Machine Learning Computer Lab 2 (Group A7)

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

2023-11-26

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:plyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

## Loading required package: ggplot2

## Loading required package: lattice

## Loading required package: Matrix

## Loaded glmnet 4.1-8
```

Assignment 1: Explicit regularization(by Qinyuan Qi(qinqi464))

Answer:

(1)

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

```
######################  Assignment 1.1 ########################################

selected_x_columns <- grep(paste0("^", "Channel"), names(train_set), value = TRUE)

X_data_train <- train_set[, selected_x_columns]
Y_data_train <- train_set[names(train_set) == "Fat"]
train_data_set <- cbind(X_data_train, Y_data_train)

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

```r
# predict the test set and train set
X_data_test <- test_set[, selected_x_columns]
Y_data_test <- test_set[names(test_set) == "Fat"]
test_data_set <- cbind(X_data_test, Y_data_test)
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")
```

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

```r
cat("train_mse is:",train_mse,"\n")
```

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

```r
summary(lm_model)
```

```
##
## 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:

$$MSE(y, y_pred) + \alpha * \sum_{i=1}^{n} |\theta_i|$$

(3)

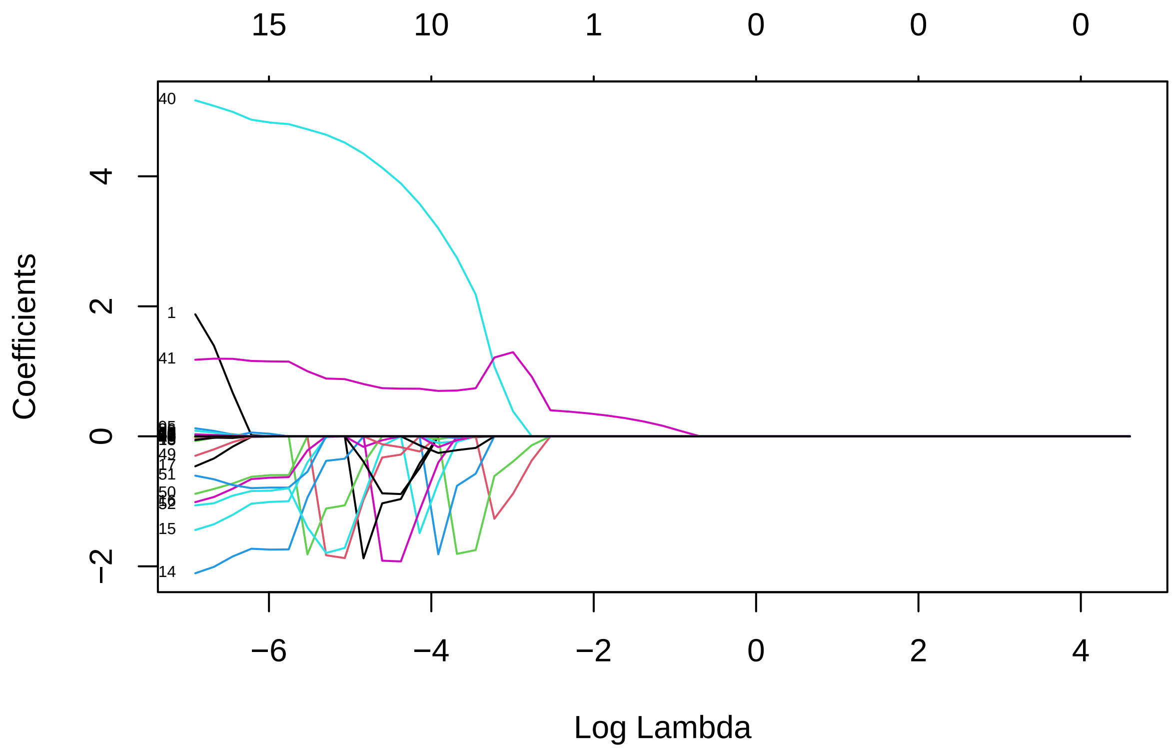
According to the plot of lasso_reg, we choose top edge number to 3 which contains 3 curves and choose the appropriate lambda value , which is around -3.

```
######################  Assignment 1.3 ########################################
# regularize the data before we fit the model
pre_proc_val <- preProcess(train_data_set, method = c("center", "scale"))
train_data_set <- predict(pre_proc_val, train_data_set)
test_data_set <- predict(pre_proc_val, test_data_set)

new_cols_num <- ncol(train_data_set)

X_data_train <- train_data_set[, 1:(new_cols_num - 1)]
Y_data_train <- train_data_set[, new_cols_num]
X_data_train <- as.matrix(X_data_train)

dummies <- dummyVars(Fat ~ ., data = train_data_set)
```

```r
train_dummies <- predict(dummies, newdata = train_data_set)
test_dummies <- predict(dummies, newdata = test_data_set)

x <- as.matrix(train_dummies)
y_train <- train_data_set$Fat

x_test <- as.matrix(test_dummies)
y_test <- test_data_set$Fat

# set range of lambda, we can know the range of lambda, then set it manually
grid <- 10^seq(2, -3, by = -.1)

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

summary(lasso_reg)
```

```
##           Length Class     Mode
## a0          51   -none-    numeric
## beta      5100   dgCMatrix S4
## df          51   -none-    numeric
## dim          2   -none-    numeric
## lambda      51   -none-    numeric
## dev.ratio   51   -none-    numeric
## nulldev      1   -none-    numeric
## npasses      1   -none-    numeric
## jerr         1   -none-    numeric
## offset       1   -none-    logical
## call         6   -none-    call
## nobs         1   -none-    numeric
```

```r
plot(lasso_reg, xvar = "lambda", label = TRUE)
```



(4)

the cost function in ridge regression is:

$$MSE(y, y_pred) + \alpha * \sum_{i=1}^{n} (\theta_i)^2$$

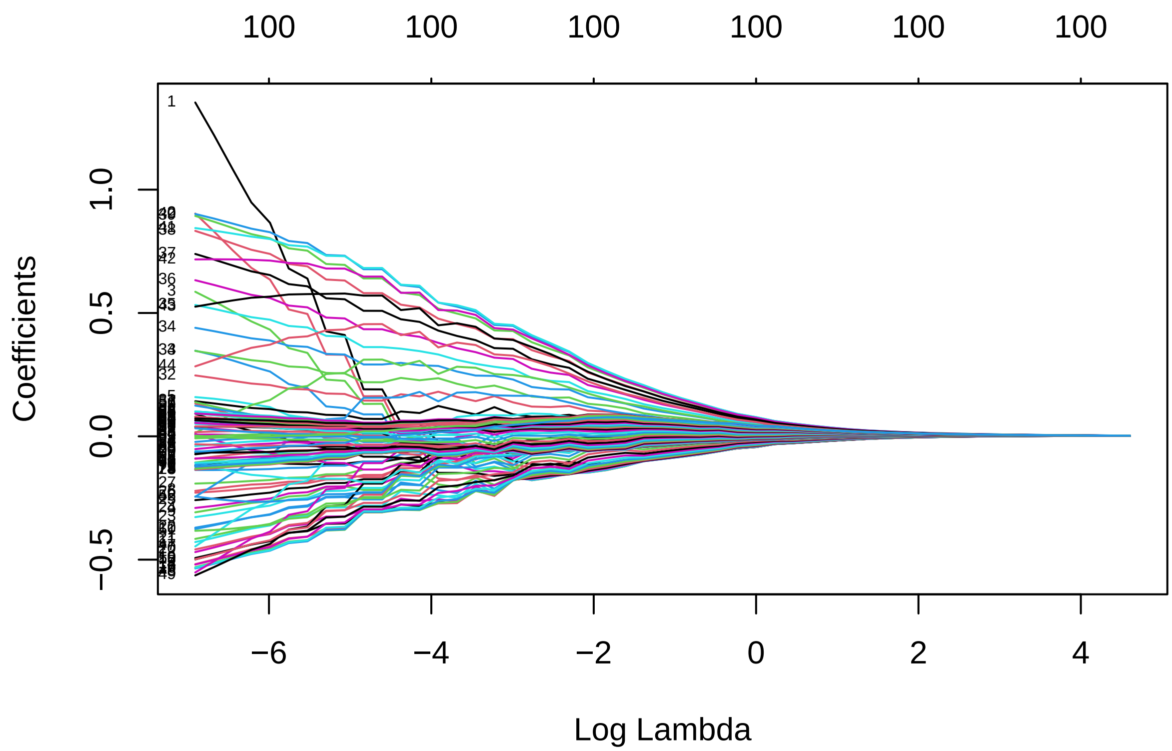
Compare 2 plots, we can find that for lasso regression, most of the features' coefficient convergence speed is very different. however, when using ridge regression, most of the features' coefficient convergence speed is almost same when log(lambda) increase.

```
######################  Assignment 1.4 #########################################
# when alpha = 0, it is ridge regression
ridge_reg <- glmnet(x, y_train, alpha = 0, lambda = grid)

summary(ridge_reg)
```

```
##             Length Class     Mode
## a0            51   -none-    numeric
## beta        5100   dgCMatrix S4
## df            51   -none-    numeric
## dim            2   -none-    numeric
## lambda        51   -none-    numeric
## dev.ratio     51   -none-    numeric
## nulldev        1   -none-    numeric
## npasses        1   -none-    numeric
## jerr           1   -none-    numeric
```

```
## offset       1    -none-    logical
## call         5    -none-    call
## nobs         1    -none-    numeric
```

```r
plot(ridge_reg, xvar = "lambda", label = TRUE)
```



(5)

The plot as follows, when log(lambda) increase, CV score increase. Optimized lambda is -5.39019, and in this case, around 10 variables were chosen.

When lambda = -4, MSE is higher than when lambda=-5.39019. we can say that it results in a statistically significantly better prediction.

The scatter plot as follow.

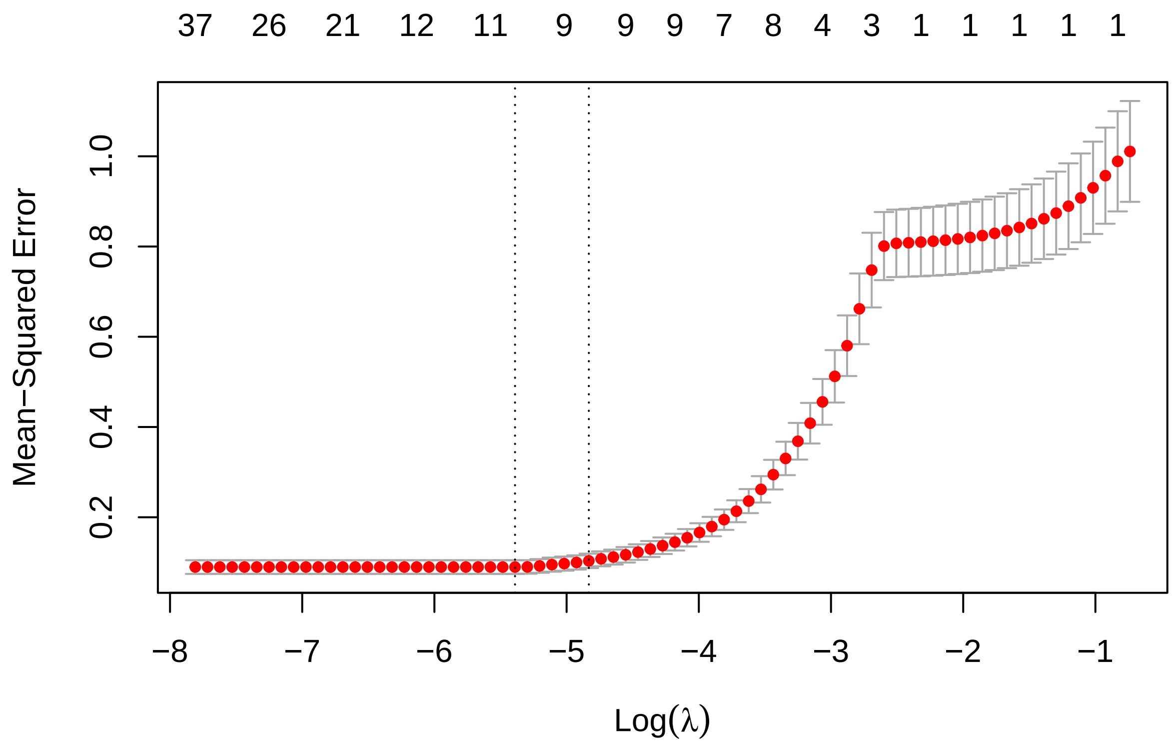
Since almost all points are around the 45-degree line,it indicate accurate predictions.

```r
# when alpha = 1, it is lasso regression
cv_lasso_model <- cv.glmnet(x, y_train, alpha = 1)
summary(cv_lasso_model)
```

```
##              Length Class  Mode
## lambda       77     -none- numeric
## cvm          77     -none- numeric
## cvsd         77     -none- numeric
## cvup         77     -none- numeric
## cvlo         77     -none- numeric
## nzero        77     -none- numeric
```

```
## call        4     -none- call
## name        1     -none- character
## glmnet.fit 12     elnet  list
## lambda.min  1     -none- numeric
## lambda.1se  1     -none- numeric
## index       2     -none- numeric
```

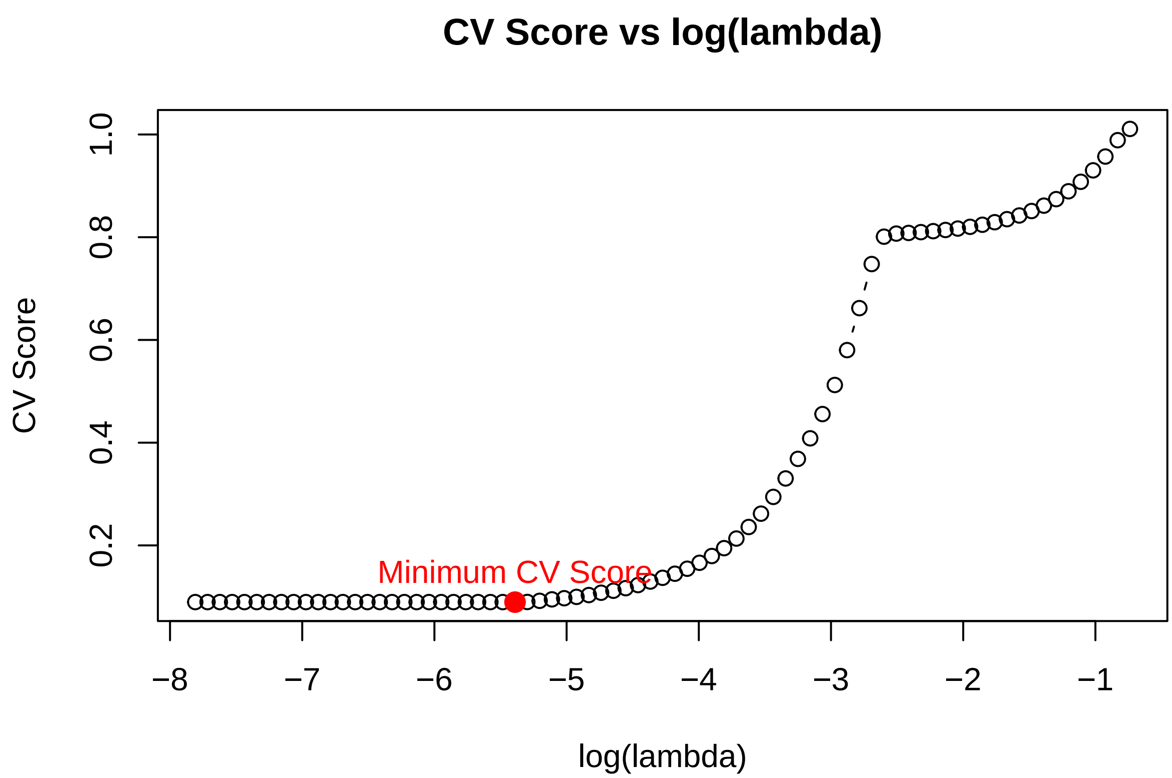
```r
# Extract lambda values and CV scores
lambda_values <- log(cv_lasso_model$lambda)
cv_scores <- cv_lasso_model$cvm

# Plot the dependence of CV score on log(lambda)
plot(cv_lasso_model)
```



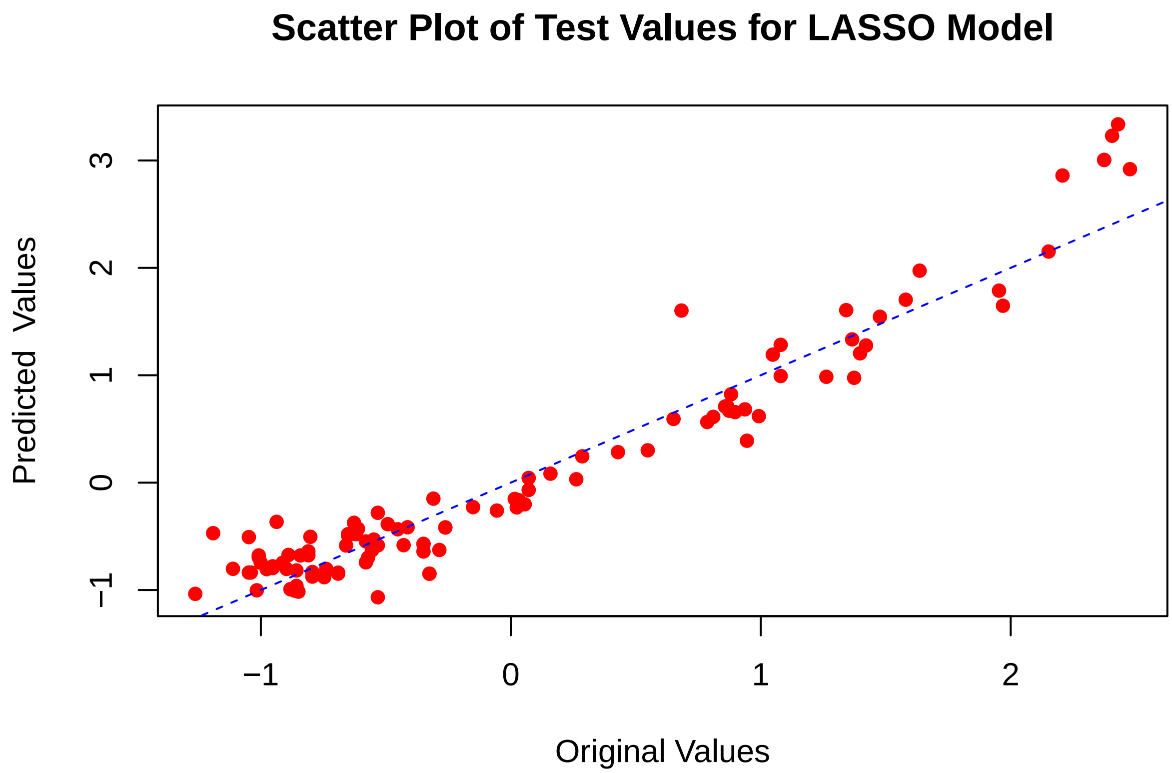

```r
plot(lambda_values, cv_scores, type = "b", xlab = "log(lambda)", ylab = "CV Score",main = "CV Score vs

min_cv_lambda <- log(cv_lasso_model$lambda.min)
min_cv_score <- min(cv_scores)
cat("min_cv_lambda is:",min_cv_lambda,"\n")
```

```
## min_cv_lambda is: -5.39019
```

```r
points(min_cv_lambda, min_cv_score, col = "red", pch = 16, cex = 1.5)
text(min_cv_lambda, min_cv_score, "Minimum CV Score", pos = 3, col = "red")
```




```r
predictions <- predict(cv_lasso_model, newx = as.matrix(x_test), s = "lambda.min")

plot(y_test, predictions, pch = 16, col = "red",
     xlab = "Original Values", ylab = "Predicted  Values",
     main = "Scatter Plot of Test Values for LASSO Model")
abline(a = 0, b = 1, col = "blue", lty = 2)
```



Assignment 2: Decision trees and logistic regression for bank marketing(by Satya Sai Naga Jaya Koushik Pilla)

Answer:

(1)

```
######################  Assignment 2.1 #########################################
# read data
data <- read.csv("bank-full.csv",header = TRUE, sep = ";")

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 training,validate and test dataset
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, ]
```

(2)

Decision trees are as follows.

misclassification is listed below.

	misclassification rate of train	misclassification rate of validation
a	0.0995355	0.0995355
b	0.1142446	0.1207697
c	0.06287326	0.1041805

According to the data in the table above, the first one has a good misclassification rate of train and validation. the last one has a good misclassification rate of train but a little bit large misclassification rate on validation set. b will not be considered, since both of values are biggest among 3 methods.

```
######################  Assignment 2.2 ##########################################
tree_a <- rpart(y ~ ., data = train_set, method = "class")
tree_b <- rpart(y ~ ., data = train_set, method = "class", control = rpart.control(minbucket = 7000))
tree_c <- rpart(y ~ ., data = train_set, method = "class", control = rpart.control(cp = 0.0005))

train_predictions_a <- predict(tree_a, train_set, type = "class")
validation_predictions_a <- predict(tree_a, validation_set, type = "class")

train_predictions_b <- predict(tree_b, train_set, type = "class")
validation_predictions_b <- predict(tree_b, validation_set, type = "class")

train_predictions_c <- predict(tree_c, train_set, type = "class")
validation_predictions_c <- predict(tree_c, validation_set, type = "class")

# misclassification rates
train_misclassification_rate_a <- mean(train_predictions_a != train_set$y)
validation_misclassification_rate_a <- mean(validation_predictions_a != validation_set$y)

train_misclassification_rate_b <- mean(train_predictions_b != train_set$y)
validation_misclassification_rate_b <- mean(validation_predictions_b != validation_set$y)

train_misclassification_rate_c <- mean(train_predictions_c != train_set$y)
validation_misclassification_rate_c <- mean(validation_predictions_c != validation_set$y)




cat("train_misclassification_rate_a is ",train_misclassification_rate_a,"\n")
```

```
## train_misclassification_rate_a is  0.0995355
```

```
cat("validation_misclassification_rate_a is ",validation_misclassification_rate_a,"\n")
```

```
## validation_misclassification_rate_a is  0.0995355
```

```
cat("train_misclassification_rate_b is ",train_misclassification_rate_b,"\n")
```

```
## train_misclassification_rate_b is  0.1142446
```

```
cat("validation_misclassification_rate_b is ",validation_misclassification_rate_b,"\n")
```

```
## validation_misclassification_rate_b is  0.1207697
```

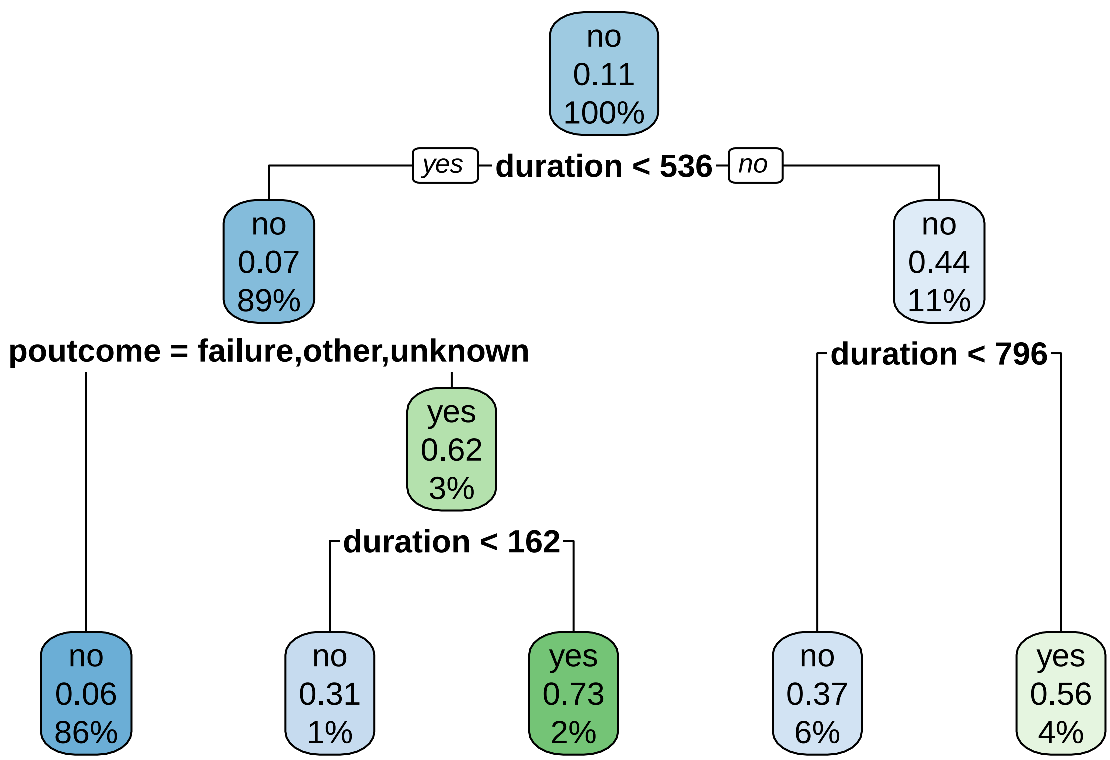
```
cat("train_misclassification_rate_c is ",train_misclassification_rate_c,"\n")
```

```
## train_misclassification_rate_c is  0.06287326
```

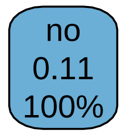
```
cat("validation_misclassification_rate_c is ",validation_misclassification_rate_c,"\n")
```

```
## validation_misclassification_rate_c is  0.1041805
```

```
rpart.plot(tree_a)
```

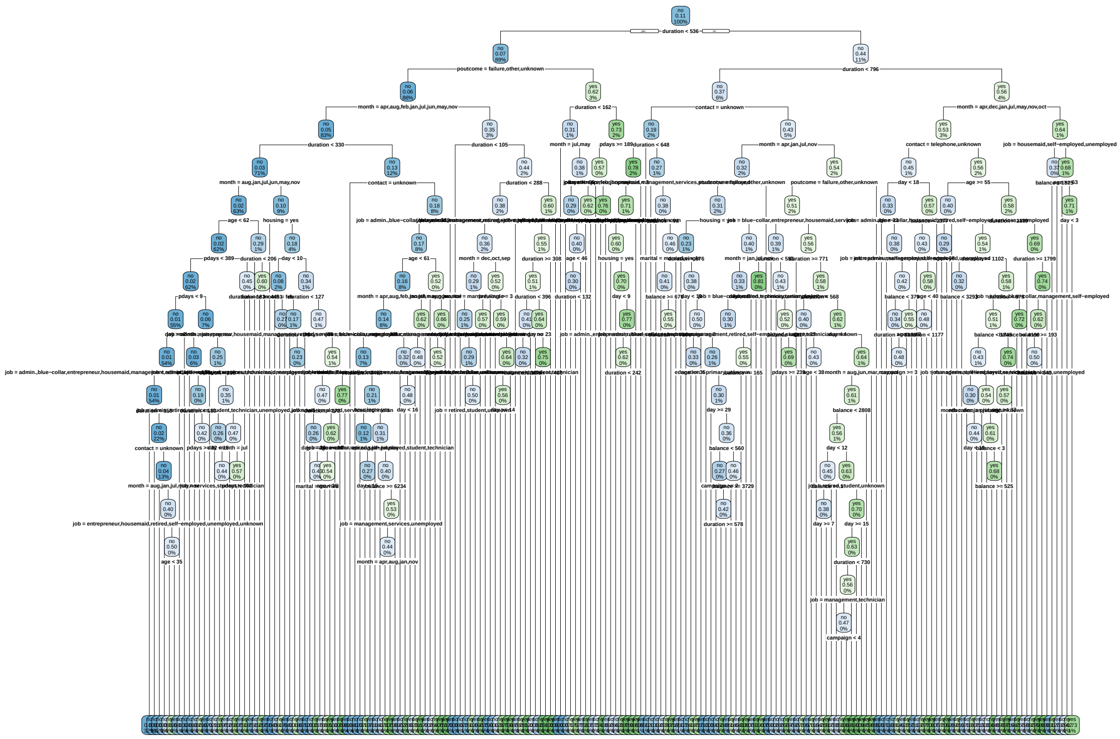



```
rpart.plot(tree_b)
```




```
rpart.plot(tree_c)
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



(3)

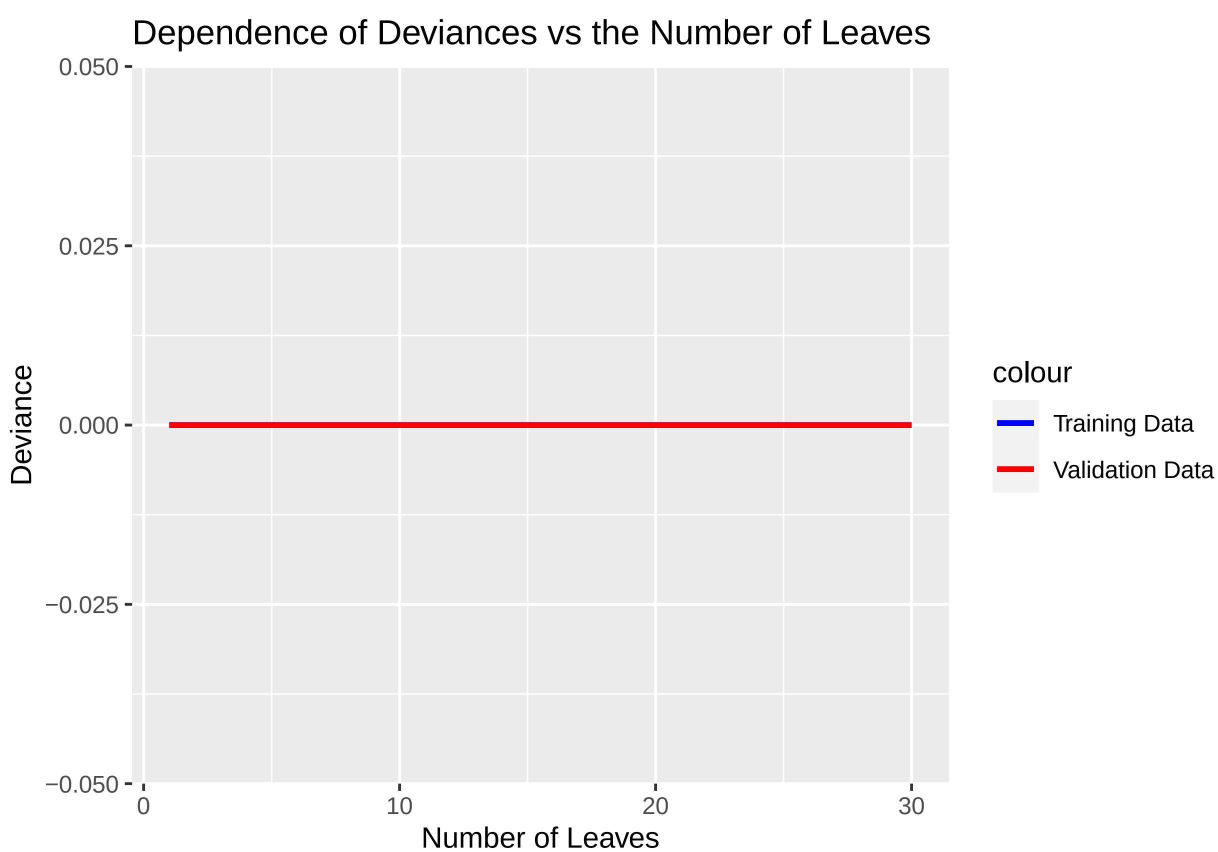
```r
######################  Assignment 2.3 #########################################
# Set folds number
num_folds <- 5
cv_inds <- cut(seq(1, nrow(train_set)), breaks = num_folds, labels = FALSE)
cv_errors <- numeric(30)
train_deviances_depth <- numeric(30)
validation_deviances_depth <- numeric(30)

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, type = "class")

  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)
```

```r
cat("Optimal tree depth:", optimal_depth, "\n")
```

```
## Optimal tree depth: 1
```

```r
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"))
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



(4)

An optimized tree depth = 1.

```r
######################  Assignment 2.4 ########################################
final_tree <- rpart(y ~ ., data = train_set, method = "class",
```

```r
control = rpart.control(cp = 0.0005), maxdepth = 1)
```

```r
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)

```r
######################  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")
```

```
## Confusion Matrix:
```

```r
print(conf_matrix)
```

```
##       Predicted
## Actual    no   yes
##    no  11471   508
##    yes   928   657
```

(6)

```r
######################  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")
```

```r
cat("False Positive Rate (FPR):", FPR, "\n")
```

Assignment 3. Principal components and implicit regularization(by Daniele Bozzoli(danbo826))

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.

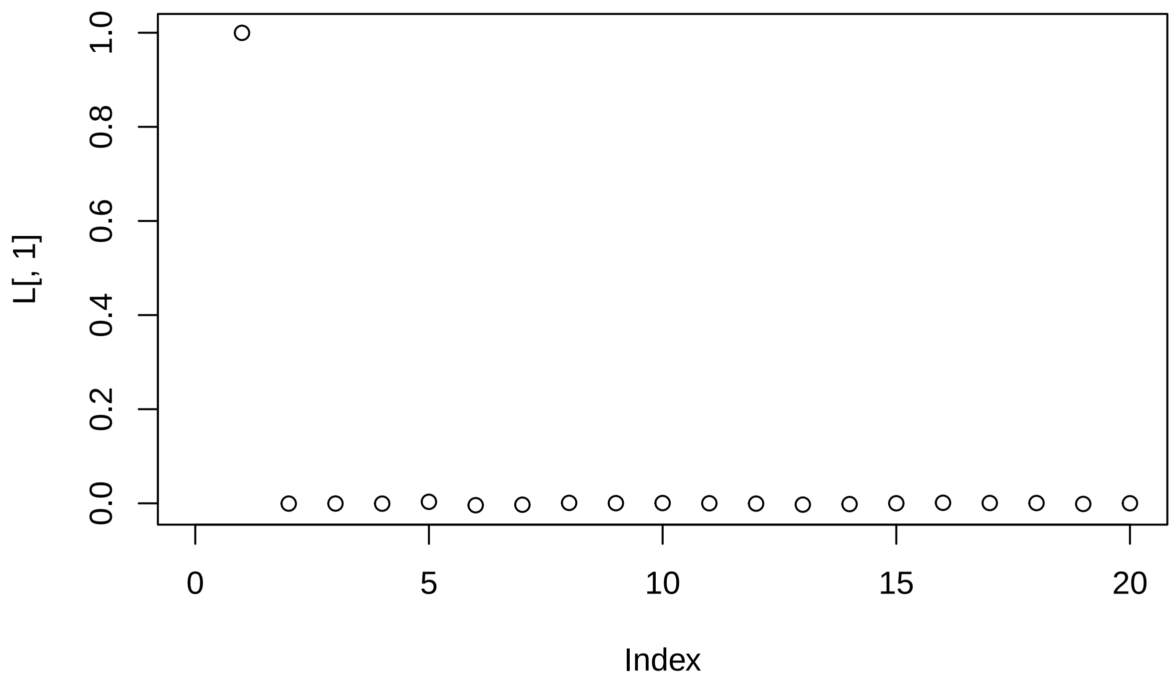
```r
######################  Assignment 3.1 #########################################
n <- nrow(data)
features <- data[, -101]

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]

# 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
```

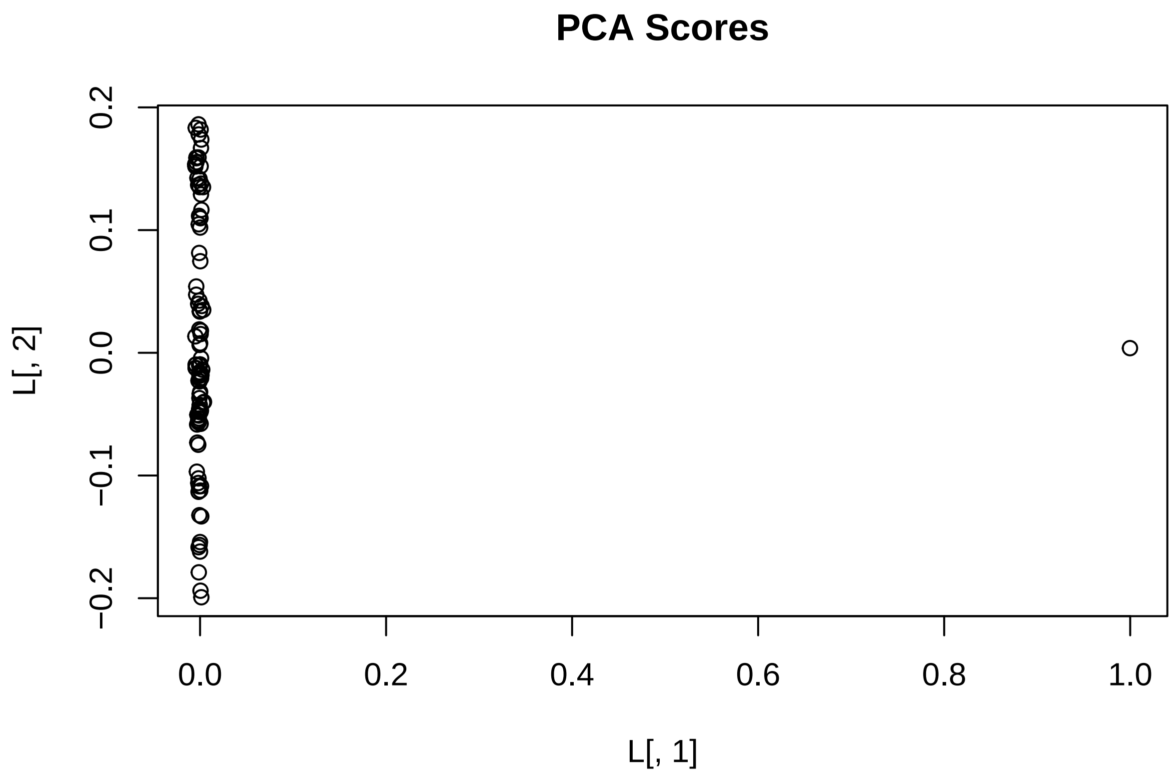
(2)

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.

```r
######################  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
```

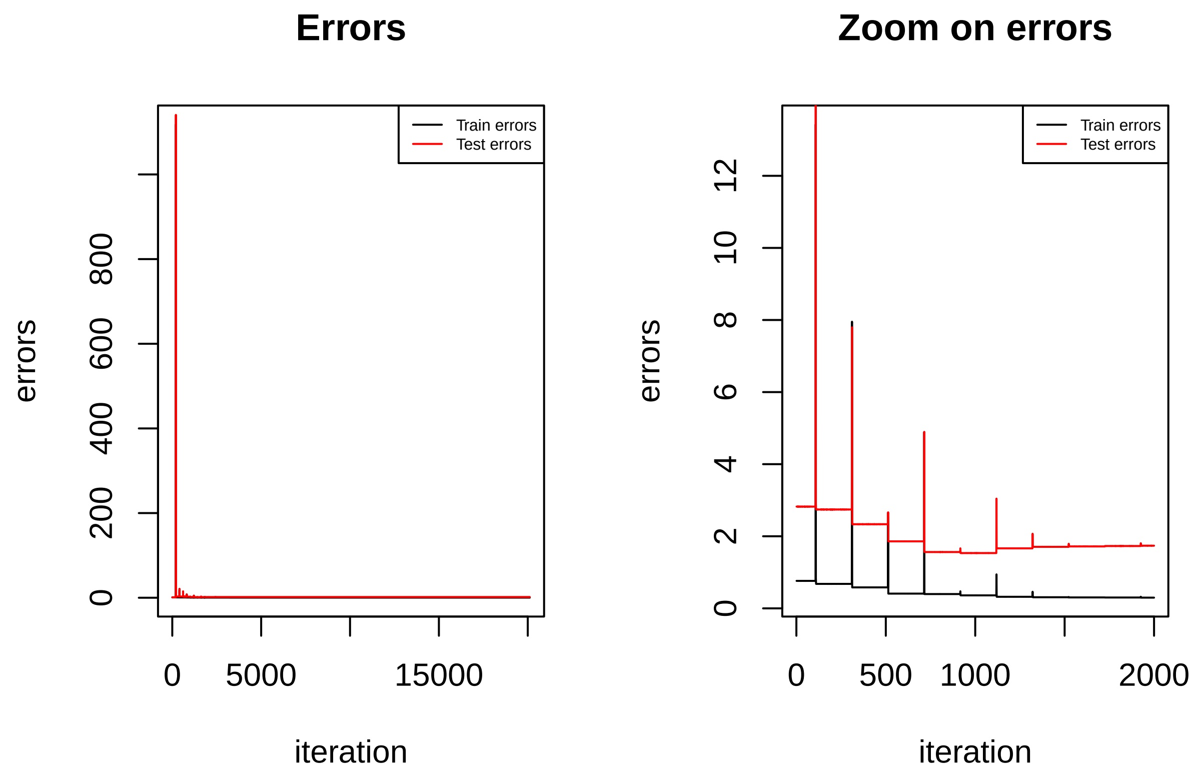
(4)

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.

```r
######################  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
```

```r
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

```
###########################  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(rpart.plot)

# read data
data <- read.csv("tecator.csv")

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

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

# set random seed
set.seed(12345)

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

test_id <- setdiff(1:row_num, train_id)
test_set <- data[test_id, ]
######################  Assignment 1.1 #########################################

selected_x_columns <- grep(paste0("^", "Channel"), names(train_set), value = TRUE)

X_data_train <- train_set[, selected_x_columns]
Y_data_train <- train_set[names(train_set) == "Fat"]
train_data_set <- cbind(X_data_train, Y_data_train)

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


# predict the test set and train set
X_data_test <- test_set[, selected_x_columns]
Y_data_test <- test_set[names(test_set) == "Fat"]
test_data_set <- cbind(X_data_test, Y_data_test)
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)
```

```r
cat("test_mse is:",test_mse,"\n")
cat("train_mse is:",train_mse,"\n")

summary(lm_model)
######################  Assignment 1.3 #########################################
# regularize the data before we fit the model
pre_proc_val <- preProcess(train_data_set, method = c("center", "scale"))
train_data_set <- predict(pre_proc_val, train_data_set)
test_data_set <- predict(pre_proc_val, test_data_set)

new_cols_num <- ncol(train_data_set)

X_data_train <- train_data_set[, 1:(new_cols_num - 1)]
Y_data_train <- train_data_set[, new_cols_num]
X_data_train <- as.matrix(X_data_train)

dummies <- dummyVars(Fat ~ ., data = train_data_set)

train_dummies <- predict(dummies, newdata = train_data_set)
test_dummies <- predict(dummies, newdata = test_data_set)

x <- as.matrix(train_dummies)
y_train <- train_data_set$Fat

x_test <- as.matrix(test_dummies)
y_test <- test_data_set$Fat

# set range of lambda, we can know the range of lambda, then set it manually
grid <- 10^seq(2, -3, by = -.1)

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

summary(lasso_reg)

plot(lasso_reg, xvar = "lambda", label = TRUE)
######################  Assignment 1.4 #########################################
# when alpha = 0, it is ridge regression
ridge_reg <- glmnet(x, y_train, alpha = 0, lambda = grid)

summary(ridge_reg)

plot(ridge_reg, xvar = "lambda", label = TRUE)
# when alpha = 1, it is lasso regression
cv_lasso_model <- cv.glmnet(x, y_train, alpha = 1)
summary(cv_lasso_model)

# Extract lambda values and CV scores
lambda_values <- log(cv_lasso_model$lambda)
cv_scores <- cv_lasso_model$cvm

# Plot the dependence of CV score on log(lambda)
plot(cv_lasso_model)
```

```
plot(lambda_values, cv_scores, type = "b", xlab = "log(lambda)", ylab = "CV Score",main = "CV Score vs l

min_cv_lambda <- log(cv_lasso_model$lambda.min)
min_cv_score <- min(cv_scores)
cat("min_cv_lambda is:",min_cv_lambda,"\n")
points(min_cv_lambda, min_cv_score, col = "red", pch = 16, cex = 1.5)
text(min_cv_lambda, min_cv_score, "Minimum CV Score", pos = 3, col = "red")

predictions <- predict(cv_lasso_model, newx = as.matrix(x_test), s = "lambda.min")

plot(y_test, predictions, pch = 16, col = "red",
     xlab = "Original Values", ylab = "Predicted  Values",
     main = "Scatter Plot of Test Values for LASSO Model")
abline(a = 0, b = 1, col = "blue", lty = 2)
###########################  Init code For  Assignment 1 #######################
rm(list = ls())

######################  Assignment 2.1 #########################################
# read data
data <- read.csv("bank-full.csv",header = TRUE, sep = ";")

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 training,validate and test dataset
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 #########################################
tree_a <- rpart(y ~ ., data = train_set, method = "class")
tree_b <- rpart(y ~ ., data = train_set, method = "class", control = rpart.control(minbucket = 7000))
tree_c <- rpart(y ~ ., data = train_set, method = "class", control = rpart.control(cp = 0.0005))

train_predictions_a <- predict(tree_a, train_set, type = "class")
validation_predictions_a <- predict(tree_a, validation_set, type = "class")

train_predictions_b <- predict(tree_b, train_set, type = "class")
```

```r
validation_predictions_b <- predict(tree_b, validation_set, type = "class")

train_predictions_c <- predict(tree_c, train_set, type = "class")
validation_predictions_c <- predict(tree_c, validation_set, type = "class")

# misclassification rates
train_misclassification_rate_a <- mean(train_predictions_a != train_set$y)
validation_misclassification_rate_a <- mean(validation_predictions_a != validation_set$y)

train_misclassification_rate_b <- mean(train_predictions_b != train_set$y)
validation_misclassification_rate_b <- mean(validation_predictions_b != validation_set$y)

train_misclassification_rate_c <- mean(train_predictions_c != train_set$y)
validation_misclassification_rate_c <- mean(validation_predictions_c != validation_set$y)



cat("train_misclassification_rate_a is ",train_misclassification_rate_a,"\n")
cat("validation_misclassification_rate_a is ",validation_misclassification_rate_a,"\n")

cat("train_misclassification_rate_b is ",train_misclassification_rate_b,"\n")
cat("validation_misclassification_rate_b is ",validation_misclassification_rate_b,"\n")

cat("train_misclassification_rate_c is ",train_misclassification_rate_c,"\n")
cat("validation_misclassification_rate_c is ",validation_misclassification_rate_c,"\n")

rpart.plot(tree_a)
rpart.plot(tree_b)
rpart.plot(tree_c)
######################  Assignment 2.3 #########################################
# Set folds number
num_folds <- 5
cv_inds <- cut(seq(1, nrow(train_set)), breaks = num_folds, labels = FALSE)
cv_errors <- numeric(30)
train_deviances_depth <- numeric(30)
validation_deviances_depth <- numeric(30)

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, type = "class")

  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 ##########################################
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")
```

```r
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]

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
```

```r
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
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)
```