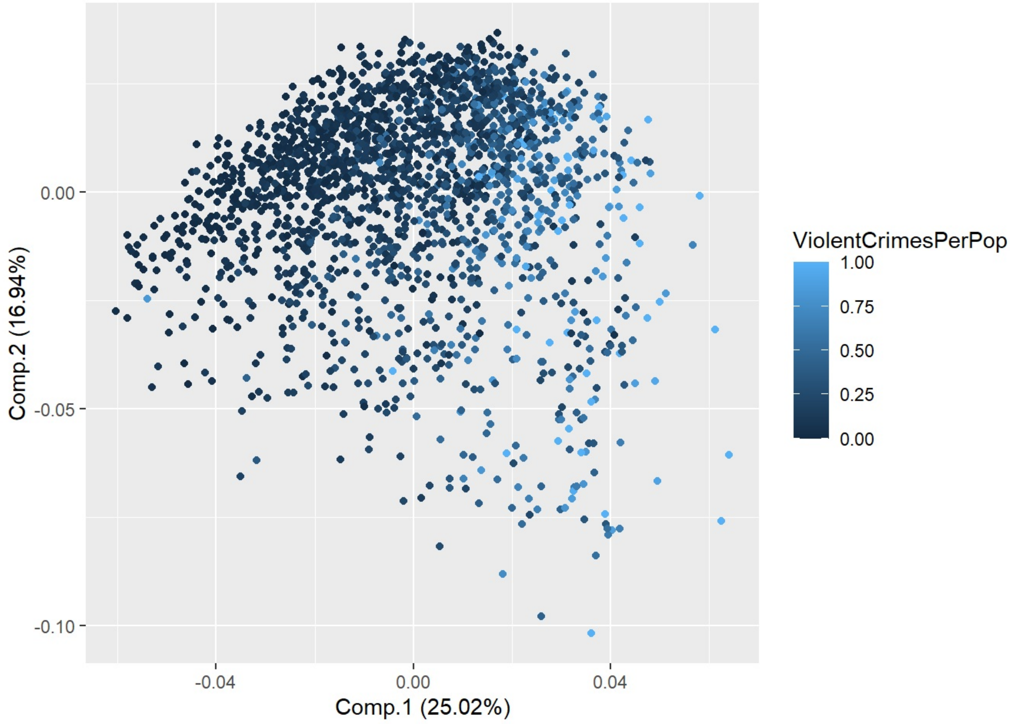
```
## [1] 34
```

```
## [1] 0.2501699 0.1693597
```

95% at 35, 25% and 17% (16.9)

Exercise 2

```
##      medFamInc      medIncome    PctKids2Par     pctWInvInc PctPopUnderPov
##     -0.1833080     -0.1819830     -0.1755423     -0.1748683      0.1737978
```



Yes many features seem to have a relatively big contribution.

The 5 values sound reasonable and should have a logical relationship to the crime level

the area up to left seems both most dense and the darkest, a low pc1 seems to contibute alot toward lower VCPP

Pov1 = 0.2502 Pov2 = 0.1693

Exercise 3

```
## [1] "Train error"
```

```
## [1] 0.2591772
```

```
## [1] "Test error"
```

```
## [1] 0.4000579
```

Compute training and test errors for these data and comment on the quality of model.

The the error for both test and train seems to be high. It is a complex case with a lot of affecting factors so some errors are to be expected. The difference between train and test does not seem to be that big which indicates a relatively well fitted model.

Exercise 4

```
## [1] "calculated optimal train"
```

```
## [1] 0.2592247
```

```
## [1] "Lm train error"
```

```
## [1] 0.2591772
```

```
## [1] "calculated optimal test"
```

```
## [1] 0.3997238
```

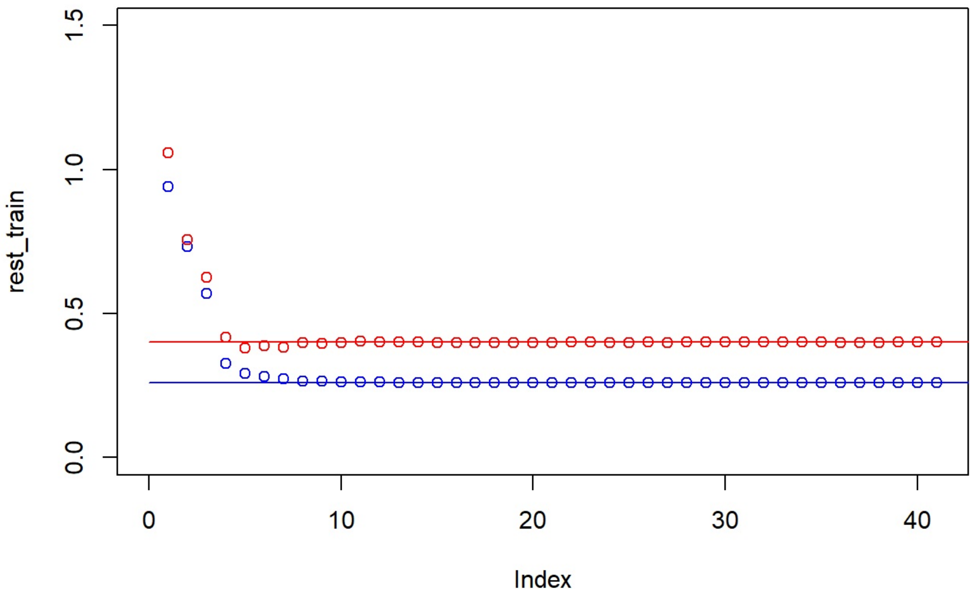
```
## [1] "Lm test error"
```

```
## [1] 0.4000579
```

```
## [1] "Early stopping index and MSE"
```

```
## [1] 2183
```

```
## [1] 0.3769468
```



Min test_error and the early stopping point appears at index 2183 with MSE of 0.377. The results from the 3rd task and the computed optimal values in this exercise are basically the same both in the plot and the printed results. We can also see that the test error as indicated by the early stopping point did reach a lower error rate at an earlier theta but this is with a higher train error.

```
## ----setup,
include=FALSE----------------------------------------------------------
knitr::opts_chunk$set(echo = TRUE)
library(glmnet)
library(tree)
library(caret)
library(dplyr)



## ----
echo=FALSE------------------------------------------------------------------
tecator = read.csv("tecator.csv", header = T)
n = dim(tecator)[1]
set.seed(12345)
df = data.frame(tecator[c(2:102)])

id=sample(1:n, floor(n*0.5))
train = df[id,]
test = df[-id,]

fit = lm(Fat~ ., data = train)
train_preds = predict(fit, train)
test_preds = predict(fit, test)
sum = summary(fit)

MSE_train=mean((train_preds - train$Fat)^2)
MSE_test=mean((test_preds - test$Fat)^2)

print("Test error")
MSE_test
print("Train error")
MSE_train


## ---- dev='png',warning=FALSE,
echo=FALSE-----------------------------------------

y = train$Fat
x = train[1:100]
model_lasso= glmnet(as.matrix(x), as.matrix(y), alpha=1,family="gaussian")

plot(model_lasso, xvar = "lambda")
ynew=predict(model_lasso, newx=as.matrix(x), type="response")


## ----
echo=FALSE------------------------------------------------------------------

y = train$Fat
x = train[1:100]
model_lasso= glmnet(as.matrix(x), as.matrix(y), alpha=0,family="gaussian")

plot(model_lasso, xvar = "lambda")
```

```
ynew=predict(model_lasso, newx=as.matrix(x), type="response")


## ---- warning=FALSE,
echo=FALSE---------------------------------------------------


model_lasso= cv.glmnet(as.matrix(x), as.matrix(y), alpha=1,family="gaussian")


lambda_min = model_lasso$lambda.min
plot(model_lasso, xvar = "lambda")

better_model = glmnet(as.matrix(x), as.matrix(y), lambda = lambda_min, alpha =
1, family = "gaussian")

ynew=predict(better_model, newx=as.matrix(x), s = lambda_min ,
type="response")

plot(y, ylab = "y", col = "red", main = "Scatter plot")
points(ynew, col="blue")


## ---- echo=FALSE,
warning=FALSE---------------------------------------------------

d = read.csv("bank-full.csv", sep = ";", stringsAsFactors = TRUE)
data = d
data$duration = c() #remove duration column
output = d['y']
n = dim(data)[1]

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

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

id3=setdiff(id1,id2)
test=data[id3,]




## ---- echo=FALSE,
warning=FALSE---------------------------------------------------

fit=tree(as.factor(y)~., data=train)
plot(fit)
```

```
text(fit, pretty=0)


fit2=tree(as.factor(y)~., data=train, minsize=7000)
plot(fit2)
text(fit2, pretty=0)


fit3=tree(as.factor(y)~., data=train, mindev=0.0005)
plot(fit3)
text(fit3, pretty=0)




## ---- echo=FALSE,
warning=FALSE----------------------------------------------------
Yfit_t=predict(fit, newdata=train, type="class")
t1<-table(train$y,Yfit_t)
mis_t1 <- 1-sum(diag(t1))/sum(t1)

Yfit_t2=predict(fit2, newdata=train, type="class")
t2<-table(train$y,Yfit_t2)
mis_t2 <- 1-sum(diag(t2))/sum(t2)

Yfit_t3=predict(fit3, newdata=train, type="class")
t3<-table(train$y,Yfit_t3)
mis_t3 <- 1-sum(diag(t3))/sum(t3)

Yfit_v=predict(fit, newdata=valid, type="class")
v1<-table(valid$y,Yfit_v)
mis_v1<-1-sum(diag(v1))/sum(v1)

Yfit_v2=predict(fit2, newdata=valid, type="class")
v2<-table(valid$y,Yfit_v2)
mis_v2<-1-sum(diag(v2))/sum(v2)

Yfit_v3=predict(fit3, newdata=valid, type="class")
v3<-table(valid$y,Yfit_v3)
mis_v3<-1-sum(diag(v3))/sum(v3)


print("1) Training and validation")
print(mis_t1)
print(mis_v1)
print("2) Training and validation")
print(mis_t2)
print(mis_v2)
print("3) Training and validation")
print(mis_t3)
print(mis_v3)
```

```
## ---- echo=FALSE,
warning=FALSE---------------------------------------------------
trainScore=rep(0,50)
testScore=rep(0,50)
for(i in 2:50) {
  prunedTree=prune.tree(fit3,best=i)
  pred=predict(prunedTree, newdata=valid, type="tree")
  trainScore[i]=deviance(prunedTree)
  testScore[i]=deviance(pred)
}
plot(2:50, trainScore[2:50], type="b", col="red", ylim=c(min(testScore[-1]),
max(trainScore[-1])))
points(2:50, testScore[2:50], type="b", col="blue")
print(which.min(testScore[2:50]))

finalTree=prune.tree(fit3, best=20)
finalfit=predict(finalTree, newdata=valid, type="class")
tab = table(valid$y,finalfit)
plot(finalTree)
#text(fit3, pretty=0)



## ---- echo=FALSE,
warning=FALSE---------------------------------------------------


ffitTest<-predict(finalTree, newdata=train, type="class")
conf_mat = confusionMatrix(train$y,ffitTest, mode="everything")



## ---- echo=FALSE,
warning=FALSE---------------------------------------------------



tree5 <- tree(as.factor(y)~., data=train, mindev=0.0005)
predtree5 <- predict(tree5, newdata=test, type="vector")
#L = matrix(c(0,5,1,0), nrow=2, byrow=T)
#probY=predict(tree5, type="response")
probY <- predtree5[,2]
probN <- predtree5[,1]

pred5 <- ifelse(probY/probN>5, "yes", "no")

tab <- table(test$y, pred5)
conf2_mat = confusionMatrix(test$y,as.factor(pred5), mode="everything")
```

```
## ---- echo=FALSE,
warning=FALSE-----------------------------------------------------
optimalTree <- tree(as.factor(y)~., data=train, mindev=0.0005)
optimalTree <- prune.tree(optimalTree, best=20)

pi <- seq(0.05, 0.95, 0.05)

logic_model <- glm(as.factor(y)~.,data = train  ,family="binomial")
pred6_probY = predict(logic_model, newdata = test, type = "response")
pred6_probN = 1 -pred6_probY

tree_pred = predict(optimalTree, newdata = test, type = "vector")

fpr_1 <- c(1:length(pi))
tpr_1 <- c(1:length(pi))
fpr_2 <- c(1:length(pi))
tpr_2 <- c(1:length(pi))


for (i in 1:length(pi)){

  #Tree
  tpr_1[i] = 0
  fpr_1[i] = 0
  pred6 <- ifelse(tree_pred[,2]>pi[i],  "yes", "no")
  pred6_matrix <- table(pred6, test$y)


  #Logistic regression#
  tpr_2[i] = 0
  fpr_2[i] = 0
  pred6_logic <- ifelse(pred6_probY > pi[i], "yes", "no")
  pred6_logic_matrix <- table(pred6_logic, test$y)

  tpr_2[i] <- pred6_logic_matrix[2,2] / (pred6_logic_matrix[2,1]
+pred6_logic_matrix[2,2])
  fpr_2[i] <- (pred6_logic_matrix[1,2] / (pred6_logic_matrix[1,1]
+pred6_logic_matrix[1,2]))
  if(nrow(pred6_matrix) > 1){
    tpr_1[i] <- pred6_matrix[2,2] / (pred6_matrix[2,1]+pred6_matrix[2,2])
    fpr_1[i] <- (pred6_matrix[1,2] / (pred6_matrix[1,1]+pred6_matrix[1,2]))

  }else {
    fpr_1[i] <- (pred6_matrix[1,2] / (pred6_matrix[1,1])) #No values for tpr
  }

}

cut_fpr = fpr_1[1:15]
cut_tpr = tpr_1[1:15]
plot(c(0, fpr_1, 1),c(0, tpr_1, 1), type='b',xlim = c(0,1),
     xlab='FPR', ylab='TPR', col='red')

plot(c(0, fpr_2, 1),c(0, tpr_2, 1), type='b',xlim = c(0,1),
```

```
     xlab='FPR', ylab='TPR', col='red')



## ----echo=FALSE,
warning=FALSE-------------------------------------------------------
rm(list = ls(all = TRUE))
graphics.off()
shell("cls")

data = read.csv(file = "communities.csv",
                header = TRUE)

index <- names(data) %in% "ViolentCrimesPerPop"

data.scaled <- scale(x = data[, !index],
                     center = TRUE,
                     scale = TRUE)

e = eigen(cov(data[, -1]))

e.scaled = eigen(cov(data.scaled))

cum_var = cumsum(e.scaled$values/sum(e.scaled$values))
sum(cum_var<0.95)
e.scaled$values[1:2]/sum(e.scaled$values)




## ---- warning=FALSE,
echo=FALSE---------------------------------------------------
data = read.csv(file = "communities.csv",
                header = TRUE)

index <- names(data) %in% "ViolentCrimesPerPop"

data.scaled <- scale(x = data[, !index],
                     center = TRUE,
                     scale = TRUE)
pr=princomp(data.scaled)

#eigenvalues
lambda=pr$sdev^2


#proportion of variation
var = sprintf("%2.3f",lambda/sum(lambda)*100)


ev1 = pr$loadings[,1]

ev1[order(abs(ev1),decreasing = TRUE)[1:5]]
```

```
library(ggfortify)

autoplot(pr, data = data, colour = "ViolentCrimesPerPop")



## ----
echo=FALSE----------------------------------------------------------------
df = read.csv("communities.csv") #reload the data.

#scale and split 50/50
df = scale(df, TRUE, TRUE)
set.seed(12345)

n <- dim(df)[1]
id <- sample(1:n,floor(n*0.5))
df_train <- data.frame(df[id,])
df_test <- data.frame(df[-id,])


lr = lm(ViolentCrimesPerPop ~ .,df_train)

train.pred = predict(lr, df_train)
test.pred = predict(lr, df_test)

train_MSE = mean((train.pred - df_train$ViolentCrimesPerPop) ^ 2)
test_MSE = mean((test.pred - df_test$ViolentCrimesPerPop) ^ 2)

print("Train error")
train_MSE
print("Test error")
test_MSE



## ----
echo=FALSE----------------------------------------------------------------
train_error <<- numeric(0)
test_error <<-numeric(0)

set.seed(12345)

cost <- function(theta, train, acc_train, test, acc_test){
  pred_train = train %*% theta
  pred_test = test %*% theta


  mse_train = mean((acc_train-pred_train)^2)
  train_error <<- append(train_error,mse_train)

  mse_test = mean((acc_test - pred_test)^2)
  test_error <<- append(test_error,mse_test)
```

```
  return(mse_train)
}



trainy = as.matrix(df_train[,1:(dim(df_train)[2]-1)])
acc_train = as.matrix(df_train['ViolentCrimesPerPop'])

testy = as.matrix(df_test[,1:(dim(df_test)[2]-1)])
acc_test = as.matrix(df_test['ViolentCrimesPerPop'])



theta =numeric(dim(trainy)[2])
theta = as.matrix(theta)


opt = optim(par=theta,fn=cost, train = trainy, acc_train = acc_train,
test=testy, acc_test=acc_test,method = "BFGS")

opt_theta = opt$par

train_opt_error = opt$value

test_opt_error = mean((acc_test - (testy %*% opt_theta))^2)

print("calculated optimal train")
train_opt_error
print("Lm train error")
train_MSE

print("calculated optimal test")
test_opt_error
print("Lm test error")
test_MSE

excluded = c(TRUE,rep(FALSE,500))

rest_train = train_error[excluded]
rest_test = test_error[excluded]

test_min_ind = which(test_error==min(test_error))

print("Early stopping index and MSE")
test_min_ind
min(test_error)

plot(rest_train, xlim=c(0,length(rest_train)), ylim=c(0,1.5), col = "blue")
points(rest_test, col="red")
lines(c(0,1000), rep(train_MSE, 2), col="blue")
lines(c(0,1000), rep(test_MSE, 2), col="red")
```

Statement of Contribution

We divided the initial work into three where Linus Rundin started the first task, Matthias Gerdin the second and Simon Ågren the third. Discussions were then held to assist each other and explain the concepts. Comments were made mostly by the author but also in collaboration with the other members.