Computer Lab 2 Block 1 - Machine Learning | 732A99

Group A-20

Adesijibomi Aderinto | Simon Alsén | Mohamed Ali

12/03/2021

Question 1

Explicit regularization

Assissment 1 Assume that Fat can be modeled as a linear regression in which absorbency characteristics (Channels) are used as features. Report the underlying probabilistic model, fit the linear regression to the training data and estimate the training and test errors.Comment on the quality of fit and prediction and therefore on the quality of model?

Solution
Underlying probabilistic model:

$$p(\mathbf{y}_n|\beta,\sigma^2) = N(\mathbf{y}_n|\beta^T\mathbf{x}_n,\sigma^2)$$

```
#Divide data randomly into train and test (50/50)
tecator_s<-as.data.frame(scale(tecator[2:104]))
tecator_s$Sample<-tecator$Sample
n=dim(tecator_s)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=tecator_s[id,]
id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.5))
test=tecator_s[id2,]
####
lm<-lm(Fat~.-Protein-Moisture-Sample,train)
print.sum2(summary(lm))
```

```
##
## Call:
## lm(formula = Fat ~ . - Protein - Moisture - Sample, data = train)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -0.0158160 -0.0032428 -0.0000817  0.0029541  0.0147453
##
## Residual standard error: 0.02505 on 6 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:  0.9994
## F-statistic:  1651 on 100 and 6 DF,  p-value: 1.058e-09
```

```
mse_tran<- mean((lm$residuals)^2)
pred<-predict(lm,test)
mse_test<- mean((test$Fat-pred)^2)
mse<-data.frame(MSE_Test=mse_test,MSE_Train=mse_tran)
mse
```

```
##   MSE_Test    MSE_Train
## 1 4.491604 3.517304e-05
```

Comment
Although the training MSE should be lower than our test MSE because you we are optimizing for a low training MSE whereas our test MSE is calculated over data unseen during the optimization, the results shows a signification differences between the train MSE and test MSE and since we have test MES is relatively poor than the training score then it's the problem of over-fitting in which we can conclude that the quality of the model would be not sufficient.Thus we can use one of the regularization techniques such as Rigde and LASSO to over come model complexity.

Assissment 2 Assume now that Fat can be modeled as a LASSO regression in which all Channels are used as features. Report the cost function that should be optimized in this scenario.

Solution

With $L1$ regularization LASSO (Least Absolute Shrinkage and Selection Operator),we add penalty term $||\theta||_1$ to the cost function in which the final cost function to minimize can be formlized as:

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

Where $||\theta||_1$:

$$||\theta||_1 = |\theta_0| + |\theta_1| + |\theta_2| + ... + |\theta_p|$$

In another words we can represent the formula as:

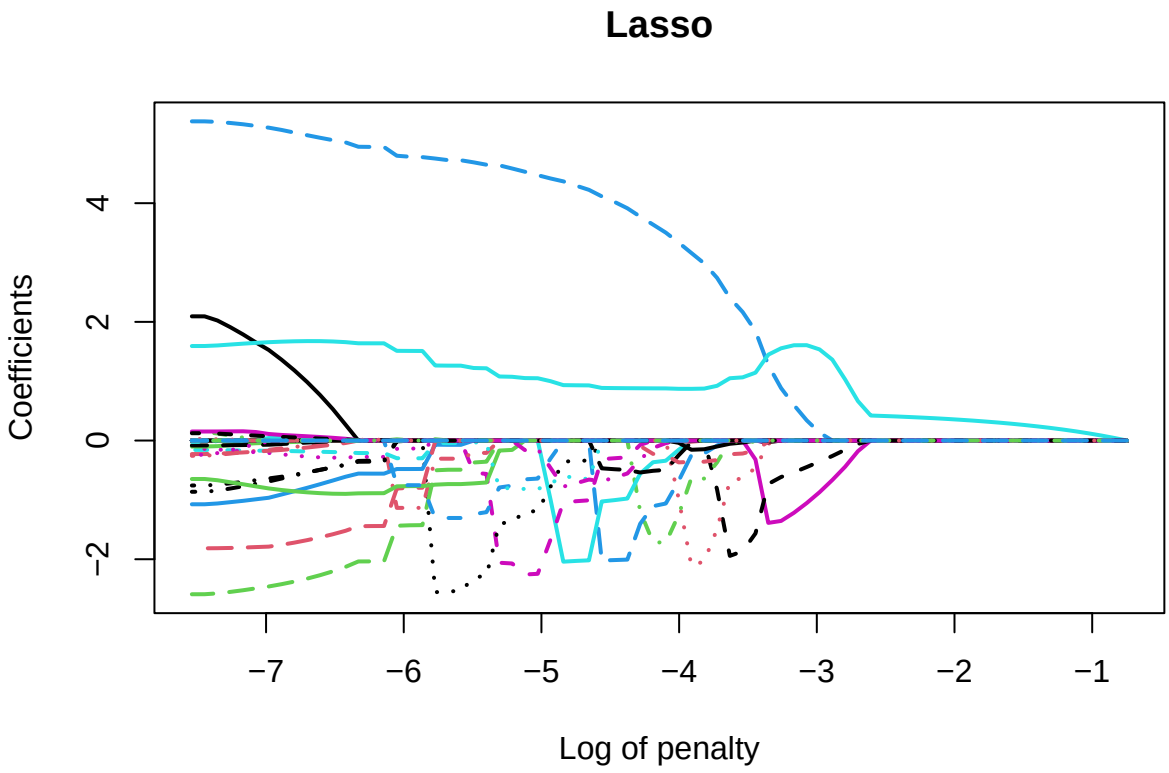
$$\sum_{i=1}^{n}(y_i - \theta_0 - \sum_{j=1}^{p}\theta_j X_{ij})^2 + \lambda\sum_{j=1}^{p}|\theta|$$

Assissment 3

Fit the LASSO regression model to the training data. Present a plot illustrating how the regression coefficients depend on the log of penalty factor log-lambda and interpret this plot. What value of the penalty factor can be chosen if we want to select a model with only three features?

Solution
figure 1.3

```
y <- train$Fat
x <- data.matrix(train[,1:100])


# lasso
la.eq<-glmnet(x, y, family="gaussian",
                intercept = F, alpha=1)
# plot
matplot(log(la.eq$lambda), t(la.eq$beta),
              type="l", main="Lasso", lwd=2,xlab = "Log of penalty",ylab = "Coefficients")
```



Comment
The above plot illustrating how the regression coefficients depend on the log of penalty factor $log\ \lambda$, the $x - axis$ represent the $log\ \lambda$ values and the $y - axis$ represent the coefficients values each colored line represents the value taken by a different coefficient in your model, while lambda represent our $LASSO$ regularization (or penalty) term. $LASSO$ model tends to remove features from the model by shrinking the coefficients completely to zero. The penalty term affects the model coefficients selection in which, when lambda is very small, the $LASSO$ solution should be very close to the OLS solution, and all of the model features are included. In contrast as lambda grows, the regularization term has greater effect and thus we'll observe less features in our model. We can observe clearly this result in our graph as $log\ \lambda$ grows (i.e greater than -4) the model coefficients are decreased.

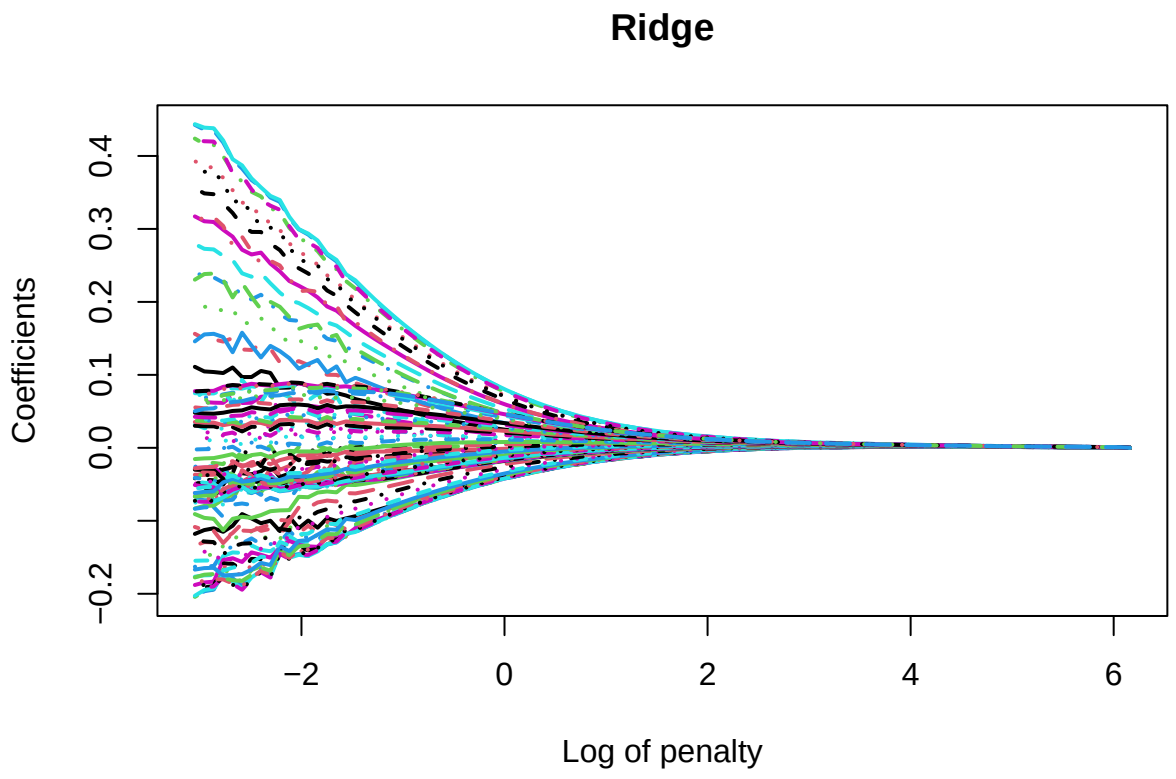
What value of the penalty factor can be chosen if we want to select a model with only three features?
By interpreting the graph visually, we can see that as the value of lambda approaches to zero we observe lees features in our model, the approximate penalty factor that can be chosen if we want to select a model with only three features could be $log\ \lambda\ = -3$.

Assissment 4 Repeat step 3 but fit Ridge instead of the LASSO regression and compare the plots from steps 3 and 4 Conclusions?

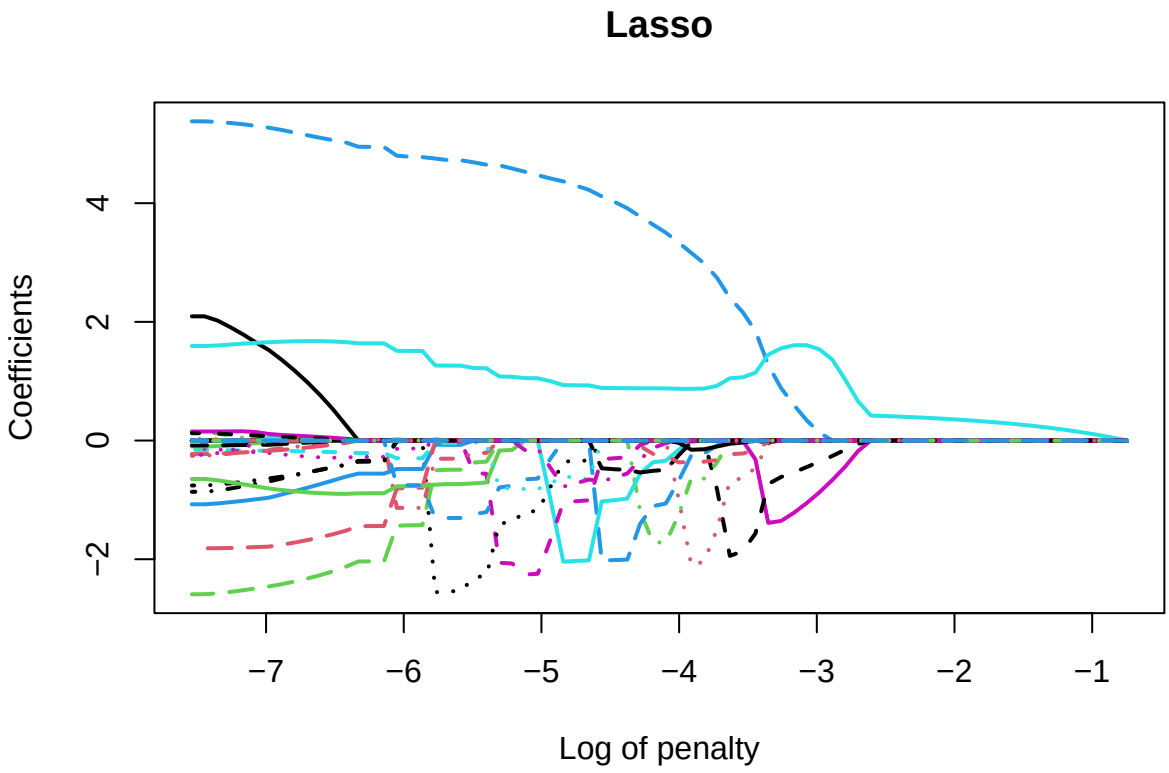
Solution
figure 1.4

```
# Ridge
re.eq<-glmnet(x, y, family="gaussian",
              intercept = F, alpha=0)
# plot
```

```
matplot(log(re.eq$lambda), t(re.eq$beta),
              type="l", main="Ridge", lwd=2,xlab = "Log of penalty",ylab = "Coefficients")
```



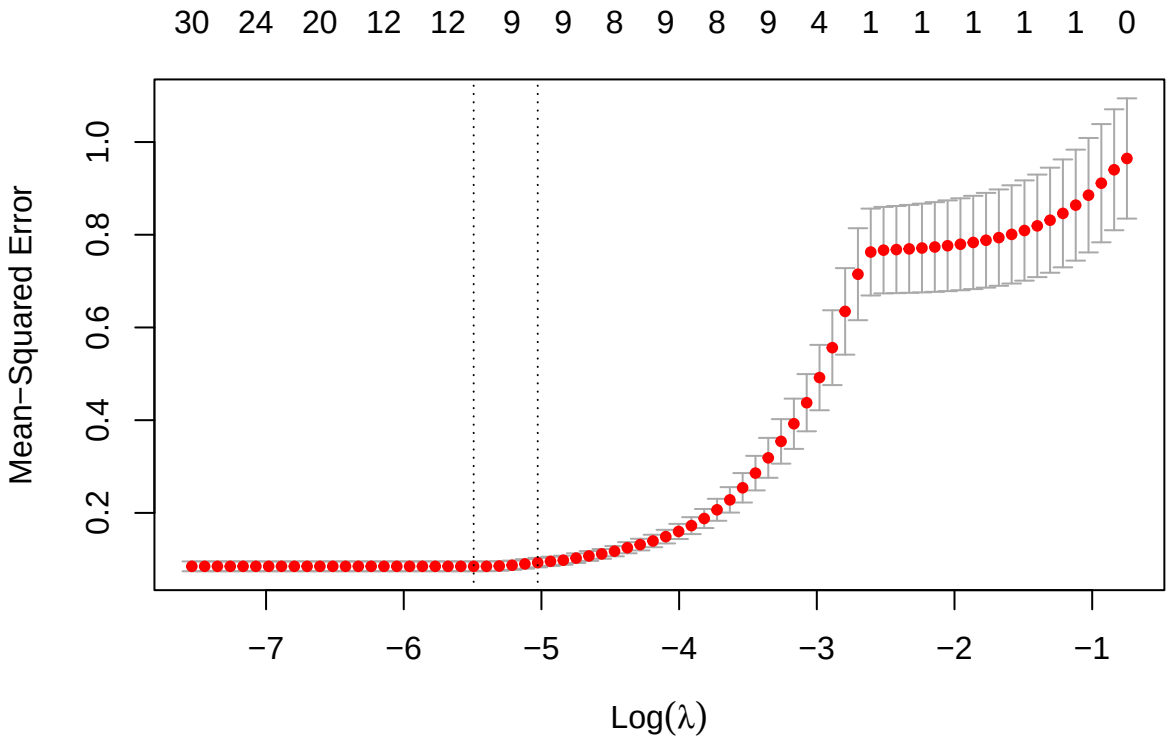

```
matplot(log(la.eq$lambda), t(la.eq$beta),
              type="l", main="Lasso", lwd=2,xlab = "Log of penalty",ylab = "Coefficients")
```



Conclusions?
The above two plots illustrate regression coefficients depend on the log of penalty factor *log* λ, the fist plot (on the above) shows *Ridge regression* used as model to training data-set, while the second plot shows *LASSO* used as model to training data-set. In both plots the $x - axis$ represent the *log* λ values and the $y - axis$ represent the coefficients values each colored line represents the value taken by a different coefficient in our model. The ridge model tends to penalize our coefficients, such that coefficients are the least effective in our estimation will *shrink* the fastest, it's obvious that all the model coefficients shrink in the same manner as lambda reach 6, however an adequate interpretation for the graph and the choose of the lambda is a difficult task to be done visually as we need to consult *MSE vs lambda plot* to give us a better understanding on how to analyze the results. On the other hand, the *LASSO* model shows clearly how different model coefficient depend on the value of the *log* λ, and which coefficient are less important, in addition to the approximate the approximate penalty factor that can be chosen as we showed in *figure 1.3*

Assissment 5 Use cross-validation with default number of folds to compute the optimal LASSO model. Present a plot showing the dependence of the CV score on log-lambda and comment how the CV score changes with log-lambda. Report the optimal lambda and how many variables were chosen in this model. Does the information displayed in the plot suggests that the optimal lambda value results in a statistically significantly better prediction than log-lambda = -4?

```
mod_cv<-cv.glmnet(x=x, y=y, family="gaussian",
                  intercept = F, alpha=1)


plot(mod_cv)
```




```
#coef(mod_cv, c(mod_cv$lambda.min,
 #               mod_cv$lambda.1se))
print(paste(mod_cv$lambda.min,
            log(mod_cv$lambda.min)))
```

```
## [1] "0.00411853285849963 -5.4922582813461"
```

```
print(paste(mod_cv$lambda.1se,
            log(mod_cv$lambda.1se)))
```

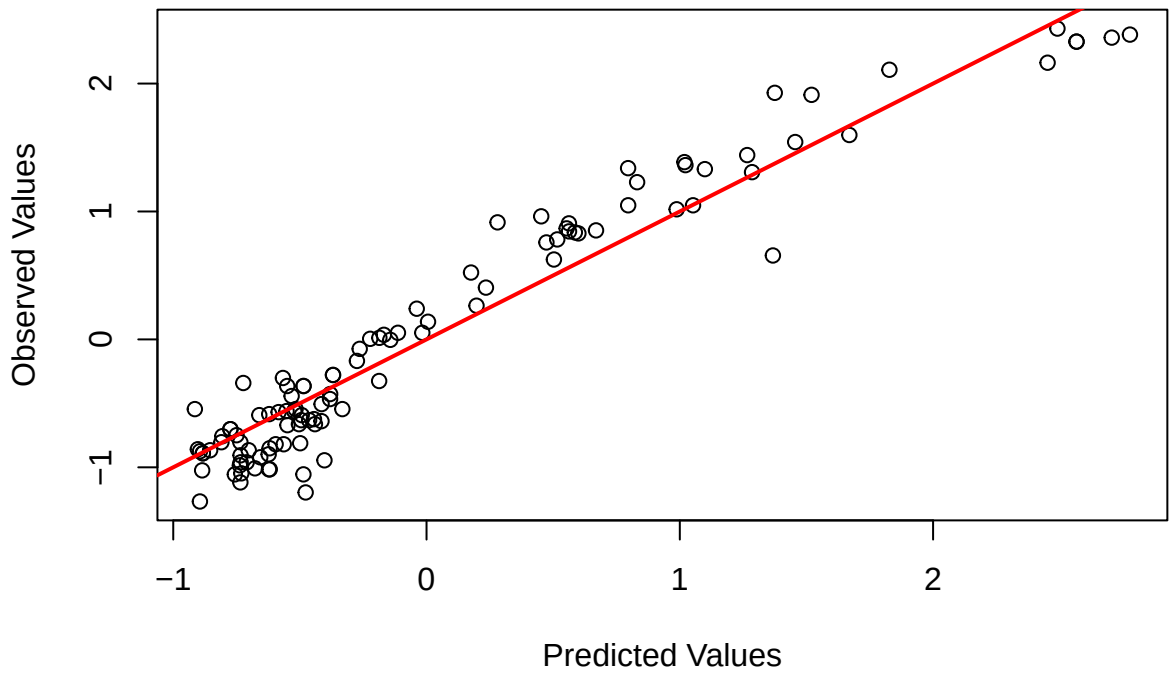
```
## [1] "0.00655786900439887 -5.02708957569074"
```

Comment

The above graph shows dependence of the CV score on Log-lambda, the x-axis represents the value of Log-lambda and the y-axis represent the Mean Square Error, while the numbers on top of the figure shows the number of non-zero coefficients. The CV score change with respect to the value of log-lambda, we can observe that as the Log-lambda increase the value of the Mean Square Error increases, while in contrast the number of coefficients decrease (the top line numbers). The plot shows that the at lambda equals to -4.9 and -5.3 we can end up with almost 9 variables selected into our model. The plot suggests that the optimal log-lambda value in between -5 and -6 in which we can approximate it at -5.3, however, when we use lambda.1se (which defined as largest value of lambda such that error is within 1 standard error of the minimum) from cv.glmnet() we observe that lambda is equal to -5.39922454021503, which is to some extend close to the value from the plot.

Finally, create a scatter plot of the original test versus predicted test values for the model corresponding to optimal lambda and comment whether the model predictions are good

```r
y <- test$Fat
x_new <- data.matrix(test[,1:100])
best_lambda<- mod_cv$lambda.min
best_model <- glmnet(x_new, y, alpha = 1, lambda = best_lambda)
orig<- glmnet(x, y, alpha = 1)

best_predict<-predict(best_model, s = best_lambda, newx = x_new)

plot(best_predict,y,xlab = "Predicted Values",ylab = "Observed Values")
abline(a = 0,  b = 1,col = "red",lwd = 2)
```



Comment The above scatter blot shows the predicted values of the Fat vs the Actual values, from the scatter plot we can observe that the values of Fat (Predicted vs Actual) are quite related, except for some outlines at the upper end and the middle of the graph, this mean that the used model (LASSO with optimal lambda) can be interpreted as good model for prediction.

Assignment 2

2.1

```r
data <- read.csv("bank-full.csv", sep = ";")

data <- data[,-12]

data[,c(2:5, 7:9, 11, 15:16)] <- lapply(data[,c(2:5, 7:9, 11, 15:16)], as.factor)
```

```
data$y <- factor(data$y, levels = c("yes", "no"))

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,]
```

2.2

```
fit_def <- tree(y ~ ., data = train)
fit_minsize <- tree(y ~ ., data = train, control = tree.control(nobs = nrow(train),
                                                                minsize = 7000))
fit_mindev <- tree(y ~ ., data = train, control = tree.control(nobs = nrow(train),
                                                               mindev = 0.0005))

# a. Decision Tree with default settings (train)
misclass_train_def <- sum(predict(fit_def, newdata = train, type = "class")
                          != train$y) / nrow(train)

# a. Decision Tree with default settings (validation)
misclass_val_def <- sum(predict(fit_def, newdata = valid, type = "class")
                        != valid$y) / nrow(valid)

# b. Decision Tree with smallest allowed node size equal to 7000 (train)
misclass_train_minsize <- sum(predict(fit_minsize, newdata = train, type = "class")
                              != train$y) / nrow(train)

# b. Decision Tree with smallest allowed node size equal to 7000 (validation)
misclass_val_minsize <- sum(predict(fit_minsize, newdata = valid, type = "class")
                            != valid$y) / nrow(valid)

# c. Decision trees minimum deviance to 0.0005 (train)
misclass_train_mindev <- sum(predict(fit_mindev, newdata = train, type = "class")
                             != train$y) / nrow(train)

# c. Decision trees minimum deviance to 0.0005 (validation)
misclass_val_mindev <- sum(predict(fit_mindev, newdata = valid, type = "class")
                           != valid$y) / nrow(valid)

misclass_df <- data.frame(Data = c("Train", "Validation"),
                          Default = c(misclass_train_def, misclass_val_def),
                          Min.size = c(misclass_train_minsize, misclass_val_minsize),
                          Min.dev = c(misclass_train_mindev, misclass_val_mindev))
misclass_df
```

```
##         Data   Default  Min.size    Min.dev
## 1      Train 0.1048441 0.1048441 0.09323159
```

```
## 2 Validation 0.1092679 0.1092679 0.11162722
```

```
size_df <- data.frame(Tree = c("Default", "Min.size", "Min.dev"),
                      Size = c(nrow(fit_def$frame), nrow(fit_minsize$frame),
                               nrow(fit_mindev$frame)))
size_df
```

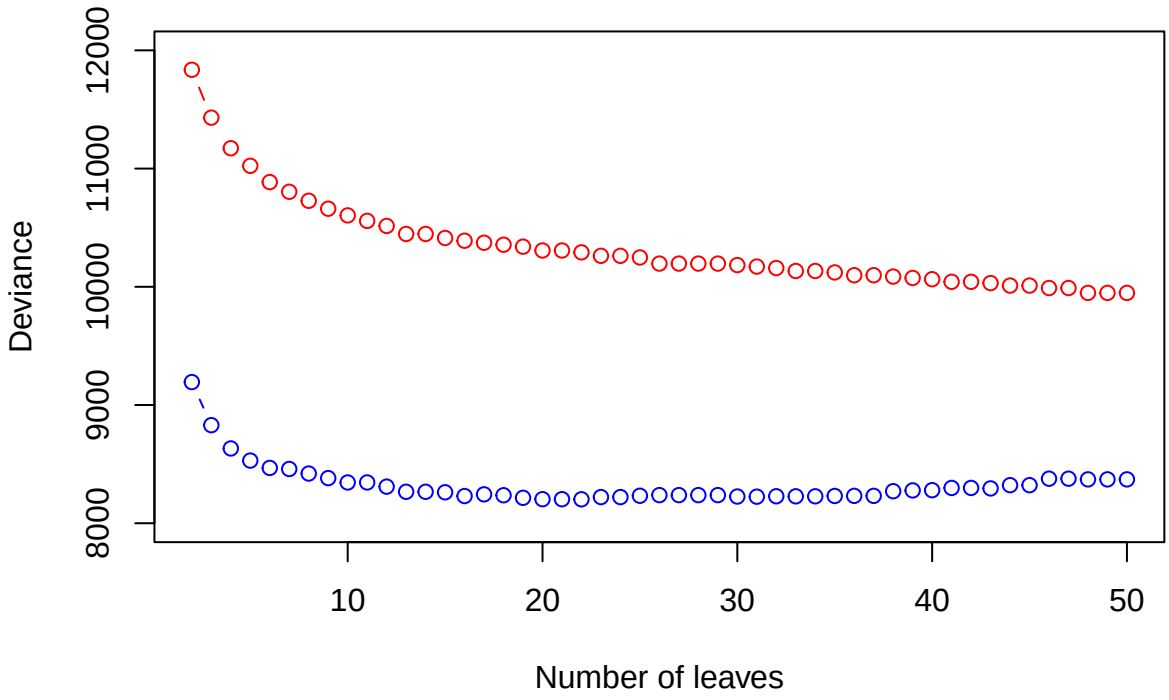
```
##       Tree Size
## 1  Default   11
## 2 Min.size    9
## 3  Min.dev  243
```

The decision tree with minimum deviance set to 0.0005 gives the lowest misclassification error for the test data, but the highest misclassification error for the validation data. This is most likely a result of overfitting. The tree with default settings and the tree with the minimum node size set to 7000 gives the lowest misclassification error for the validation data.

The tree with minimum node size set to 7000 results in a slightly smaller tree compared to when default settings are being used. Despite the tree with minimum node size set to 7000 being smaller than the tree with default settings, it does not have a lower accuracy. The tree with minimum deviance set to 0.0005 results in by far the biggest tree, which can lead to overfitting and thereby explain why the tree with minimum deviance set to 0.0005 gives a lower misclassification rate for the training data, but a higher misclassification rate for the validation data.

2.3

```
trainScore <- rep(0, 50)
testScore <- rep(0, 50)

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

plot(2:length(trainScore), trainScore[2:length(trainScore)], type = "b",
     col = "red", ylim = c(8000, 12000), xlab = "Number of leaves", ylab = "Deviance")
points(2:length(testScore), testScore[2:length(testScore)], type = "b", col = "blue")
```




```
# Optimal amount of leaves
opt_leaves <- which.min(testScore[2:length(testScore)])
opt_leaves
```

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

```
fit_opt <- prune.tree(fit_mindev, best = opt_leaves)
```

The plot shows that the deviance for the training data decreases as the number of leaves increases. However, the deviance for the validation data is only decreasing until about 20 leaves, after which the deviance slowly start increasing. As the number of leaves increases, the bias gets smaller but the variance increases which results in overfitting.

21 leaves results in the lowest deviance for the validation data, and 21 leaves can therefore be considered to be the optimal amount of leaves.

The tree uses deviance as the splitting criterion, and the tree is first split by the variable *poutcome* which has the highest deviance. This means that *poutcome* is best at separating the classes, and can be considered to be the most important variable. The second and third most important variables are *pdays* and *month*.

If the outcome of the previous marketing campaign (*poutcome*) was success it is more likely that the client subscribed a term deposit. If the *poutcome* is not success, last contact month of year (*month*) is not december, mars, october, september, april, february or july, contact communication type (*contact*) is cellular or telephone, and number of days that passed by after the client was last contacted from a previous campaign (*pdays*) was 383.5 days or more it is also more likely that the client subscribed a term deposit. In other cases, it is not likely that the client subscribed a term deposit.

2.4

```
pred <- predict(fit_opt, newdata = test, type = "class")

# Confusion matrix
cm <- table(test$y, pred)
cm
```

```
##      pred
##         yes    no
##   yes   291  1294
##   no    167 11812
```

```
# Accuracy
acc <- (cm[1] + cm[4]) / sum(cm)
acc
```

```
## [1] 0.8922884
```

```
# Precision
prec <- cm[1] / (cm[1] + cm[2])

# Recall
rec <- cm[1] / (cm[1] + cm[3])

# F1 score
f1 <- (2 * prec * rec) / (prec + rec)
f1
```

```
## [1] 0.2848752
```

The overall accuracy is 0.892, meaning that 89.2% of the predictions are correct. However, the accuracy is much higher for class *no*. 98.6% of all observations belonging to class *no* are correct, while only 18.4% of the observations belonging to class *yes* is correct.

F1 score is a measure that takes both false positives and false negatives into account, and is thereby the preferred measure to assess the model's predictive power in this particular case. The F1 score is only 0.285 which means that the model does not have a very high predictive power for both the classes.

2.5

```
loss_matrix <- matrix(c(0,1,5,0), 2, 2)

fit_lmat <- rpart(y ~ ., method = "class", data = train, parms = list(loss = loss_matrix))

pred2 <- predict(fit_lmat, newdata = test, type = "class")

# Confusion matrix
cm2 <- table(test$y, pred2)
cm2
```

```
##      pred2
##         yes    no
##   yes   778   807
##   no   1099 10880
```

```
# Accuracy
acc2 <- (cm2[1] + cm2[4]) / sum(cm2)
```

```
acc2
```

```
## [1] 0.859481
```

```
# Precision
prec2 <- cm2[1] / (cm2[1] + cm2[2])

# Recall
rec2 <- cm2[1] / (cm2[1] + cm2[3])

# F1 score
f1_2 <- (2 * prec2 * rec2) / (prec2 + rec2)
f1_2
```

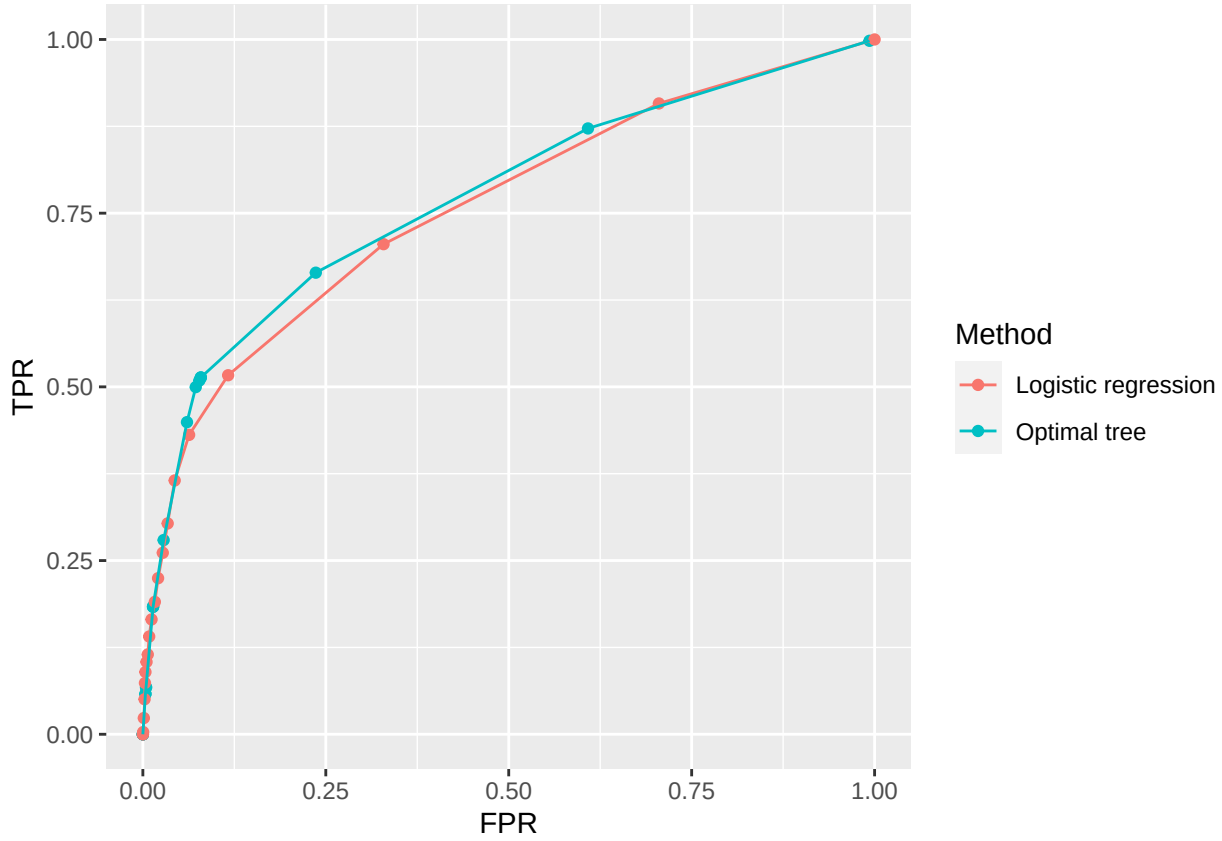
```
## [1] 0.4494512
```

The use of the loss matrix makes the overall accuracy go down, but the F1 score increases. The loss matrix increases the weights of false negatives and false positives, which greatly improves the model's ability to correctly predict observations belonging to class *yes*.

2.6

```
fit_logreg <- glm(y ~ ., data = train, family = "binomial")

thresholds <- seq(0, 1, by = 0.05)

res_tree <- as.data.frame(matrix(NA, nrow = length(thresholds), ncol = 3))
colnames(res_tree) <- c("Threshold", "TPR", "FPR")
res_logreg <- res_tree

for(i in 1:length(thresholds)) {
  pred_tree <- predict(fit_opt, newdata = test)
  tree_class <- ifelse(pred_tree[,1] > thresholds[i], "yes", "no")
  # True positive
  TP_tree <- sum((test$y == tree_class)[which(test$y == "yes")])
  # False positive
  FP_tree <- sum((test$y != tree_class)[which(tree_class == "yes")])
  # True negative
  TN_tree <- sum((test$y == tree_class)[which(test$y == "no")])
  # False negative
  FN_tree <- sum((test$y != tree_class)[which(tree_class == "no")])

  # True positive rate
  TPR_tree <- TP_tree / (FN_tree + TP_tree)
  # False positive rate
  FPR_tree <- FP_tree / (TN_tree + FP_tree)

  res_tree[i,1] <- thresholds[i]
  res_tree[i,2] <- TPR_tree
  res_tree[i,3] <- FPR_tree

  pred_logreg <- predict(fit_logreg, newdata = test, type = "response")
  logreg_class <- ifelse(pred_logreg > thresholds[i], "no", "yes")
  # True positive
  TP_logreg <- sum((test$y == logreg_class)[which(test$y == "yes")])
```

```
  # False positive
  FP_logreg <- sum((test$y != logreg_class)[which(logreg_class == "yes")])
  # True negative
  TN_logreg <- sum((test$y == logreg_class)[which(test$y == "no")])
  # False negative
  FN_logreg <- sum((test$y != logreg_class)[which(logreg_class == "no")])

  # True positive rate
  TPR_logreg <- TP_logreg / (FN_logreg + TP_logreg)
  # False positive rate
  FPR_logreg <- FP_logreg / (TN_logreg + FP_logreg)

  res_logreg[i,1] <- thresholds[i]
  res_logreg[i,2] <- TPR_logreg
  res_logreg[i,3] <- FPR_logreg
}

res_tree$Method <- "Optimal tree"
res_logreg$Method <- "Logistic regression"

res <- rbind(res_tree, res_logreg)

ggplot(res, aes(x = FPR, y = TPR, color = Method)) +
  geom_point() +
  geom_line()
```



According to the ROC curve, the optimal tree model perform slightly better than the logistic regression model. This can potentially be due to the relationship between the response and the features being non-linear.

A precision-recall curve usually gives a more accurate picture of a model's predictive power when the classes are imbalanced, like in this case where only 11.69% of the observations belong to the class *yes*.

Assignment 3

3.1

```
df3 <- read.csv("communities.csv")

df3[,1:100] <- scale(df3[,-c(101)])

cov_mat <- cov(df3[,1:100])

cov_eigen <- eigen(cov_mat)

comp_var <- cov_eigen$values / sum(cov_eigen$values)
comp_var_cum <- cumsum(comp_var)

test <- prcomp(df3[,1:100], center = TRUE, scale = TRUE)

sum(cov_eigen$values[1:35])
```

```
## [1] 95.27018
```

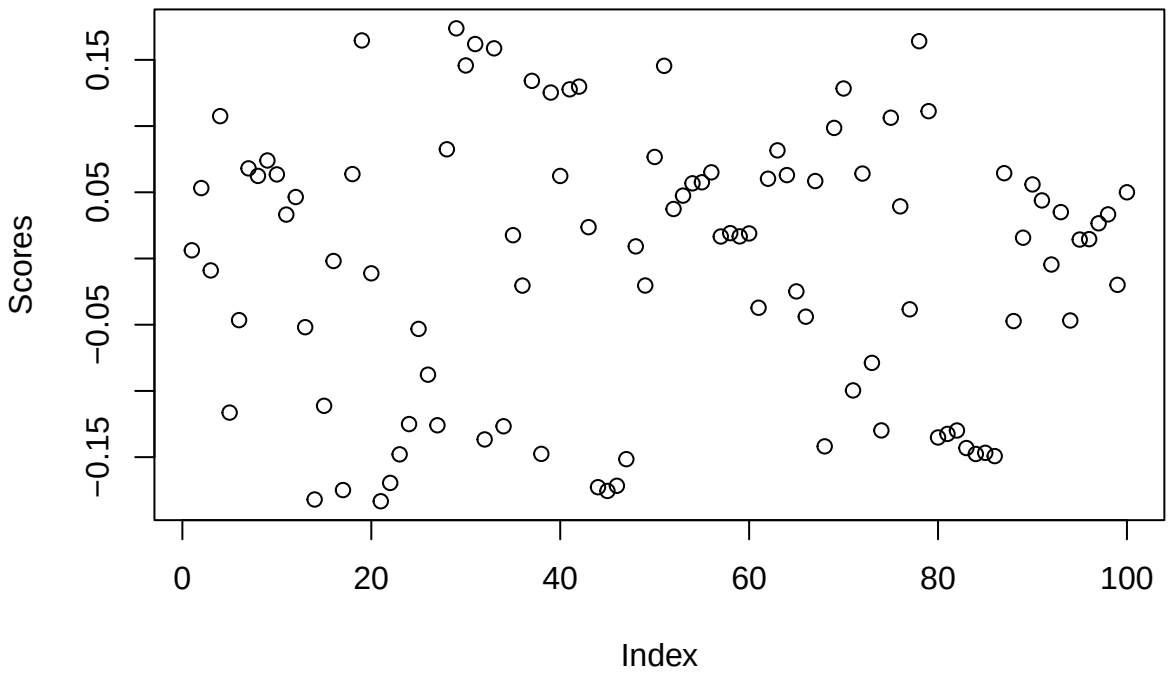
```
cov_eigen$values[1:2]
```

```
## [1] 25.01699 16.93597
```

35 features is needed in order to obtain at least 95% of the variance in the data. The first two principal components explains 25.02% and 16.94% of the variation.

3.2

```
plot(cov_eigen$vectors[,1], ylab = "Scores")
```



It seems like many features have a notable contribution to the first principal component.

```
pca_res <- princomp(df3[,1:100])
# Correlation between PC1 and ViolentCrimesPerPop
cor(pca_res$scores[,1], df3$ViolentCrimesPerPop)
```

```
## [1] 0.6293296
```

```
# The 5 features that contribute most to PC1
colnames(df3)[sort(abs(cov_eigen$vectors[,1]), index.return = TRUE,
                   decreasing = TRUE)$ix[1:5]]
```

```
## [1] "medFamInc"      "medIncome"      "PctKids2Par"    "pctWInvInc"
## [5] "PctPopUnderPov"
```

```
# Scores of the 5 features that contribute most to PC1
cov_eigen$vectors[,1][sort(abs(cov_eigen$vectors[,1]), index.return = TRUE,
                           decreasing = TRUE)$ix[1:5]]
```

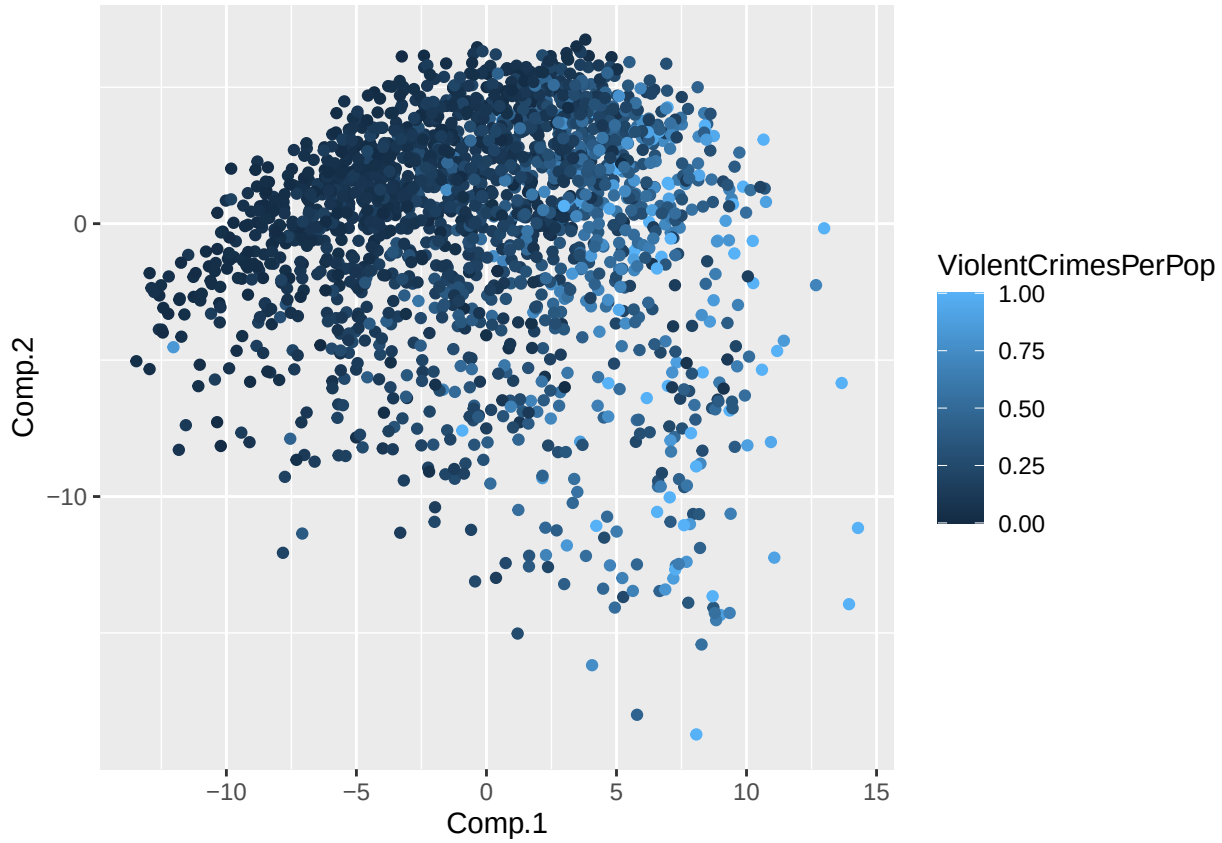
```
## [1] -0.1833080 -0.1819830 -0.1755423 -0.1748683  0.1737978
```

medFamInc, *medIncome*, *PctKids2Par*, *pctWInvInc* and *PctPopUnderPov* are the 5 features that contribute most to the first principal component. PC1 has a strong, positive relationship with crime level.

Median family income (*medFamInc*), median household income (*medIncome*), percentage of kids in family housing with two parents (*PctKids2Par*), and percentage of households with investment / rent income (*pctWInvInc*) have a negative relationship with PC1 which means that locations with low values in these 4 features tend to have higher crime level. Percentage of people under the poverty level (*PctPopUnderPov*) has a positive relationship with PC1 meaning that locations with higher values in this feature tend to have

higher crime levels.

```
# PC scores
df_scores <- as.data.frame(pca_res$scores[,1:2])
df_scores$ViolentCrimesPerPop <- df3$ViolentCrimesPerPop

ggplot(df_scores, aes(x = Comp.1, y = Comp.2, color = ViolentCrimesPerPop)) +
  geom_point()
```



Both PC1 and PC2 seem to have a positive relationship with crime level, meaning that locations with higher scores in the first and second principal component also tend to have a higher crime level.

3.3

```
n <- dim(df3)[1]
set.seed(12345)
id <- sample(1:n, floor(n*0.5))
train <- df3[id,]
test <- df3[-id,]

fit_lm <- lm(ViolentCrimesPerPop ~ .-1, data = train)

# Training MSE
mean((fit_lm$residuals)^2)
```

```
## [1] 0.06687745
```

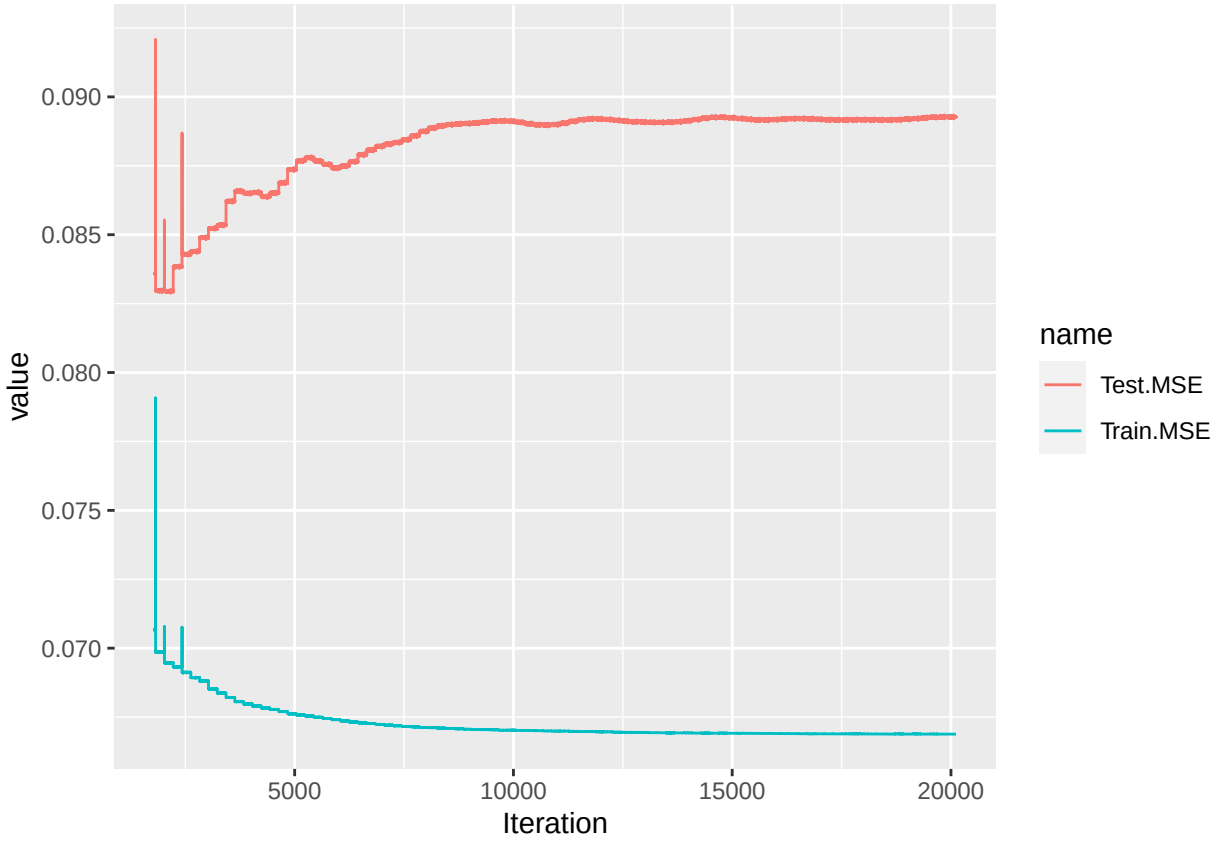
```r
# Test MSE
pred_lm <- predict(fit_lm, newdata = test)
mean((test$ViolentCrimesPerPop - pred_lm)^2)
```

```
## [1] 0.08929595
```

Both the training and test MSE is low, which means that the model has a low predictive power.

3.4

```r
res_train <- c()
res_test <- c()

linreg <- function(par) {
  fitted_values <- as.matrix(train[,-101]) %*% par

  resid_train <- train$ViolentCrimesPerPop - fitted_values
  MSE_train <- mean(resid_train^2)

  pred_test <- as.matrix(test[,-101]) %*% par
  resid_test <- test$ViolentCrimesPerPop - pred_test
  MSE_test <- mean(resid_test^2)

  res_train <<- c(res_train, MSE_train)
  res_test <<- c(res_test, MSE_test)

  return(MSE_train)
}

opt_res <- optim(par = rep(0, ncol(train)-1), fn = linreg, method = "BFGS")

res <- data.frame(Iteration = 1:length(res_train), Train.MSE = res_train,
                  Test.MSE = res_test)

res_long <- pivot_longer(res[-c(1:1800),], cols = c(2:3))

ggplot(res_long, aes(x = Iteration, y = value, color = name)) +
  geom_line()
```




```
# Optimal iteration according to the early stopping criterion
which.min(res$Test.MSE)
```

```
## [1] 2166
```

```
# Training error in optimal model
MSE_train_opt <- res$Train.MSE[which.min(res$Test.MSE)]

# Training MSE from step 3
MSE_train_s3 <- mean((fit_lm$residuals)^2)

# Test error in optimal model
MSE_test_opt <- res$Test.MSE[which.min(res$Test.MSE)]

# Test MSE from step 3
MSE_test_s3 <- mean((test$ViolentCrimesPerPop - pred_lm)^2)

res_df <- data.frame(Optimal.model = c(MSE_train_opt, MSE_test_opt),
                     Step.3.model = c(MSE_train_s3, MSE_test_s3))
rownames(res_df) <- c("Training", "Test")
res_df
```

```
##          Optimal.model Step.3.model
## Training    0.06947240   0.06687745
## Test        0.08288133   0.08929595
```

According to the early stopping criterion, iteration 2166 is the optimal iteration number as the test MSE starts to increase after this iteration. The optimal model results in higher MSE for the training data than

the model from step 3 did. However, the optimal model gives lower MSE for the test data than what the model from step 3 did. This is most likely due to overfitting to the training data.

Statement of Contribution:

Simon, Mohamed and Adesijibomi devised the whole assignment together, the main conceptual ideas and codes outline. Mohamed worked out Assignment 1 (Explicit regularization), Adesijibomi worked out Assignment 2 (Decision trees and logistic regression for bank marketing), Simon , worked out Assignment 3 (Principal components and implicit regularization) and the report creation using r markdown.