Machine Learning Computer Lab 2 (Group A7)

Qinyuan Qi(qinqi464) Satya Sai Naga Jaya Koushik Pilla (satpi345)
Daniele Bozzoli(danbo826)

2024-01-13

Statement of Contribution

For Lab 2, we decided to split the three assignments equally, Qinyuan completed Assignment 1, Satya completed Assignment 2 and Daniele completed Assignment 3, after which, for verification's sake, we completed each other's assignments as well and validated our findings. The report was also compiled by three of us, with each handling their respective assignments.

Assignment 1: Explicit regularization

Answer:

(1)

According to the question, we can create a linear model like the following.

$$Fat = \beta_0 + \beta_1 Channel_1 + \beta_2 Channel_2 + ... + \beta_{100} Channel_{100} + \epsilon = \sum \beta_i Channel_i + \epsilon$$

In the above formula, β_0 is the intercept, while the remaining β are the parameters corresponding to each channel.

According to the output, the model generated uses 100 channel features, and almost all the channels provide contributions to the target. However, the p-value shows that only very limited channels are useful. And the $MSE_{test} = 722.4294$, $MSE_{training} = 0.005709117$. It means the training data fit pretty well, however, the test data fit not as expected, and the model overfits the data.

```
## test_mse is: 722.4294
```

```
## train_mse is: 0.005709117
```

```
##
## Call:
## lm(formula = Fat ~ ., data = train_data_set)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.201500 -0.041315 -0.001041  0.037636  0.187860
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.815e+01  5.488e+00  -3.306  0.01628 *
## Channel1     2.653e+04  1.126e+04   2.357  0.05649 .
## Channel2    -5.871e+04  3.493e+04  -1.681  0.14385
## Channel3     1.154e+05  7.373e+04   1.565  0.16852
## Channel4    -2.432e+05  1.175e+05  -2.070  0.08387 .
## Channel5     3.026e+05  1.193e+05   2.536  0.04430 *
## Channel6    -2.365e+05  8.160e+04  -2.898  0.02741 *
## Channel7     1.090e+05  3.169e+04   3.440  0.01380 *
## Channel8    -6.054e+04  1.508e+04  -4.015  0.00700 **
## Channel9     7.871e+04  2.160e+04   3.643  0.01079 *
## Channel10   -1.730e+04  1.640e+04  -1.055  0.33215
## Channel11    9.562e+04  3.529e+04   2.710  0.03512 *
## Channel12   -2.114e+05  6.198e+04  -3.410  0.01431 *
## Channel13    9.725e+04  4.424e+04   2.198  0.07026 .
## Channel14    5.296e+04  4.666e+04   1.135  0.29968
## Channel15   -7.855e+04  5.245e+04  -1.498  0.18491
## Channel16   -8.209e+03  1.893e+04  -0.434  0.67969
## Channel17    3.769e+04  1.987e+04   1.897  0.10666
## Channel18    3.306e+04  7.934e+03   4.167  0.00590 **
## Channel19   -8.405e+04  1.929e+04  -4.358  0.00478 **
## Channel20    1.510e+05  3.361e+04   4.492  0.00414 **
## Channel21   -2.069e+05  4.256e+04  -4.862  0.00282 **
## Channel22    1.348e+05  3.824e+04   3.526  0.01243 *
## Channel23   -4.094e+04  3.546e+04  -1.154  0.29222
## Channel24    2.023e+04  2.761e+04   0.733  0.49134
## Channel25    3.269e+03  1.071e+04   0.305  0.77045
## Channel26   -1.297e+04  7.636e+03  -1.699  0.14028
## Channel27    4.131e+03  1.422e+04   0.291  0.78120
## Channel28   -4.548e+03  2.988e+04  -0.152  0.88402
## Channel29    1.089e+04  1.768e+04   0.616  0.56072
## Channel30   -7.985e+04  2.653e+04  -3.010  0.02371 *
## Channel31    1.756e+05  5.279e+04   3.326  0.01589 *
## Channel32   -1.107e+05  2.904e+04  -3.813  0.00883 **
## Channel33   -6.525e+04  5.407e+04  -1.207  0.27294
## Channel34    1.007e+05  6.589e+04   1.528  0.17738
## Channel35   -2.841e+03  1.214e+04  -0.234  0.82266
## Channel36   -2.268e+04  2.295e+04  -0.988  0.36127
## Channel37   -4.479e+04  1.292e+04  -3.468  0.01334 *
## Channel38    3.209e+04  1.843e+04   1.742  0.13221
## Channel39    1.992e+04  2.067e+04   0.964  0.37246
## Channel40   -9.833e+03  2.431e+04  -0.404  0.69988
## Channel41    1.659e+04  3.648e+04   0.455  0.66531
## Channel42   -1.829e+04  3.528e+04  -0.519  0.62260
## Channel43   -2.423e+04  2.427e+04  -0.998  0.35669
## Channel44    3.246e+04  2.013e+04   1.613  0.15793
## Channel45   -8.089e+03  4.023e+04  -0.201  0.84728
## Channel46    7.065e+03  2.810e+04   0.251  0.80990
## Channel47   -4.062e+04  1.007e+04  -4.034  0.00685 **
## Channel48    9.080e+04  2.618e+04   3.469  0.01332 *
## Channel49   -6.647e+04  2.372e+04  -2.803  0.03105 *
## Channel50   -4.196e+04  2.856e+04  -1.469  0.19213
## Channel51    1.097e+05  5.572e+04   1.968  0.09661 .
## Channel52   -1.148e+05  6.376e+04  -1.800  0.12196
## Channel53    9.525e+04  7.450e+04   1.278  0.24830
## Channel54   -4.534e+04  7.363e+04  -0.616  0.56067
## Channel55   -1.535e+03  4.933e+04  -0.031  0.97618
## Channel56   -2.377e+03  2.109e+04  -0.113  0.91394
## Channel57    3.174e+04  1.005e+04   3.158  0.01961 *
## Channel58    2.221e+03  1.048e+04   0.212  0.83915
## Channel59   -8.504e+04  2.574e+04  -3.304  0.01634 *
## Channel60    6.382e+04  1.607e+04   3.972  0.00735 **
## Channel61    2.151e+04  1.234e+04   1.742  0.13211
## Channel62   -2.859e+04  1.065e+04  -2.685  0.03631 *
## Channel63    1.796e+04  9.187e+03   1.955  0.09838 .
## Channel64    5.759e+04  3.526e+04   1.633  0.15354
## Channel65   -1.470e+05  6.911e+04  -2.127  0.07752 .
## Channel66    9.121e+04  4.461e+04   2.045  0.08688 .
## Channel67   -5.733e+03  2.197e+04  -0.261  0.80288
## Channel68   -6.290e+04  2.192e+04  -2.870  0.02843 *
## Channel69    6.421e+04  2.074e+04   3.096  0.02121 *
## Channel70   -1.749e+04  1.581e+04  -1.106  0.31111
## Channel71   -7.248e+03  1.934e+04  -0.375  0.72075
## Channel72    3.406e+04  1.185e+04   2.873  0.02830 *
## Channel73   -2.100e+04  1.132e+04  -1.855  0.11308
## Channel74   -3.314e+04  1.220e+04  -2.717  0.03480 *
## Channel75    7.039e+04  2.054e+04   3.427  0.01402 *
## Channel76   -3.187e+04  1.736e+04  -1.836  0.11597
## Channel77    2.061e+04  1.810e+04   1.138  0.29832
## Channel78   -1.180e+04  2.273e+04  -0.519  0.62225
## Channel79    2.669e+04  2.997e+04   0.890  0.40750
## Channel80   -6.051e+04  1.483e+04  -4.080  0.00650 **
## Channel81    1.386e+03  2.628e+04   0.053  0.95966
## Channel82    1.020e+05  4.694e+04   2.173  0.07275 .
## Channel83   -1.706e+05  4.688e+04  -3.640  0.01083 *
## Channel84    1.097e+05  2.892e+04   3.792  0.00905 **
## Channel85   -1.294e+05  3.600e+04  -3.594  0.01145 *
## Channel86    2.130e+05  4.345e+04   4.903  0.00270 **
## Channel87   -1.198e+05  3.818e+04  -3.139  0.02011 *
## Channel88   -2.199e+04  6.085e+04  -0.361  0.73021
## Channel89    7.974e+04  5.077e+04   1.571  0.16733
## Channel90   -1.711e+05  5.499e+04  -3.112  0.02079 *
## Channel91    2.107e+05  6.406e+04   3.289  0.01663 *
## Channel92   -1.959e+05  7.171e+04  -2.733  0.03407 *
## Channel93    2.874e+05  9.937e+04   2.892  0.02762 *
## Channel94   -3.064e+05  9.601e+04  -3.191  0.01881 *
## Channel95    2.048e+05  6.220e+04   3.292  0.01656 *
## Channel96   -5.600e+04  2.929e+04  -1.912  0.10441
## Channel97   -1.318e+04  3.050e+04  -0.432  0.68065
## Channel98   -2.724e+04  2.107e+04  -1.292  0.24375
## Channel99    3.556e+04  1.382e+04   2.573  0.04218 *
## Channel100  -1.206e+04  4.264e+03  -2.828  0.03006 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3191 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
```

(2)

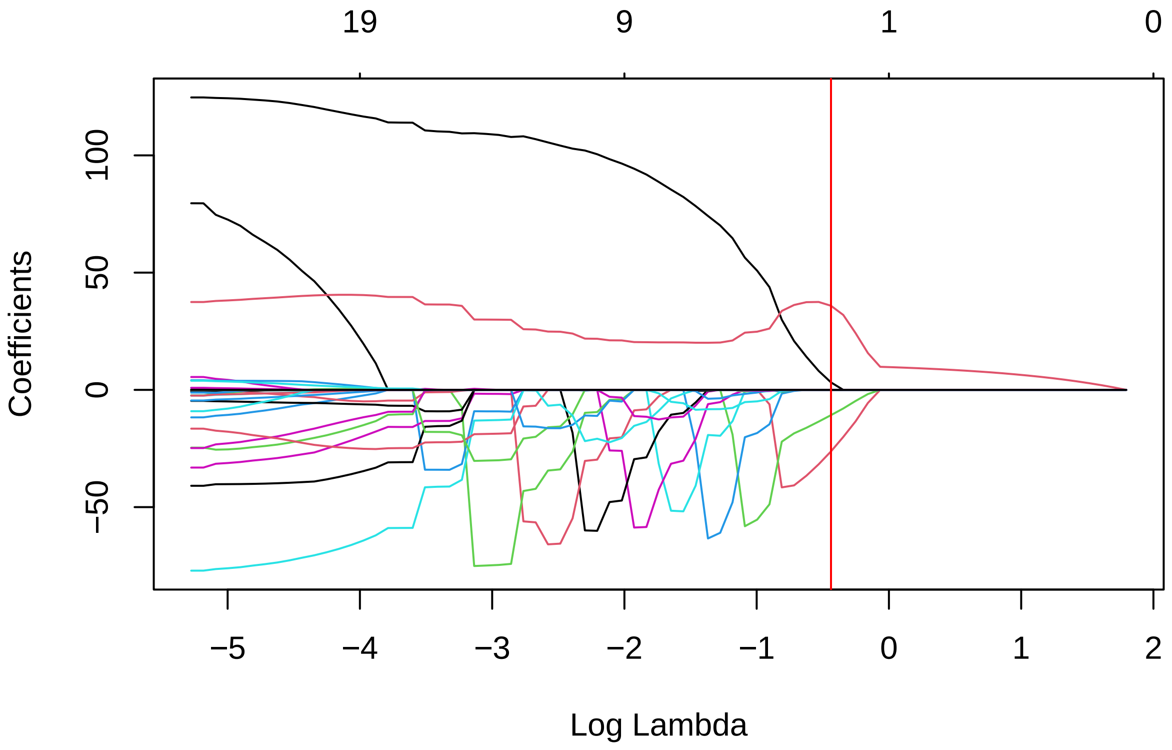
The cost function in lasso regression is:

$$\hat{\theta} = argmin_{\theta}\frac{1}{n}||X\theta||^2 + \lambda||\theta||_1$$

(3)

According to the plot of lasso_reg, we find that when $log(\lambda)$ is small, the degree of freedom is big which means more features explain the target variable, but when $log(\lambda)$ increases, the degree of freedom decrease at the same time, and coefficient get close to 0 in most of the time. We also find that some of the feature's coefficient line fluctuates when it is less than 0. But all of the features's coefficients will converge to 0 at last.

According to the calculation and output of the code, we find that when the feature number is 4 (including the intercept), lambda is: 0.6452974 and $log(\lambda)$ is -0.438044 .



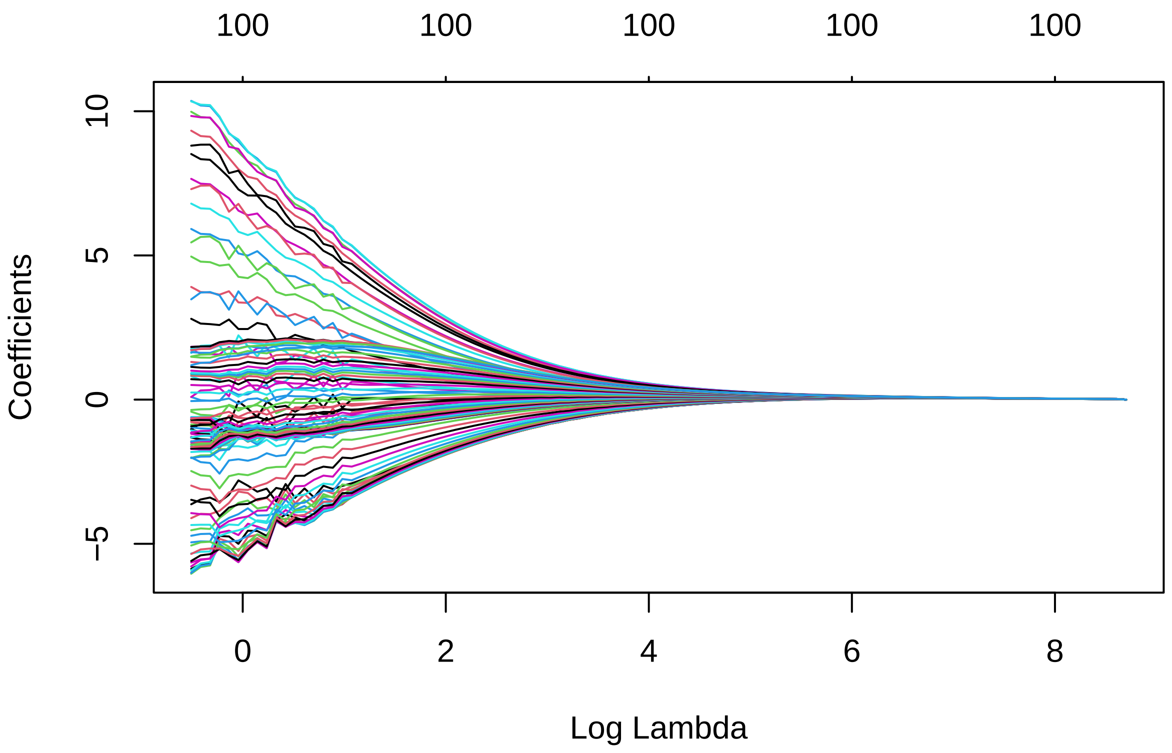
(4)

The cost function in ridge regression is:

$$\hat{\theta} = argmin_{\theta}\frac{1}{n}||X\theta||^2 + \lambda||\theta||_2^2$$

For ridge regression, we find that the degree of freedom always keeps the same when $log(\lambda)$ increases, but the coefficient of features seems to converge to 0, after checking the beta value of ridge_reg, we found that the coefficient of features will not converge to 0, it will become a very small value.

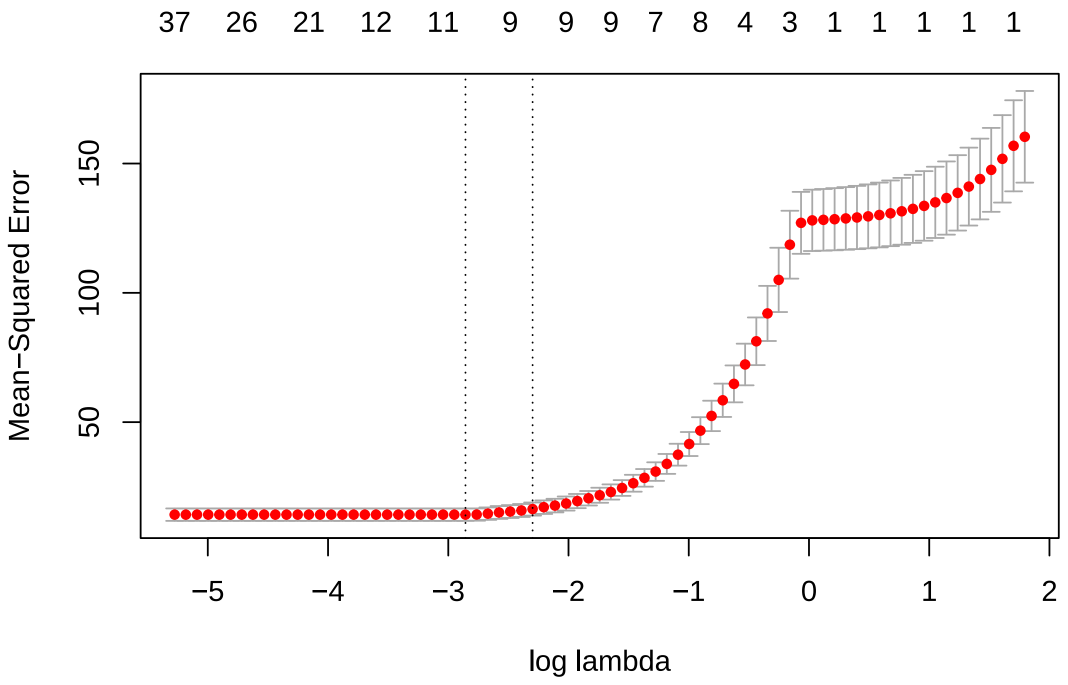
Compared to lasso regression, we can not find the required lambda value when the degree of freedom is 4, since the coefficient value not converge to 0, but converges to a very small value as stated above.



(5)

The plot is as follows, when $log(\lambda)$ increases, features explaining the model get fewer, and the model becomes simpler.

From the plot, we also find that when there are at least 9 features, the MSE is very stable and keeps at a lower level. However, when $log(\lambda)$ increases, MSE increases rapidly until 3 features. It means those 9 features are significant.



And optimized λ is 0.0574453 , $log(\lambda)$ is -2.8569213, in this case, 8 features were chosen.

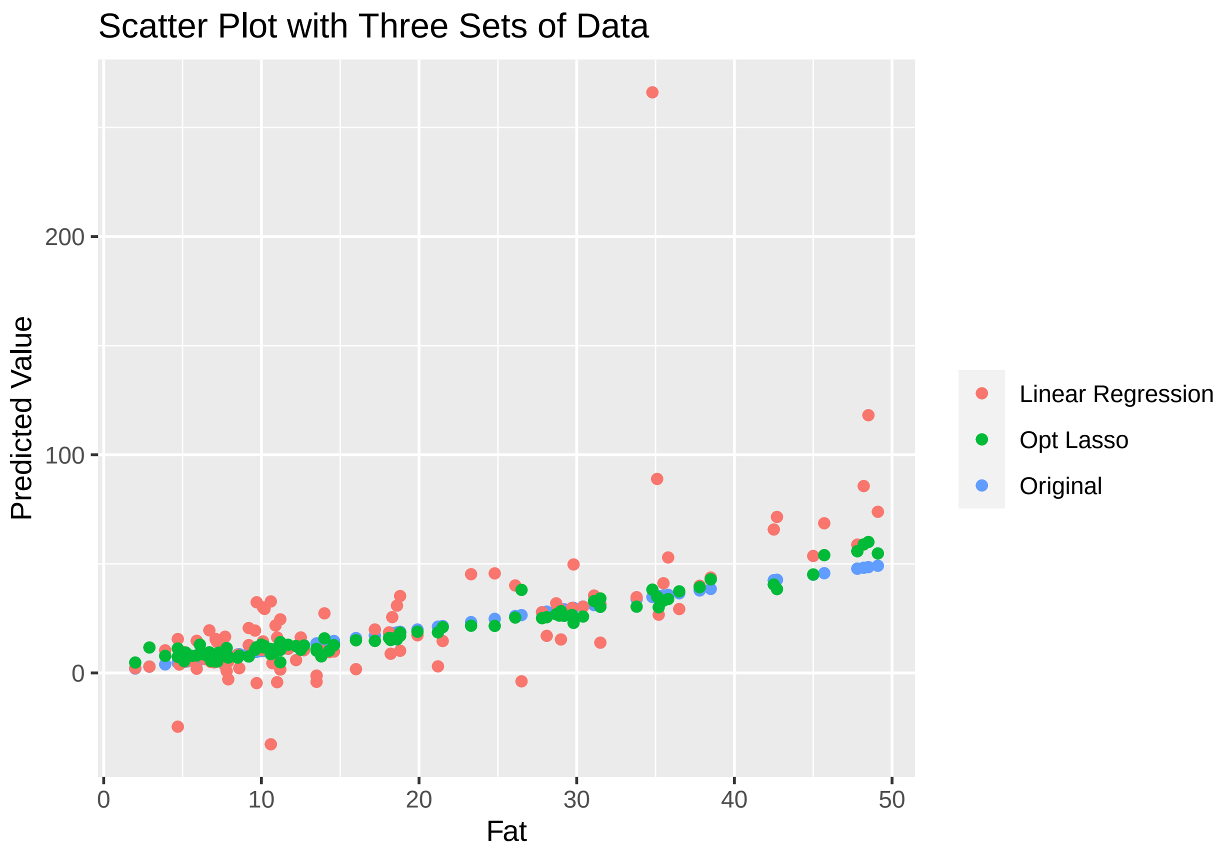
Compare the optimized λ and corresponding $log(\lambda)$ with the $log(\lambda) = -4$, according to the plot, it's relatively flat in this range. So we can say that it does not indicate it's possible for $log(\lambda) = -4$ to get statistical significally better prediction.

To compare the models, a scatter plot was used with different colors.

As we can see from the plot, the green dots(opt lasso) are closer to the original blue dots than the red dots(linear regression).

This means the linear regression model is not as good as the opt lasso model in this case.

When Fat(<10) is small, linear regression fits well, when Fat gets bigger, linear regression fit points have a big gap with the original data. However, the opt lasso model fits well in all the ranges. So regularization used in lasso helps to improve the model.



Assignment 2: Decision trees and logistic regression for bank marketing

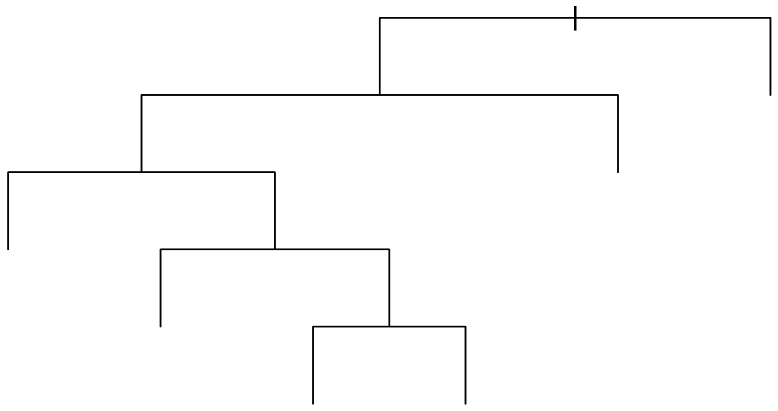
Answer:

(1) Divide the data

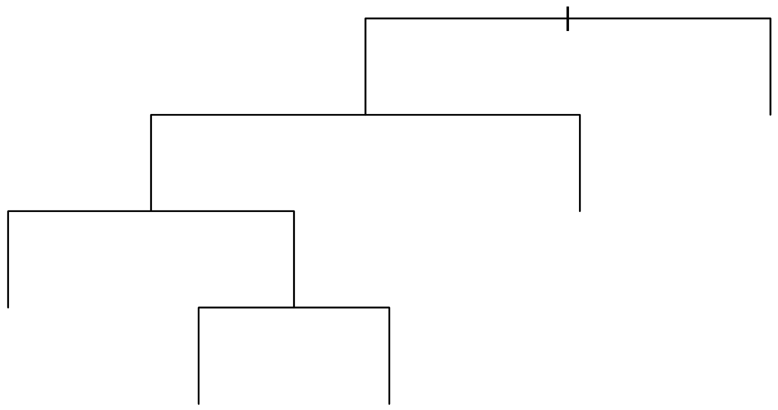
Please check the appendix for the code

(2)

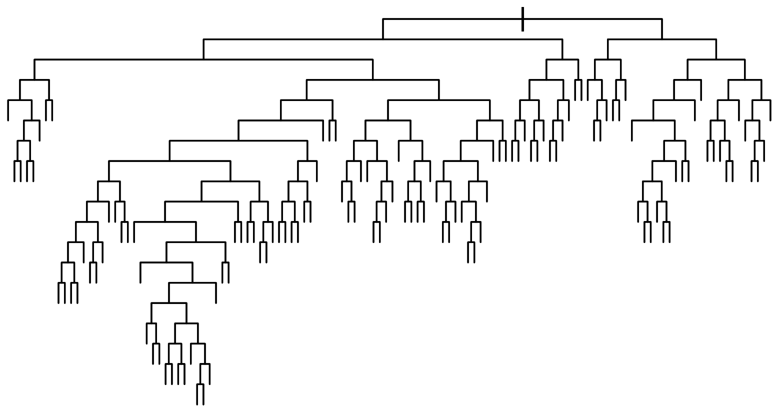
The following is the default tree.



Following is the tree with minimal node size = 7000.



Following is the tree with mindev = 0.0005.



```
## [1] "misclass rate of  tree_a  and  train_set  is  0.104844061048441 "
## [1] "misclass rate of  tree_a  and  validation_set  is  0.109267861092679 "
## [1] "misclass rate of  tree_b  and  train_set  is  0.104844061048441 "
## [1] "misclass rate of  tree_b  and  validation_set  is  0.109267861092679 "
## [1] "misclass rate of  tree_c  and  train_set  is  0.0936186684361867 "
## [1] "misclass rate of  tree_c  and  validation_set  is  0.11170095111701 "
```

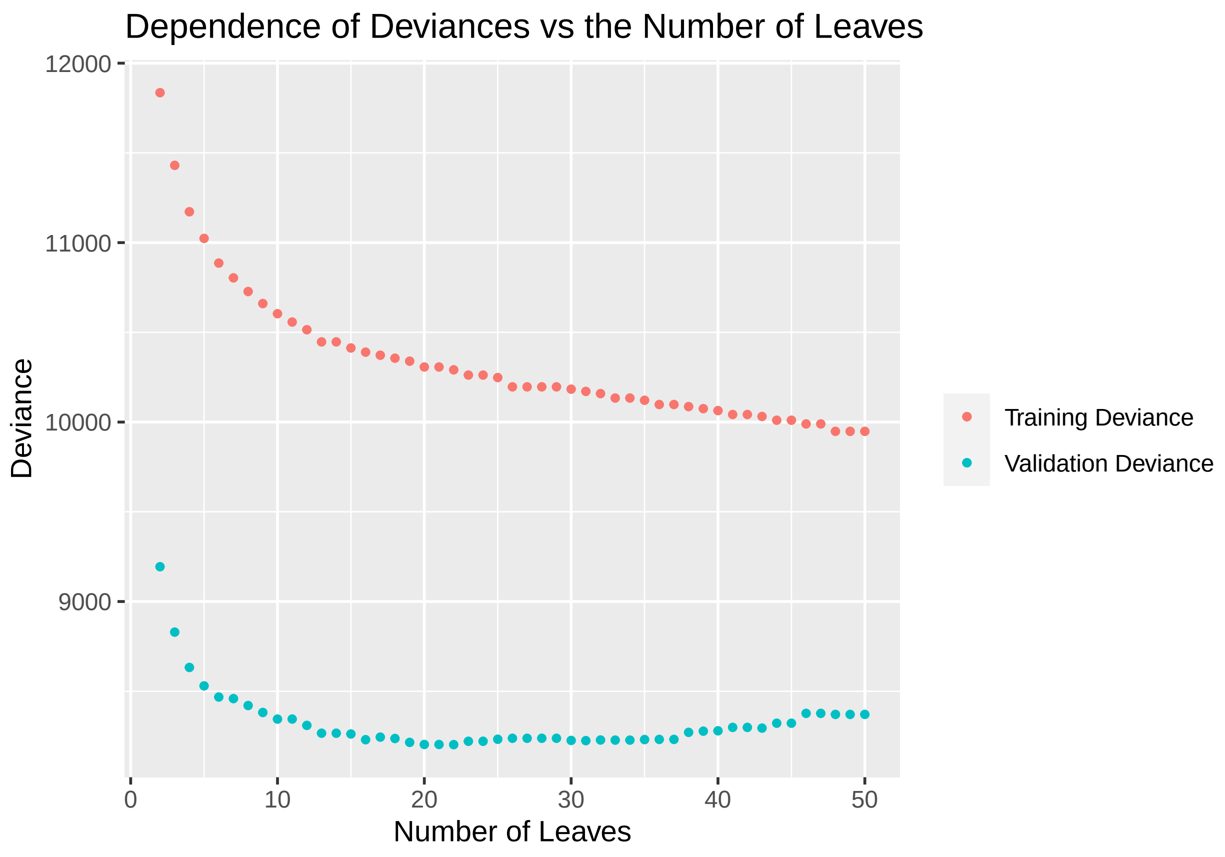
Misclassifications for different trees and datasets are listed above.

According to the misclassification rates and the plots, we can say that Tree B is the best choice, it has the same value as Tree A but is simpler than Tree A and much simpler than Tree C.

According to the information given, we know that a large node size allows leaf nodes to have more observations which will lead to fewer splits, resulting in a smaller tree.

A smaller deviance, however, allows the tree to be more flexible which leads to trees with many branches and leaves.

(3)



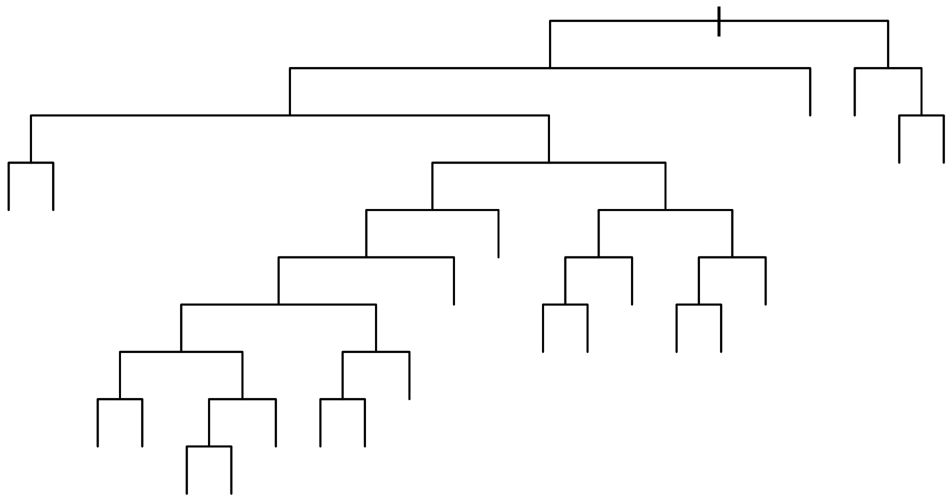
The above is the plot for the number of leaves vs deviance of training and validation data.

From the graph, we can see that when the number of leaves grows, the training deviance decreases, while the validation deviance decreases but then at around 20, it begins to grow.

That means on the left side of that point, the model training and validation data both fit well, but on the right side of that point, the validation data does not fit well.

The optimal point according to the output of the code, is 22.

We draw the tree based on the optimal number of leaves 22, followed by a summary of this tree.



```
##
## Classification tree:
## snip.tree(tree = tree_c, nodes = c(581L, 17L, 577L, 79L, 37L,
## 77L, 14L, 576L, 153L, 580L, 6L, 1157L, 15L, 16L, 5L, 1156L, 156L,
## 152L, 579L))
## Variables actually used in tree construction:
## [1] "poutcome" "month"    "contact"  "pdays"    "age"      "day"      "balance"
## [8] "housing"  "job"
## Number of terminal nodes:  22
## Residual mean deviance:  0.5698 = 10290 / 18060
## Misclassification error rate: 0.1039 = 1879 / 18084
```

According to the output, we know that the variances that are very important in the decision tree are:

poutcome, month, contact, pdays, age, day, balance, housing, job

(4)

An optimized tree depth = 1.

```
#######################  Assignment 2.4 ###########################################
final_tree <- rpart(y ~ ., data = train_set, method = "class",
control = rpart.control(cp = 0.0005), maxdepth = 1)
```

```
validation_predictions <- predict(final_tree, validation_set, type = "class")

conf_matrix <- confusionMatrix(validation_predictions, validation_set$y)

accuracy <- conf_matrix$overall["Accuracy"]
f1_score <- conf_matrix$byClass["F1"]

# Print the confusion matrix, accuracy, and F1 score
cat("Confusion Matrix:\n")
cat(conf_matrix$table,"\n")

cat("Accuracy:", accuracy, "\n")
cat("F1 Score:", f1_score, "\n")
```

(5)

```
######################  Assignment 2.5 ##########################################
loss_matrix <- matrix(c(0, 1, 5, 0), nrow = 2)
test_predictions <- predict(final_tree, test_set, type = "class", parms = list(loss = loss_matrix))
conf_matrix <- table(Actual = test_set$y, Predicted = test_predictions)
cat("Confusion Matrix:\n")
print(conf_matrix)
```

(6)

```
######################  Assignment 2.6 ##########################################
threshold <- 0.05
logistic_model <- glm(y ~ ., data = train_set, family = "binomial")

logistic_probabilities <- predict(logistic_model, test_set, type = "response")
logistic_predictions <- ifelse(logistic_probabilities > threshold, "yes", "no")

conf_matrix_logistic <- confusionMatrix(logistic_predictions, test_set$y)

TP <- conf_matrix_logistic$table["yes", "yes"]
FN <- conf_matrix_logistic$table["no", "yes"]
FP <- conf_matrix_logistic$table["yes", "no"]
TN <- conf_matrix_logistic$table["no", "no"]

TPR <- TP / (TP + FN)
FPR <- FP / (FP + TN)

# Print the results
cat("True Positive Rate (TPR):", TPR, "\n")
cat("False Positive Rate (FPR):", FPR, "\n")
```

Assignment 3. Principal components and implicit regularization

Answer:

(1)

According to the output, we need 35 components to obtain at least 95% of variance in the data. The proportion of variation explained by first and second principal components are 0.2501699 and 0.1693597 respectively.

```
######################  Assignment 3.1 ##########################################
n <- nrow(data)
features <- data[, -101]

s_features <- scale(features)

S <- (t(s_features) %*% s_features)/n  # sample covariance matrix

Eig <- eigen(S)

# eigen in descending order
s_indx <- order(Eig$values, decreasing = TRUE)
s_eig <- Eig$values[s_indx]
```

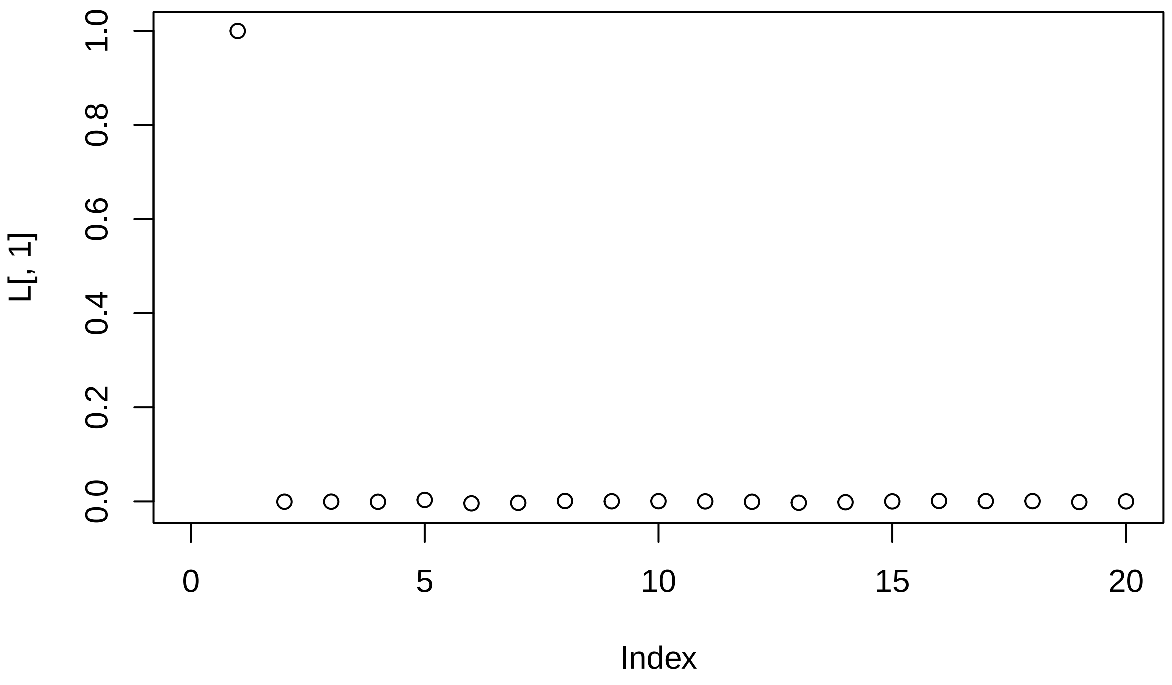
```r
# cumulative explained variance
cum_var <- cumsum(s_eig) / sum(s_eig)

q_95 <- which(cum_var >= 0.95)[1]  # q for 95% var

first_two_components <- Eig$vectors[, 1:2]  # first two PC

# proportion of variation explained by each of the first two components
PC1_var <- s_eig[1] / sum(s_eig)
PC2_var <- s_eig[2] / sum(s_eig)

cat("Number of components needed for 95% variance:", q_95, "
Proportion of variation explained by the first component:", PC1_var, "
Proportion of variation explained by the second component:", PC2_var, "\n")
```

```
## Number of components needed for 95% variance: 35
## Proportion of variation explained by the first component: 0.2501699
## Proportion of variation explained by the second component: 0.1693597
```

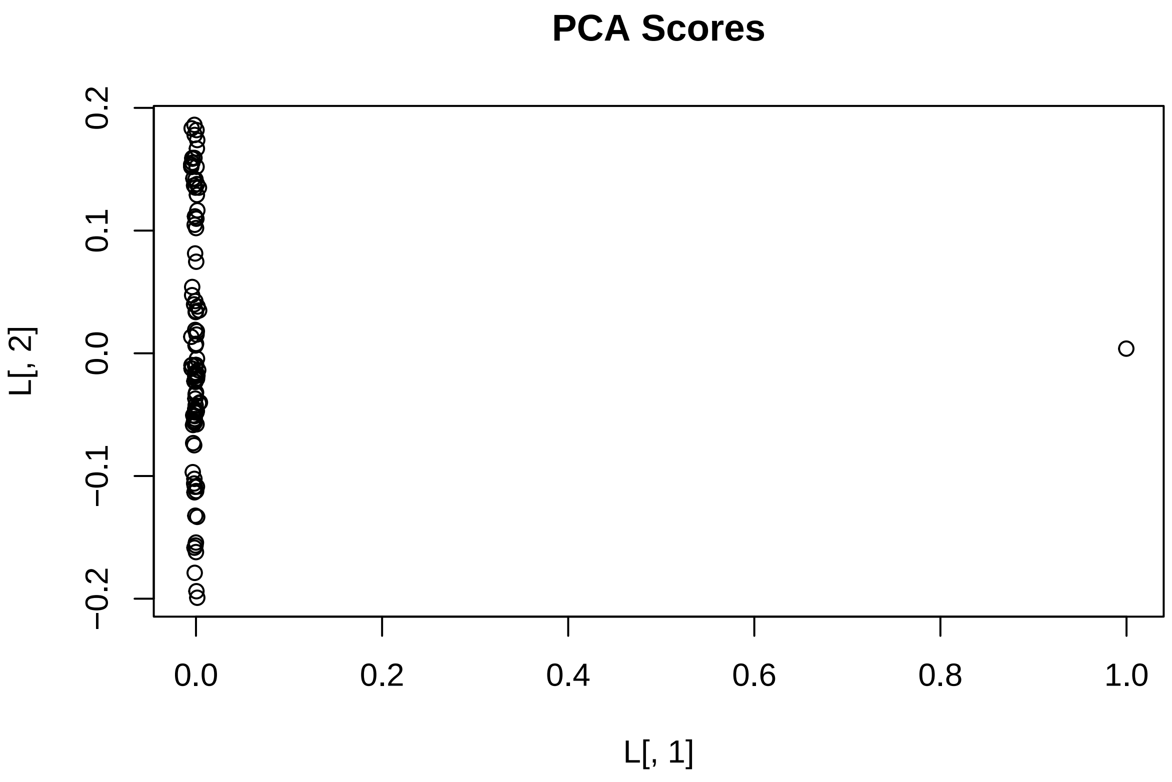
(2)

We observe from the plot how the variable "state", the first feature of PC1, explains most of the data. The other variables carry significantly less explanation. The other 4 features that contribute the most are features 93 (PctBornSameState), 61 (PctSpeakEnglOnly), 5 (racePctWhite) and 76 (PersPerRentOccHous).

```r
######################  Assignment 3.2 #########################################
PCA <- princomp(features)
L <- PCA$loadings
plot(L[,1], xlim=c(0,20))
```



```r
highest5 <- order(L[,1], decreasing=TRUE)[1:5]
plot(L[,1], L[,2], main="PCA Scores")
```



(3)

Train MSE is 0.2752071 and Test MES is 0.4248011. We observe how the MSE of the test data is significantly bigger than the MSE obtained from train data. This might happen because the model chosen is overfitting our given data.

```
######################  Assignment 3.3 ##########################################
# train and test data
id <- sample(1:n, floor(n*0.5))
trn <- data[id,]
tst <- data[-id,]

# scaling
scaler <- preProcess(trn)
trainS <- predict(scaler,trn)
testS <- predict(scaler,tst)

# linear regression model and test data predictions
linmod <- lm(trainS$ViolentCrimesPerPop ~ ., trainS)
test_pred <- predict(linmod, testS[,-101])

# training and test data MSE
train_MSE <- mean((trainS$ViolentCrimesPerPop - linmod$fitted.values)^2)
test_MSE <- mean((testS$ViolentCrimesPerPop - test_pred)^2)

cat("Train mean squared error:", train_MSE, "\nTest mean squared error:", test_MSE)
```

```
## Train mean squared error: 0.2752071
## Test mean squared error: 0.4248011
```

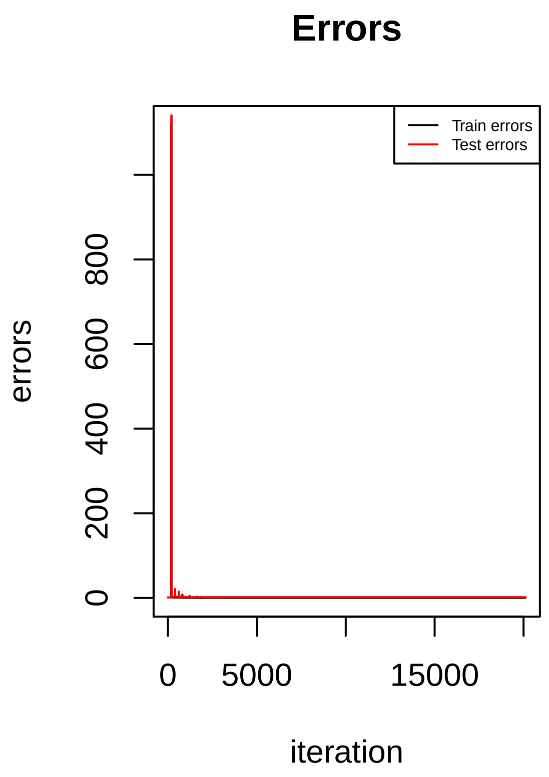
(4)

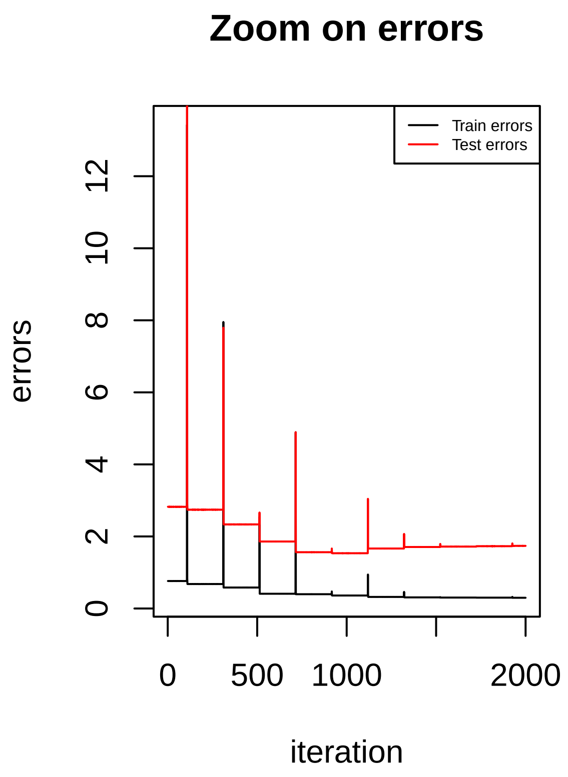
Looking at the plots we can see how the errors converge to zero after 1700 iterations (approx 1200 in the second graph but considering that we took off the first 500). Hence 1700 is the optimal iteration number to get good results. The following iterations do not significally improve our model and hence can led us to overfitting.

```
######################  Assignment 3.4 #########################################
train_new <- as.matrix(trainS[,-101])  # training data - response variable
train_r <- trainS[,101] # training response variable

test_new <- as.matrix(testS[,-101])  # test data - response variable
test_r <- trainS[,101] # test response variable

# error vectors for training and test data
train_e <- c()
test_e <- c()

costfun <- function(theta_vec){
  train_cost <- mean(((train_new %*% theta_vec) - train_r)^2)
  train_e <<- c(train_e, train_cost)
  test_cost <- mean(((test_new %*% theta_vec) - test_r)^2)
  test_e <<- c(test_e, test_cost)
  return(train_cost)
}

theta0 <- rep(0,100)
opt <- optim(theta0, method="BFGS", costfun)

opt$val
```

```
## [1] 0.2752213
```

```
par(mfrow=c(1,2))

plot(ylab="errors", xlab="iteration",train_e, type = "l", main="Errors")
lines(test_e, col="red")
legend("topright", legend = c("Train errors", "Test errors"),
col = c("black", "red"), lty = 1, cex = 0.5)

plot(ylab="errors", xlab="iteration",train_e[-c(1:500)][1:2000],
type = "l", main="Zoom on errors")
lines(test_e[-c(1:500)][1:2000], col="red")
legend("topright", legend = c("Train errors", "Test errors"),
col = c("black", "red"), lty = 1, cex = 0.5)
```


Errors
Train errors
Test errors
errors
0
200
400
600
800
0
5000
15000
iteration


Zoom on errors
Train errors
Test errors
errors
0
2
4
6
8
10
12
0
500
1000
2000
iteration

Appendix: All code for this report

```r
###########################  Init code For  Assignment 1 #######################
rm(list = ls())
library(plyr)
library(readr)
library(dplyr)
library(caret)
library(ggplot2)
library(repr)
library(glmnet)
library(rpart)
library(tree)

# set random seed
set.seed(12345)
######################  Assignment 1.1 #########################################


# read data
data <- read.csv("tecator.csv")
data <- data %>% select(Fat,Channel1:Channel100)

row_num <- nrow(data)
# set data split ratio to 0.5, 0.5
ratio <- c(train = .5, test = .5)

# split data to the train dataset and the test dataset
train_id <- sample(1:row_num, floor(row_num * ratio[1]))
train_data_set <- data[train_id, ]

test_id <- setdiff(1:row_num, train_id)
test_data_set <- data[test_id, ]

# fit linear model
lm_model <- lm(Fat ~ ., data = train_data_set)

predicted_fat_test <- predict(lm_model, test_data_set)
predicted_fat_train <- predict(lm_model, train_data_set)

# calc the mean value
test_mse <- mean((predicted_fat_test - test_data_set$Fat)^2)
train_mse <- mean((predicted_fat_train - train_data_set$Fat)^2)
cat("test_mse is:",test_mse,"\n")
cat("train_mse is:",train_mse,"\n")

summary(lm_model)
######################  Assignment 1.3 #########################################
x_train <- as.matrix(train_data_set %>% select(-Fat))
y_train <- as.matrix(train_data_set %>% select(Fat))

# when alpha = 1, it is lasso regression
lasso_reg <- glmnet(x_train, y_train, alpha = 1,family = "gaussian")
```

```r
lambda_deg_freedom <- data.frame(lambda = lasso_reg$lambda,
                                 deg_freedom = lasso_reg$df)

lambda_deg_freedom <- lambda_deg_freedom %>% arrange(desc(deg_freedom))

lambda_deg_freedom_num <- lambda_deg_freedom[which(lambda_deg_freedom$deg_freedom == 4),] %>%
                        head(1) %>% pull(lambda)

#cat("lambda is ",lambda_deg_freedom_num,
#    " log(lambda) is ",log(lambda_deg_freedom_num),
#    "\nwhen feature number is 4 (including the intercept)\n")
plot(lasso_reg, xvar="lambda", xlab='Log Lambda')
abline(v=log(lambda_deg_freedom_num), col="red")
######################  Assignment 1.4 ##########################################
# when alpha = 0, it is ridge regression
ridge_reg <- glmnet(x_train, y_train, alpha = 0,family = "gaussian")

lambda_deg_freedom <- data.frame(lambda = ridge_reg$lambda,
                                 deg_freedom = ridge_reg$df)

lambda_deg_freedom <- lambda_deg_freedom %>% arrange(desc(deg_freedom))

lambda_deg_freedom_num <- lambda_deg_freedom[which(lambda_deg_freedom$deg_freedom == 4),] %>%
                        head(1) %>% pull(lambda)

plot(ridge_reg,xvar="lambda", xlab='Log Lambda')
# when alpha = 1, it is lasso regression
cv_lasso_model <- cv.glmnet(x_train, y_train, alpha = 1, family = "gaussian")

# Plot the dependence of CV score on log(lambda)
plot(cv_lasso_model,xlab = "log lambda")
# Extract lambda values and CV scores
opt_lambda <- cv_lasso_model$lambda.min
opt_features <- cv_lasso_model$nzero[which(cv_lasso_model$lambda == opt_lambda)]

#cat("Optimal log(lambda) is:",log(opt_lambda),"\n")
#cat("Corresponding features used is:", opt_features,"\n")
x_test <- as.matrix(test_data_set %>% select(-Fat))
y_test <- as.matrix(test_data_set %>% select(Fat))

opt_lasso_model <- glmnet(x_train, y_train, alpha = 1, family = "gaussian", lambda = opt_lambda)
opt_lasso_prediction <- predict(opt_lasso_model, 'response', newx = x_test)

test_len <- length(y_test)
scatter_df = data.frame(
              x = rep(y_test, 3),
              y = c(y_test, predicted_fat_test, opt_lasso_prediction),
              Model = rep(c("Original","Linear Regression","Opt Lasso"), each = test_len)
            )
ggplot(scatter_df, aes(x = x, y = y, color = Model)) +
  geom_point() +
  labs(title = "Scatter Plot with Three Sets of Data", x = "Fat", y = "Predicted Value")  +
  scale_color_discrete(name="")
```

```r
###########################  Init code For  Assignment 1 #######################
rm(list = ls())
######################  Assignment 2.1 #########################################


# Read data
data <- read.csv("bank-full.csv",header = TRUE, sep = ";")

# remove the duration column
data <- data %>% select(-duration)

# convert categorical variables to factors
data <- data %>% mutate_if(is.character, as.factor)

row_num <- nrow(data)
cols_num <- ncol(data)

# Set data split ratio to 0.4, 0.3, 0.3
ratio <- c(train = .4, validate = 0.3, test = .3)

# Set random seed
set.seed(12345)

# Split data to train, validate and test
train_id <- sample(1:row_num, floor(row_num * ratio[1]))
train_set <- data[train_id, ]

# Set random seed
set.seed(12345)

validation_test_id <- setdiff(1:row_num, train_id)
validation_id <- sample(validation_test_id, floor(row_num * ratio[2]))
validation_set <- data[validation_id, ]

test_id <- setdiff(validation_test_id, validation_id)
test_set <- data[test_id, ]
######################  Assignment 2.2 #########################################

# default fit
tree_a <- tree(y ~ ., data = train_set)
plot(tree_a,type = "uniform",main = "default fit")
# min node size = 7000
tree_b <- tree(y ~ .,
               data = train_set,
               control = tree.control(nobs = nrow(train_set), minsize = 7000),
               split = c("deviance","gini"))
plot(tree_b,type = "uniform",main = "min node size = 7000")
# min deviance = 0.0005
tree_c <- tree(y ~ .,
               data = train_set,
               control = tree.control(nobs = nrow(train_set), mindev = 0.0005),
               split = c("deviance","gini"))
plot(tree_c,type = "uniform",main = "min deviance = 0.0005")
```

```r
calc_misclassification_error <- function(A,B){
  return(1-sum(diag(table(A,B) ))/ length(A))
}

calc_misclassification_rate <- function(tree_model,dataset){
  predictions <- predict(object = tree_model, newdata=dataset)

  maxIndex <- apply(predictions, 1, which.max)
  prediction_tree_calc <- levels(dataset$y)[maxIndex]
  data_set_misclassification_rate <- calc_misclassification_error(dataset$y,prediction_tree_calc)
  info <- paste("misclass rate of ",as.character(substitute(tree_model))," and ",
             as.character(substitute(dataset)) ," is ",data_set_misclassification_rate,"")
  misclassifications <<- c(misclassifications,info)
}

misclassifications <- c()

calc_misclassification_rate(tree_a,train_set)
calc_misclassification_rate(tree_a,validation_set)
calc_misclassification_rate(tree_b,train_set)
calc_misclassification_rate(tree_b,validation_set)
calc_misclassification_rate(tree_c,train_set)
calc_misclassification_rate(tree_c,validation_set)

for (i in 1:6){
  print(misclassifications[i])
}

optimal_depth <- cv.tree(tree_c)

training_score<- rep(1,50)
validation_score <- rep(1,50)

prune_df <- data.frame(matrix(ncol = 3,nrow = 0))
colnames(prune_df) <- c("Leaves","Train_Deviance","Validation_Deviance")

# from 2 because 'predict' will have an error complaining about applied
# to an object of class "singlenode"
for(i in 2:50){
  opt_tree_prune <- prune.tree(tree_c,best = i)
  pred_valid <- predict(opt_tree_prune,
                        newdata = validation_set,
                        type = "tree")

  training_score[i] <- deviance(opt_tree_prune)
  validation_score[i] <- deviance(pred_valid)

  prune_df <- rbind(
                    prune_df,
                    data.frame(
                      'Leaves' = i,
                      'Train_Deviance' = training_score[i],
                      'Validation_Deviance' = validation_score[i]
```

```
                        )
                      )
}

# plot the dependence of deviances on training and validation data vs the number of leaves
ggplot(data = prune_df, aes(x = Leaves)) +
  geom_point(aes(y = Train_Deviance, color = "Training Deviance"), size = 1) +
  geom_point(aes(y = Validation_Deviance, color = "Validation Deviance"), size = 1) +
  labs(title = "Dependence of Deviances vs the Number of Leaves",
       x = "Number of Leaves",
       y = "Deviance") +
  scale_color_discrete(name="")

optimal_leaves <- prune_df[which.min(prune_df$Validation_Deviance),1]
opt_tree_with_optimal_leaves <- prune.tree(tree_c,best = optimal_leaves)
plot(opt_tree_with_optimal_leaves,type = "uniform",main = "tree with optimal leaves")
summary(opt_tree_with_optimal_leaves)
######################  Assignment 2.3 #########################################

# num_folds <- 5
# cv_inds <- cut(seq(1, nrow(train_set)), breaks = num_folds, labels = FALSE)
# cv_errors <- numeric(50)
# train_deviances_depth <- numeric(50)
# validation_deviances_depth <- numeric(50)

# for (depth in 1:30) {
#   tree <- rpart(y ~ ., data = train_set, method = "class",
#   control = rpart.control(cp = 0.0005), maxdepth = depth)

#   validation_predictions <- predict(tree, validation_set)

#   cv_errors_depth <- mean(validation_predictions != validation_set$y)

#   cv_errors[depth] <- cv_errors_depth

#   train_deviances_depth[depth] <- sum(deviance(tree))
#   validation_deviances_depth[depth] <- sum(deviance(tree, newdata = validation_set))
# }

# optimal_depth <- which.min(cv_errors)
# cat("Optimal tree depth:", optimal_depth, "\n")

# plot_data <- data.frame(Leaves = 1:30, Train_Deviance = train_deviances_depth,
# Validation_Deviance = validation_deviances_depth)

# ggplot(plot_data, aes(x = Leaves)) +
#   geom_line(aes(y = Train_Deviance, color = "Training Data"), size = 1) +
#   geom_line(aes(y = Validation_Deviance, color = "Validation Data"), size = 1) +
#   labs(title = "Dependence of Deviances vs the Number of Leaves",
#        x = "Number of Leaves",
#        y = "Deviance") +
#   scale_color_manual(values = c("Training Data" = "blue", "Validation Data" = "red"))
######################  Assignment 2.4 #########################################
```

```r
final_tree <- rpart(y ~ ., data = train_set, method = "class",
control = rpart.control(cp = 0.0005), maxdepth = 1)

validation_predictions <- predict(final_tree, validation_set, type = "class")

conf_matrix <- confusionMatrix(validation_predictions, validation_set$y)

accuracy <- conf_matrix$overall["Accuracy"]
f1_score <- conf_matrix$byClass["F1"]

# Print the confusion matrix, accuracy, and F1 score
cat("Confusion Matrix:\n")
cat(conf_matrix$table,"\n")

cat("Accuracy:", accuracy, "\n")
cat("F1 Score:", f1_score, "\n")
######################  Assignment 2.5 #########################################
loss_matrix <- matrix(c(0, 1, 5, 0), nrow = 2)
test_predictions <- predict(final_tree, test_set, type = "class", parms = list(loss = loss_matrix))
conf_matrix <- table(Actual = test_set$y, Predicted = test_predictions)
cat("Confusion Matrix:\n")
print(conf_matrix)

######################  Assignment 2.6 #########################################
threshold <- 0.05
logistic_model <- glm(y ~ ., data = train_set, family = "binomial")

logistic_probabilities <- predict(logistic_model, test_set, type = "response")
logistic_predictions <- ifelse(logistic_probabilities > threshold, "yes", "no")

conf_matrix_logistic <- confusionMatrix(logistic_predictions, test_set$y)

TP <- conf_matrix_logistic$table["yes", "yes"]
FN <- conf_matrix_logistic$table["no", "yes"]
FP <- conf_matrix_logistic$table["yes", "no"]
TN <- conf_matrix_logistic$table["no", "no"]

TPR <- TP / (TP + FN)
FPR <- FP / (FP + TN)

# Print the results
cat("True Positive Rate (TPR):", TPR, "\n")
cat("False Positive Rate (FPR):", FPR, "\n")
###########################  Init code For  Assignment 3 ######################
rm(list=ls(all.names = T))
library(ggplot2)
library(caret)
set.seed(12345)
data <- read.csv("communities.csv")
######################  Assignment 3.1 #########################################
n <- nrow(data)
features <- data[, -101]
```

```r
s_features <- scale(features)

S <- (t(s_features) %*% s_features)/n  # sample covariance matrix

Eig <- eigen(S)

# eigen in descending order
s_indx <- order(Eig$values, decreasing = TRUE)
s_eig <- Eig$values[s_indx]

# cumulative explained variance
cum_var <- cumsum(s_eig) / sum(s_eig)

q_95 <- which(cum_var >= 0.95)[1]  # q for 95% var

first_two_components <- Eig$vectors[, 1:2]  # first two PC

# proportion of variation explained by each of the first two components
PC1_var <- s_eig[1] / sum(s_eig)
PC2_var <- s_eig[2] / sum(s_eig)

cat("Number of components needed for 95% variance:", q_95, "
Proportion of variation explained by the first component:", PC1_var, "
Proportion of variation explained by the second component:", PC2_var, "\n")
######################  Assignment 3.2 ########################################
PCA <- princomp(features)
L <- PCA$loadings
plot(L[,1], xlim=c(0,20))
highest5 <- order(L[,1], decreasing=TRUE)[1:5]
plot(L[,1], L[,2], main="PCA Scores")
######################  Assignment 3.3 ########################################
# train and test data
id <- sample(1:n, floor(n*0.5))
trn <- data[id,]
tst <- data[-id,]

# scaling
scaler <- preProcess(trn)
trainS <- predict(scaler,trn)
testS <- predict(scaler,tst)

# linear regression model and test data predictions
linmod <- lm(trainS$ViolentCrimesPerPop ~ ., trainS)
test_pred <- predict(linmod, testS[,-101])

# training and test data MSE
train_MSE <- mean((trainS$ViolentCrimesPerPop - linmod$fitted.values)^2)
test_MSE <- mean((testS$ViolentCrimesPerPop - test_pred)^2)

cat("Train mean squared error:", train_MSE, "\nTest mean squared error:", test_MSE)

######################  Assignment 3.4 ########################################
train_new <- as.matrix(trainS[,-101])  # training data - response variable
```

```r
train_r <- trainS[,101] # training response variable

test_new <- as.matrix(testS[,-101])  # test data - response variable
test_r <- trainS[,101] # test response variable

# error vectors for training and test data
train_e <- c()
test_e <- c()

costfun <- function(theta_vec){
  train_cost <- mean(((train_new %*% theta_vec) - train_r)^2)
  train_e <<- c(train_e, train_cost)
  test_cost <- mean(((test_new %*% theta_vec) - test_r)^2)
  test_e <<- c(test_e, test_cost)
  return(train_cost)
}

theta0 <- rep(0,100)
opt <- optim(theta0, method="BFGS", costfun)

opt$val

par(mfrow=c(1,2))

plot(ylab="errors", xlab="iteration",train_e, type = "l", main="Errors")
lines(test_e, col="red")
legend("topright", legend = c("Train errors", "Test errors"),
col = c("black", "red"), lty = 1, cex = 0.5)

plot(ylab="errors", xlab="iteration",train_e[-c(1:500)][1:2000],
type = "l", main="Zoom on errors")
lines(test_e[-c(1:500)][1:2000], col="red")
legend("topright", legend = c("Train errors", "Test errors"),
col = c("black", "red"), lty = 1, cex = 0.5)
```