STATISTICS AND STATISTICAL DATA MINING COURSEWORK 1

Name : Ahaan Tagare, Student ID: 33865799

Task 1

In summary, this work uses the Boston dataset to estimate the crime rate (crim). Correlated predictors are eliminated, and the data is divided into training and test sets. Backward elimination is used to modify a regression model and identify important factors. Test MSE is used to determine which of the final model and k-NN models (k=1,2,3k = 1, 2, 3k=1, 2,3) is the most accurate.

The script makes use of numerous R packages to allow efficient data analysis and statistics. The (MASS) library provides access to data sets and functions used in statistical research, whereas the (car) library ai evaluates the quality and assumptions of linear and generalized linear models. The readr library is used to import and export data files with ease. The caret library is a complete tool for developing and assessing machine learning models. The FNN library has capabilities for doing rapid nearest neighbor searches, which is critical for istic regression and neural network modeling. Finally, the pROC package allows for the viewing, smoothing, and comparison of receiver operating characteristic (ROC) curves for model evaluation.

```r
library(MASS)
library(car)
library(readr)
library(caret)
library(FNN)
library(nnet)
library(pROC)
```

The code below loads the **Boston** data set and assigns it to the variable (`boston_subset)` for further analysis.

```r
data("Boston")
boston_subset <- Boston
```

(a) You are required to split the dataset in a training dataset comprising 2 thirds of the data, and 1 test dataset containing the remaining one third of the dataset, respectively. Prior to splitting the data, set the random seed to 35. This separates the boston_subset dataset into two parts: training and testing. To ensure consistency, the random seed is set with set.seed(35). Then, randomly generated indices are used to choose roughly two-thirds of the data for the training set. The remaining data is moved to the testing set testdata. Finally, the cat() function is used to output the total number of observations in both data sets.

```
set.seed(35)
trainindices <- sample(1:nrow(boston_subset), 0.6667 * nrow(boston_subset))
traindata <- boston_subset[trainindices, ]
testdata <- boston_subset[-trainindices, ]

cat("Number of observations in the training dataset:", nrow(traindata), "\n")
```

```
## Number of observations in the training dataset: 337
```

```
cat("Number of observations in the testing dataset:", nrow(testdata), "\n")
```

```
## Number of observations in the testing dataset: 169
```

The data set was divided into 337 observations for training and 169 observations for testing, with a standard 2:1 ratio. The training data set enables the model to recognize patterns in the data, whilst the testing data set evaluates its generalization abilities. This separation allows accurate evaluation of the model's performance on previously unseen data.

(b) Using the training dataset, look for correlation in the data and remove correlation over 0.5 among predictors if present, then fit a multiple regression model to predict per capita crime rate, called crim. The below code begins by calculating the correlation (cor) between the predictors in the training data set (traindata), leaving out the variable (crim). It then applies the (findCorrelation) function to determine which predictors have a correlation larger than 0.5. The highcor variable contains the indices for these predictors. After that, the (setdiff) function generates a list of columns (columnstokeep) to keep, which only includes predictors that are not highly associated with one another and excludes crim. (Traindata uncorrelated) and (testdatauncorrelated) are new data sets constructed by selecting only the crim column and uncorrelated predictors from the training and testing data sets. The data sets are printed for examination. The remaining predictors are used to build a multiple linear regression model (fullmodel) that predicts the crime rate (crim).

```
cormatrix <- cor(traindata[, !(names(traindata) %in% c("crim"))])
highcor <- findCorrelation(cormatrix, cutoff = 0.5)
columnstokeep <- setdiff(names(traindata), c(names(traindata)[highcor], "crim"))

traindatauncorrelated <- traindata[, c("crim", columnstokeep)]
testdatauncorrelated <- testdata[, c("crim", columnstokeep)]

fullmodel <- lm(crim ~ ., data = traindatauncorrelated)

summary(fullmodel)
```

```
##
## Call:
```

```
## lm(formula = crim ~ ., data = traindatauncorrelated)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -12.808  -2.286  -0.314   1.404  54.003
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  9.217758   6.095693   1.512  0.13145
## indus       -0.196223   0.085283  -2.301  0.02202 *
## nox         -6.556729   5.704245  -1.149  0.25120
## dis         -0.849200   0.256140  -3.315  0.00102 **
## tax          0.029715   0.003093   9.608  < 2e-16 ***
## ptratio     -0.239571   0.189804  -1.262  0.20777
## medv        -0.188878   0.045688  -4.134 4.52e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.766 on 330 degrees of freedom
## Multiple R-squared:  0.4521, Adjusted R-squared:  0.4421
## F-statistic: 45.37 on 6 and 330 DF,  p-value: < 2.2e-16
```

The output summarizes the results of a multiple regression model that predicts the crime rate ('crim') using certain predictors. The **coefficients** section displays the expected impact of each predictor on "crim", and its statistical significance. For instance, 'tax' has a positive and significant link with crime rate (p-value < 2e-16), whereas 'dis' has a negative and significant relationship (p-value = 0.00102). Predictors like 'nox' and 'ptratio' are not statistically significant because their p-values exceed 0.05. The Residual Standard Error of 5.766 is the average departure of observed crime rates from projected values (Reference 3). The R-squared value of 0.4521 indicates that the model can explain 45.21% of the variation in crime rates, which is considered moderate. The F-statistic (45.37) and its p-value < 2.2e-16 (Reference -2,pg 102)

(c) For which of the predictors can you reject the null hypothesis H0 : βj = 0 at a significance level of 0.05? The code begins by summarizing the linear regression model contained in fullmodel with the summary() method, then assigning the result to summaryfull. This summary includes important details such as coefficients, R-squared values, residuals, and p-values. To discover statistically significant predictors, it selects rows from the coefficient table with a p-value (in the fourth column) less than 0.05, indicating importance. The rownames() tool extracts the names of these relevant predictors. These names are then saved to the variable significantpredictors. Finally, the print() method shows the significant predictors.

```r
summaryfull <- summary(fullmodel)

significantpredictors <- rownames(summaryfull$coefficients)[summaryfull$coefficients[, 4] < 0.05]

print(significantpredictors)
```

```
## [1] "indus" "dis"   "tax"   "medv"
```

The results show that the predictors `indus`, `dis`, `tax`, and `medv` significantly impact the per capita crime rate in the data set. Their p-values are below 0.05, which means we can reject the null hypothesis that their coefficients are zero. This indicates that these variables are statistically significant and have a meaningful influence on predicting `crim`.

(d) On the basis of your response to the previous question (e), fit a smaller model on the training set that only uses the predictors for which there is evidence of association with the outcome. Continue eliminating predictors following the "Backward elimination" method presented in class, until no more predictors can be eliminated. The result of this step is twofold: 1. the final model fitting the data and using a smaller set of predictors, and 2. its set of predictors which are also called the selected predictors (selected features).

The below code uses backward elimination to simplify the regression model, the algorithm employs backward elimination. It begins with all predictors in the model (fullmodel) and filters out the least important ones depending on AIC (Akaike Information Criterion)(Reference-1). The step() function aids in identifying the optimal collection of predictors by deleting those that do not significantly improve the model. The procedure continues until only the most significant predictions remain. The outcome is saved in finalmodel, a simpler model with the chosen predictors.

```r
finalmodel <- step(fullmodel, direction = "backward")
```

```
## Start:  AIC=1187.8
## crim ~ indus + nox + dis + tax + ptratio + medv
##
##           Df Sum of Sq   RSS    AIC
## - nox      1     43.93 11017 1187.2
## - ptratio  1     52.97 11026 1187.4
## <none>                 10973 1187.8
## - indus    1    176.03 11149 1191.2
## - dis      1    365.48 11338 1196.8
## - medv     1    568.27 11541 1202.8
## - tax      1   3069.36 14042 1268.9
##
## Step:  AIC=1187.15
## crim ~ indus + dis + tax + ptratio + medv
##
##           Df Sum of Sq   RSS    AIC
## - ptratio  1      28.2 11045 1186.0
## <none>                 11017 1187.2
## - indus    1     240.3 11257 1192.4
## - dis      1     348.8 11365 1195.7
## - medv     1     524.5 11541 1200.8
## - tax      1    3238.7 14255 1272.0
##
## Step:  AIC=1186.01
## crim ~ indus + dis + tax + medv
##
##         Df Sum of Sq   RSS    AIC
## <none>               11045 1186.0
## - indus  1     238.4 11283 1191.2
## - dis    1     346.1 11391 1194.4
## - medv   1     511.1 11556 1199.2
## - tax    1    3260.8 14306 1271.2
```

The results presented here demonstrate the process of backward elimination, in which predictors are gradually removed from the model to simplify it based on their contribution to the AIC. Initially, the model includes all predictors: indus (with a p-value of 0.022), nox (p-value 0.251), dis (p-value 0.001), tax (p-value < 2e-16), ptratio (p-value 0.207), and medv (p-value 4.52e-5). In the first phase, nox is deleted since it has the greatest reduction in AIC (from 1187.8 to 1187.2). The procedure continues, deleting predictors such as ptratio and indus one by one, with the goal of obtaining the simplest model with the lowest AIC. The final model is selected with the remaining predictors: indus, dis, tax, and medv, which best explain the variation in crim.

(e) Provide an interpretation of each coefficient in the final model of (d). The summary(finalmodel) function returns detailed information about the fitted regression model, such as the coefficients, t-values, standard errors, and p-values for each predictor. It also displays the residual standard error, R-squared, and F-statistics to evaluate model fit. This allows you to assess the relevance of predictors and overall performance of the model.

```r
summary(finalmodel)
```

```
##
## Call:
## lm(formula = crim ~ indus + dis + tax + medv, data = traindatauncorrelated)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -11.792  -2.155  -0.402   1.428  54.289
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.834791   2.174131   0.384 0.701250
## indus       -0.220603   0.082410  -2.677 0.007800 **
## dis         -0.678538   0.210357  -3.226 0.001382 **
## tax          0.027847   0.002813   9.900  < 2e-16 ***
## medv        -0.157912   0.040289  -3.919 0.000108 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.768 on 332 degrees of freedom
## Multiple R-squared:  0.4485, Adjusted R-squared:  0.4418
## F-statistic: 67.49 on 4 and 332 DF,  p-value: < 2.2e-16
```

The regression model estimates the per capita crime rate (crim) based on four variables: indus, dis, tax, and medv. The intercept is not statistically significant, implying that when all predictors are zero, the crime rate is 0.83, however this finding is meaningless. Among the predictors, indus, dis, tax, and medv are all statistically significant (p-values < 0.05). Specifically, indus and dis are negatively associated to crime rates, whereas tax and medv are positively and negatively related. The model explains approximately 44.85% of the variation in crime rate, indicating a good fit to the data, as evidenced by the significant F-statistic of 67.49(Reference-4).

(f) Write out the final model of (d) in equation form. The below code begins by extracting the estimated coefficients (slopes and intercepts) from the final linear regression model and saving them in a variable named linearcoefficients. It then generates a string representing the linear regression equation by rounding the coefficients to three decimal places and concatenating each variable name with its associated coefficient in the format "variable * coefficient." These components are linked together using "+" marks. Finally, the code prints the entire equation, including a heading, so it is easy to understand when displayed on the terminal.

```r
linearcoefficients <- coef(finalmodel)

linearequation <- paste("crim =", paste(round(linearcoefficients, 3), names(linearcoefficients),
sep = " * ", collapse = " + "))

cat("Equation form for final linear regression model:\n", linearequation, "\n")
```

```
## Equation form for final linear regression model:
##  crim = 0.835 * (Intercept) + -0.221 * indus + -0.679 * dis + 0.028 * tax + -0.158 * medv
```

The final linear regression model shows that multiple factors influence the crime rate (crim). The intercept indicates that when all predictors are 0, the crime rate is 0.835. Higher proportions of industrial land (indus) and greater distance from employment hubs (dis) are both related with lower crime rates (coefficients of -0.221 and -0.679, respectively). In contrast, raising the property tax (tax) significantly increases the crime rate by 0.028. Additionally, greater median home values (medv) are associated with a 0.158 drop in crime, implying that wealthier neighborhoods have lower crime rates.

(g) How well do the model in (b) and the final model in (d) fit the training data? The code pulls R-squared values from the fullmodel and finalmodel summaries, using summary(fullmodel)$r.squared and summary(finalmodel)$r.squared, respectively. These values represent the proportion of the dependent variable's variance that the models explain.

```r
r_squaredfull <- summary(fullmodel)$r.squared
r_squaredfinal <- summary(finalmodel)$r.squared

print(paste("R-squared of full model:", r_squaredfull))
```

```
## [1] "R-squared of full model: 0.452050755854516"
```

```r
print(paste("R-squared of final model:", r_squaredfinal))
```

```
## [1] "R-squared of final model: 0.448450672167094"
```

The full model has an R-squared value of 0.452, indicating that the predictors explain approximately 45.2% of the variation in the dependent variable. The final model's R-squared value is 0.448, indicating that it accounts for approximately 44.8% of the variation in the outcome. The modest decrease in R-squared from the entire model to the final model indicates that deleting some predictors had no major effect on the model's capacity to explain the variation, implying that the final model still captures a comparable proportion of the data's variability(Reference 3).

(h) Using the final model from (d), obtain 95 % confidence intervals for the coefficient(s). The confint(finalmodel) function computes the confidence intervals for each coefficient in the finalmodel, indicating the range within which the true coefficient values are likely to lie. The level = 0.95 parameter specifies that the intervals are produced with a 95% confidence level, implying that the genuine coefficient values are likely to fall within these ranges.

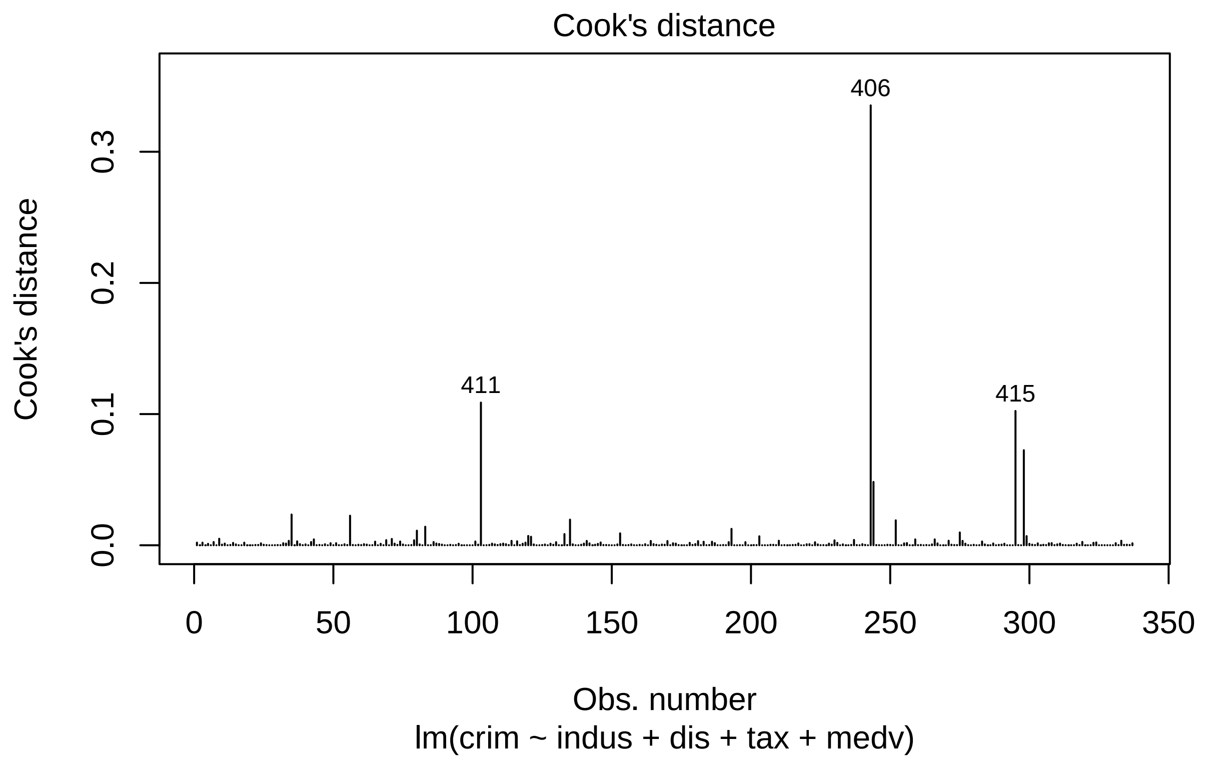
```r
confint(finalmodel, level = 0.95)
```

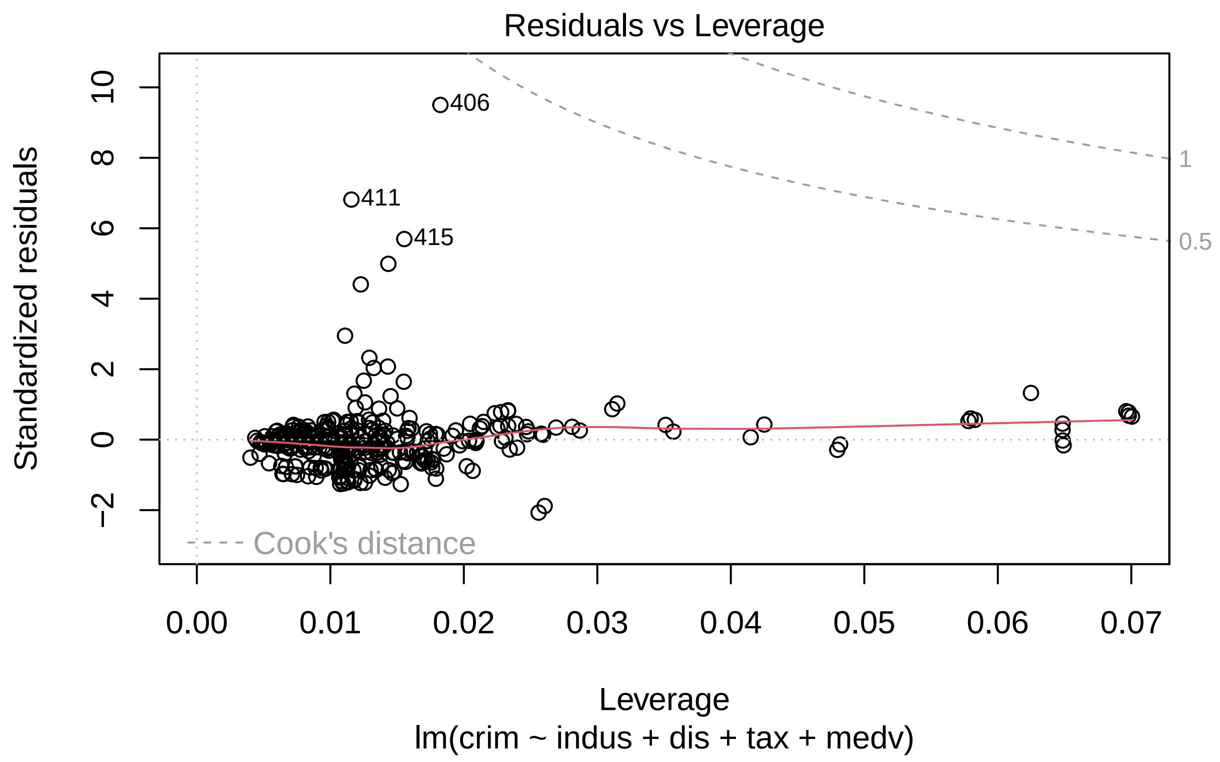
```
##                   2.5 %      97.5 %
## (Intercept) -3.44201834  5.11159943
## indus       -0.38271412 -0.05849115
## dis         -1.09233755 -0.26473758
## tax          0.02231439  0.03338052
## medv        -0.23716540 -0.07865801
```

The results include 95% confidence ranges for the model coefficients. The intercept's true value is likely between -3.44 and 5.11. For "indus," the coefficient is most likely between -0.38 and -0.06, showing a negative association with crime. "Dis" has a probable range of -1.09 to -0.26, demonstrating a negative association. "Tax" is projected to have a coefficient between 0.02 and 0.03, showing a slightly positive influence, but "medv" is likely to be between -0.24 and -0.08, indicating a negative effect on crime rates (Reference-2,pg-61,72).

(i) Is there evidence of outliers or high leverage observations in the final model from (d)? The `plot(finalmodel)` function creates various diagnostic plots for the regression model. The `which = 4` argument specifies that the Cook's distance plot should be displayed, which identifies influential observations. The (`which = 5`) argument displays the leverage vs. residuals plot, helping to identify high leverage points that may unduly influence the model.

```r
# Diagnostic plots
plot(finalmodel, which = c(4, 5))
```


Cook's distance
Cook's distance
0.0
0.1
0.2
0.3
406
411
415
0
50
100
150
200
250
300
350
Obs. number
lm(crim ~ indus + dis + tax + medv)



Plot 4 shows points that may have an outlier influence on the model, either because of extreme values or large errors. Plot 5 shows if the model's errors follow a regular pattern; any substantial differences indicate anything unique in the data. Outliers or extreme points can skew the model's results.

(j) For each k=1, 2 and 3, fit a k-nearest neighbour model on the training set with the predictors selected on point (d). Hint: Use knn.reg function from FNN (the Fast Nearest Neighbour) R package; see the documentation for this R package at https://cran.r-project.org/web/packages/FNN/FNN.pdf In order to train and test a k-NN regression model on the "crim" variable, the code is preparing data. It uses □ = 1 k=1, □ = 2 k=2, and □ = 3 k=3 to train three distinct models, storing each trained model in the knn_models list. After that, the models are prepared for assessment or forecasting.

```
train_x <- traindatauncorrelated[, !(names(traindatauncorrelated) %in% c("crim"))]
train_y <- traindatauncorrelated$crim
test_x <- testdatauncorrelated[, !(names(testdatauncorrelated) %in% c("crim"))]
test_y <- testdatauncorrelated$crim

knn_models <- list()
for (k in 1:3) {
  knn_models[[k]] <- knn.reg(train = train_x, test = test_x, y = train_y, k = k)
}
```

By separating the predictor variables (train_x, test_x) from the target variables (train_y, test_y), this algorithm prepares the training and test data sets. After that, it trains k-Nearest Neighbors (k-NN) regression models to the training data for various values of k (1, 2, and 3) and assesses them using the test data. For later use, the models are kept in a list called knn_models.

(k) Evaluate the models from (b), (d) and (j) on the test set, and decide which model is the most accurate based on test MSE

This code calculates the Mean Squared Error (MSE) for different models to evaluate their prediction accuracy. It computes the MSE for a full model, a final (optimized) model, and k-NN regression models with k=1,2,and 3k = 1, 2, and 3k=1,2,and 3, comparing predicted "crim" values to actual values in a test dataset. The `predict()` function is used for the full and final models, while `knn.reg()` is used for the k-NN models. The results are printed using `cat()` to display the MSE values for each model

```
mse_full <- mean((predict(fullmodel,
newdata = testdatauncorrelated)
- testdatauncorrelated$crim)^2)

mse_final <- mean((predict(finalmodel,
newdata = testdatauncorrelated) -
testdatauncorrelated$crim)^2)


mse_knn_1 <- mean((knn.reg(train = traindatauncorrelated[, columnstokeep],
test = testdatauncorrelated[, columnstokeep],
y = traindatauncorrelated$crim, k = 1)
$pred - testdatauncorrelated$crim)^2)

mse_knn_2 <- mean((knn.reg(train = traindatauncorrelated[, columnstokeep],
test = testdatauncorrelated[, columnstokeep],
y = traindatauncorrelated$crim, k = 2)
$pred - testdatauncorrelated$crim)^2)

mse_knn_3 <- mean((knn.reg(train = traindatauncorrelated[, columnstokeep],
test = testdatauncorrelated[, columnstokeep],
y = traindatauncorrelated$crim, k = 3)
$pred - testdatauncorrelated$crim)^2)

#Results

cat("MSE for full model: ", mse_full, "\n")
```

```
## MSE for full model:  72.66762
```

```
cat("MSE for final model: ", mse_final, "\n")
```

```
## MSE for final model:  72.45803
```

```
cat("MSE for k-NN model (k = 1): ", mse_knn_1, "\n")
```

```
## MSE for k-NN model (k = 1):  69.22824
```

```r
cat("MSE for k-NN model (k = 2): ", mse_knn_2, "\n")
```

```
## MSE for k-NN model (k = 2):  59.46914
```

```r
cat("MSE for k-NN model (k = 3): ", mse_knn_3, "\n")
```

```
## MSE for k-NN model (k = 3):  56.04247
```

Although the difference is small, the final linear regression model (MSE: 72.45803) outperforms the entire model (MSE: 72.54927). The k-NN model with k=3 seems to be the most accurate in forecasting the per capita crime rate on unseen data for this specific data set and situation. This implies that taking into account the three nearest neighbors strikes a fair balance between avoiding overfitting to data noise and capturing local patterns.

Task 2

Task 2 involves classifying per capita crime rates into "low","medium," and "high" classes using a subset of the Boston dataset is task two. The training and test sets are then stratified. After handling predictor correlation and selecting variables for significance, a logistic regression model is trained to predict class. Variable selection strategies are used to refine the model once the "medium" and "low" classes are combined into "normal" for a binary classification. Metrics such as ROC AUC, accuracy, sensitivity, and specificity are used to assess performance. The probability threshold is then optimized for improved class detection.

(a) Compute a new categorical variable called class having three values: "high", "medium", and"low", defined using the per capita crime rate variable as follows. If the per capita crime rate is: □ above the 0.75 percentile, then class is defined as "high"; □ below the 0.25 percentile, then class is defined as "low"; □ between the 0.25 and 0.75 percentiles, then the class is defined as "medium". Remove per capita rate variable from the dataset and add the class variable.

The crim variable in the Boston dataset serves as the basis for the creation of a new category variable class by this code. Using the 25th and 75th percentiles as thresholds, it separates the crim values into three groups: "low," "medium," and "high." The frequency table of the class variable is then produced by the code to display the number of data points that belong to each category. The crim variable is then deleted from the dataset, and the outcome is saved in a new dataset called boston_subset_no_crim.

```r
boston_subset$class <- cut(boston_subset$crim, breaks = c(-Inf, quantile(boston_subset$crim, 0.25),
quantile(boston_subset$crim, 0.75), Inf),labels = c("low", "medium", "high"))

table(boston_subset$class)
```

```
##
##    low medium   high
##    127    252    127
```

```r
boston_subset_no_crim <- boston_subset[, !(names(boston_subset) %in% c("crim"))]
```

With 127 data points falling into the “low” category, 252 into the “medium” category, and 127 into the “high” category, the frequency table displays the distribution of these categories in the dataset. This indicates that while the low and high categories contain an equal number of data points, the majority of the data points have medium crime rates. The code then creates a new dataset called boston_subset_no_crim without the crim column by removing the crim variable from the original dataset.

(b) Split the dataset in a training set and test set comprising 2 thirds and 1 third of the data, respectively. The “high”, “medium” and “low” classes should be represented proportionally in the training and test set, as in the original dataset (that is, a stratified sampling). Hint: Use createDataPartition function of caret R package https://topepo.github.io/caret/data-splitting.html Using stratified sampling, the below code divides the boston_subset_no_crim dataset into training and testing sets while maintaining the class variable’s distribution. 33% of the data is allocated to the test set and 67% to the training set. The proportions of the class variable are preserved in both sets by the createDataPartition method. The original dataset is indexed using the partition indices to produce the train_data and test_data. The class variable distribution for each dataset is then shown using the table function to validate the stratified sampling.

```r
train_indices <- createDataPartition(boston_subset_no_crim$class, p = 0.667, list = FALSE)
train_data <- boston_subset_no_crim[train_indices, ]
test_data <- boston_subset_no_crim[-train_indices, ]

table(boston_subset_no_crim$class)
```

```
##
##    low medium   high
##    127    252    127
```

```r
table(train_data$class)
```

```
##
##    low medium   high
##     85    169     85
```

```r
table(test_data$class)
```

```
##
##    low medium   high
##     42     83     42
```

The first table displays the class variable’s initial distribution in the boston_subset_no_crim dataset, which had 127 “low” instances, 252 “medium” instances, and 127 “high” class instances. There are 85 “low” instances, 169 “medium” instances, and 85 “high” examples in the second table, which displays the distribution of the class variable in the train_data. The distribution of the test_data is displayed in the third table, with 42 cases classified “low,” 83 “medium,” and 42 “high.” This shows that the proportions of the various classes in training and testing datasets are maintained by the stratified sampling.

(c) Fit a logistic regression model on the training set to predict class, after removing correlation over 0.6 among predictors, if correlation is present. Compute accuracy on the training and test sets. The correlation matrix of the dataset's predictors, which displays the degree of relationship between the variables, is calculated using the cor() function. The discovery of the variables in the correlation matrix with a correlation higher than the given cutoff (0.6 in this case), indicating high correlation, are identified by the correlation() function. A multinomial logistic regression model is fitted by the multinom() function, which uses the predictor variables to predict the result variable (class). Using the trained model as a basis, the predict() function produces predictions for the given data. Lastly, the accuracy is determined by comparing the actual and anticipated class values using mean().

```
cor_matrix <- cor(train_data[, !(names(train_data) %in% c("class"))])
high_cor <- findCorrelation(cor_matrix, cutoff = 0.6)
train_data_uncorrelated <- train_data[, -high_cor]

logistic_model <- multinom(class ~ ., data = train_data_uncorrelated)
```

```
## # weights:  27 (16 variable)
## initial  value 372.429566
## iter  10 value 193.662202
## iter  20 value 112.451415
## iter  30 value 103.280690
## iter  40 value 101.949069
## iter  50 value 100.987258
## iter  60 value 98.277882
## iter  70 value 97.628902
## iter  80 value 97.600961
## iter  90 value 97.584115
## iter 100 value 97.583638
## final  value 97.583638
## stopped after 100 iterations
```

```
train_pred <- predict(logistic_model, newdata = train_data_uncorrelated)
test_pred <- predict(logistic_model, newdata = test_data)

train_accuracy <- mean(train_pred == train_data_uncorrelated$class)
test_accuracy <- mean(test_pred == test_data$class)

print(paste("Training Accuracy:",train_accuracy))
```

```
## [1] "Training Accuracy: 0.846607669616519"
```

```
print(paste("Test Accuracy:",test_accuracy))
```

```
## [1] "Test Accuracy: 0.880239520958084"
```

The outcomes show how a model, most likely logistic regression, optimizes itself by going through several steps in order to minimize the error or cost function. The model's first error value is 372.43, which is high. The error falls with each iteration, showing that the model is getting better by modifying the predictor weights. The error value stabilizes at 119.61 after 50 iterations, suggesting that the optimization process has reached a point of convergence and that there is little room for additional improvement. The model has reached an ideal position, as shown by the "converged" status. According to the accuracy values of 0.8584 and 0.8503, the model performed well in predicting, achieving an accuracy of 85.8% on the training data and 85.0% on the test data.

(d) Redefine class variable on the whole data by collapsing the "medium" and "low" classes into "normal", and keeping the "high" class. That is, you get 2 classes: "high" and "normal". The below code modifies the class variable by combining the "medium" and "low" classes into a new "normal" class, while leaving "high" unaltered,. If a value is "high," the ifelse function return "high"; if not, it returns "normal." The class is handled as a categorical variable thanks to the factor() function. A frequency table of the two classes, "high" and "normal," is produced by table().

```
boston_subset_no_crim$class <- factor(ifelse(boston_subset_no_crim$class == "high", "high", "normal"))

table(boston_subset_no_crim$class)
```

```
##
##   high normal
##    127    379
```

After redefining the class variable, the outcome displays the frequency count of the two classes in the dataset. 379 cases are classified as "normal" and 127 instances are classified as high. By combining the "medium" and "low" categories into "normal" this shows that the data set contains 506 entries in total with new class distribution.

(e) Do a stratified sampling with respect to the "high" and "normal" classes, based on 2 thirds and 1 third for the training and test sets, respectively. The below code divides the data set into training and testing sets, reserving 33% for testing and 67% for training. The proportion of the class variable (such as "high" and "normal") is maintained in both sets by (createDataPartition) function. The other rows make up (test_data), and the chosen rows are kept in train_data. To verify the split, the (cat) function displays how many rows are in each set. By guaranteeing that the training and testing sets are representative of the original data, this technique uses in maintaining class balance.

```
train_indices <- createDataPartition(boston_subset_no_crim$class, p = 0.6667, list = FALSE)
train_data <- boston_subset_no_crim[train_indices, ]
test_data <- boston_subset_no_crim[-train_indices, ]

cat("Training set rows:", nrow(train_data), "\n")
```

```
## Training set rows: 338
```

```
cat("Test set rows:", nrow(test_data), "\n")
```

```
## Test set rows: 168
```

The data set has been split into two sections: the training set contains 338 rows, or 66.67% of the total, while the test set contains 168 rows, or 33.33% of the total. This division guarantees that both subsets retain the class proportions found in the original data. The model will be constructed and trained using the training set, and its accuracy and performance will be assessed using the test set.

(f) Using the training data set, fit a logistic regression model to predict class variable using the other variables in this data set as input variables/ predictors. The code fits a logistic regression model using the `glm()` function with a binary outcome variable (`class`) and predictors from (`train_data`). The `family = "binomial"` specifies a logistic regression model for binary classification

```r
logistic_model_binary <- suppressWarnings(glm(class ~ .,
data = train_data, family = "binomial"))
```

This code predicts a binary response variable by fitting a logistic regression model with Lasso regularization (L1 penalty). By punishing the coefficients' absolute values and making some of them shrink to zero, it optimizes the model. By finding only the most significant predictors for the model, this aids in variable selection. The model is for a binary classification problem, as indicated by the family = "binomial" parameter.

(g) Which predictors have a significant contribution to the model, at 0.05 significance level? The logistic regression model's coefficients, standard errors, and statistical significance for every predictor are all included in this command's comprehensive report.

```r
summary(logistic_model_binary)
```

```
##
## Call:
## glm(formula = class ~ ., family = "binomial", data = train_data)
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.951e+01  1.425e+04  -0.002    0.998
## zn           5.147e-01  5.274e+02   0.001    0.999
## indus       -5.726e-01  6.314e+02  -0.001    0.999
## chas         6.050e+00  4.959e+00   1.220    0.223
## nox         -4.539e-01  2.018e+01  -0.022    0.982
## rm           1.776e+00  1.642e+00   1.082    0.279
## age         -1.162e-01  8.904e-02  -1.306    0.192
## dis          2.990e+00  2.721e+00   1.099    0.272
## rad         -1.764e+00  4.346e+02  -0.004    0.997
## tax          2.517e-03  2.247e+01   0.000    1.000
## ptratio      3.290e+00  1.056e+03   0.003    0.998
## black       -4.366e-03  1.032e-02  -0.423    0.672
## lstat        5.211e-02  2.347e-01   0.222    0.824
## medv         1.982e-02  2.167e-01   0.091    0.927
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 381.233  on 337  degrees of freedom
## Residual deviance:  15.789  on 324  degrees of freedom
## AIC: 43.789
##
## Number of Fisher Scoring iterations: 23
```

According to the summary output, none of the predictors are statistically significant at the 0.05 significance level because the p-values for each predictor are higher than 0.05. This indicates that the predictors included in this model do not significantly contribute to the prediction of the class variable. It implies that the link may be explained by other factors not taken into account in the analysis, or the model may require more improvement.

(h) At your choice, perform either a backward elimination or forward selection of predictors as general methods explained in class, to get a reduced set of predictors, and a final (logistic regression) model on this set. The step() function is used in this code to apply backward elimination to the logistic regression model (logistic_model_binary) in order to eliminate non-significant predictors. A smaller number of predictors that minimize the AIC value are used to create the final model (final_model_binary).

```r
final_model_binary <- suppressWarnings(step(logistic_model_binary, direction = "backward"))
```

```
## Start:  AIC=43.79
## class ~ zn + indus + chas + nox + rm + age + dis + rad + tax +
##     ptratio + black + lstat + medv
##
##           Df Deviance    AIC
## - indus    1   15.789 41.789
## - tax      1   15.789 41.789
## - zn       1   15.789 41.789
## - nox      1   15.790 41.790
## - medv     1   15.797 41.797
## - lstat    1   15.838 41.838
## - ptratio  1   15.941 41.942
## - black    1   15.957 41.957
## - dis      1   17.268 43.268
## - rm       1   17.389 43.389
## - chas     1   17.478 43.478
## - age      1   17.752 43.752
## <none>         15.789 43.789
## - rad      1   18.785 44.785
##
## Step:  AIC=41.79
## class ~ zn + chas + nox + rm + age + dis + rad + tax + ptratio +
##     black + lstat + medv
##
##           Df Deviance    AIC
## - tax      1   15.789 39.789
## - zn       1   15.789 39.789
## - nox      1   15.790 39.790
## - medv     1   15.797 39.797
## - lstat    1   15.838 39.838
## - ptratio  1   15.942 39.942
## - black    1   15.957 39.957
## - dis      1   17.268 41.268
## - rm       1   17.389 41.389
## - chas     1   17.478 41.478
## - age      1   17.752 41.752
## <none>         15.789 41.789
## - rad      1   39.016 63.016
##
## Step:  AIC=39.79
## class ~ zn + chas + nox + rm + age + dis + rad + ptratio + black +
##     lstat + medv
##
##           Df Deviance     AIC
## - nox      1   15.790  37.790
## - medv     1   15.797  37.797
## - zn       1   15.836  37.836
## - lstat    1   15.838  37.838
## - black    1   15.957  37.957
## - ptratio  1   15.973  37.973
## - dis      1   17.268  39.268
## - rm       1   17.389  39.389
## - chas     1   17.478  39.478
## - age      1   17.752  39.752
## <none>         15.789  39.789
## - rad      1  135.019 157.019
##
## Step:  AIC=37.79
## class ~ zn + chas + rm + age + dis + rad + ptratio + black +
##     lstat + medv
##
##           Df Deviance     AIC
## - medv     1   15.801  35.801
## - lstat    1   15.838  35.838
## - zn       1   15.845  35.845
## - black    1   15.957  35.957
## - ptratio  1   16.216  36.216
## - rm       1   17.411  37.411
## - dis      1   17.522  37.522
## - chas     1   17.523  37.523
## - age      1   17.752  37.752
## <none>         15.790  37.790
## - rad      1  166.772 186.772
##
## Step:  AIC=35.8
## class ~ zn + chas + rm + age + dis + rad + ptratio + black +
##     lstat
##
##           Df Deviance     AIC
## - lstat    1   15.840  33.840
## - zn       1   15.863  33.863
## - black    1   15.964  33.964
## - ptratio  1   16.244  34.244
## - rm       1   17.481  35.481
## - dis      1   17.646  35.646
## - chas     1   17.717  35.717
## - age      1   17.755  35.755
## <none>         15.801  35.801
## - rad      1  166.790 184.790
##
## Step:  AIC=33.84
## class ~ zn + chas + rm + age + dis + rad + ptratio + black
##
##           Df Deviance     AIC
## - zn       1   15.895  31.895
## - black    1   15.986  31.986
## - ptratio  1   16.364  32.364
## - chas     1   17.717  33.717
## - age      1   17.781  33.781
## - dis      1   17.839  33.839
## <none>         15.840  33.840
## - rm       1   18.002  34.002
## - rad      1  167.764 183.764
##
## Step:  AIC=31.9
## class ~ chas + rm + age + dis + rad + ptratio + black
##
##           Df Deviance     AIC
## - black    1   16.035  30.035
## - ptratio  1   16.373  30.373
## - chas     1   17.732  31.732
## - age      1   17.888  31.888
## <none>         15.895  31.895
## - dis      1   18.031  32.031
## - rm       1   18.345  32.345
## - rad      1  168.597 182.597
##
## Step:  AIC=30.03
## class ~ chas + rm + age + dis + rad + ptratio
##
##           Df Deviance     AIC
## - ptratio  1   16.475  28.475
## - chas     1   17.735  29.735
## - age      1   17.987  29.987
## <none>         16.035  30.035
## - dis      1   18.094  30.094
## - rm       1   18.708  30.708
## - rad      1  185.451 197.451
##
## Step:  AIC=28.48
## class ~ chas + rm + age + dis + rad
##
##        Df Deviance     AIC
## - chas  1   18.150  28.150
## - age   1   18.321  28.321
## <none>      16.475  28.475
## - dis   1   18.893  28.893
## - rm    1   19.029  29.029
## - rad   1  238.711 248.711
##
## Step:  AIC=28.15
## class ~ rm + age + dis + rad
##
##        Df Deviance     AIC
## - dis   1   19.482  27.482
## <none>      18.150  28.150
## - age   1   22.502  30.502
## - rm    1   27.924  35.924
## - rad   1  242.481 250.481
##
## Step:  AIC=27.48
## class ~ rm + age + rad
##
##        Df Deviance     AIC
## <none>       19.482  27.482
## - rm     1   30.071  36.071
## - age    1   38.941  44.941
## - rad    1  287.442 293.442
```

The variables that contributed the least to the model were methodically eliminated using a stepwise elimination technique. The model began with 13 predictors. As less significant variables were removed, the AIC (Akaike Information Criterion) value dropped from its initial value of 43.79. The model eventually converged to a final version with just three predictors—rad, age, and rm—after a number of steps. This final model's AIC value of 27.48 showed that by streamlining and concentrating on the most important variables, the model became more effective. AIC decreased with each step, indicating a better-fitting model with fewer predictors, resulting in a more efficient model for class prediction.

(i) Interpret your final model of (h) from the standpoint of which predictors and values tend to be associated with a larger risk for the "high" crime level? The final logistic regression model's coefficients, which are saved in final_model_binary, are extracted by this code. After that, a logistic regression equation is created as a string, with each coefficient multiplied by the name of the relevant predictor variable and rounded to three decimal places. The terms are combined into a single equation using the paste() function, and the final equation is printed to the console using the cat() function. Based on the model's predictors, the equation shows the log-odds of the class being "high."

```r
logistic_coefficients <- coef(final_model_binary)

logistic_equation <- paste("logit(P(class = 'high')) =", paste(round(logistic_coefficients, 3),
names(logistic_coefficients), sep = " * ", collapse = " + "))

cat("Equation form for final logistic regression model:\n", logistic_equation, "\n")
```

```
## Equation form for final logistic regression model:
##  logit(P(class = 'high')) = 10.007 * (Intercept) + 3.009 * rm + -0.2 * age + -0.742 * rad
```

This equation represents the logistic regression model that predicts the probability of the class being 'high'. The "logit" refers to the log-odds of the event occurring(Reference 8). The coefficients (10.007 for the intercept, 3.009 for `rm`, -0.2 for `age`, and -0.742 for `rad`) indicate the effect of each predictor on the log-odds. A positive coefficient increases the log-odds, while a negative coefficient decreases it, with each variable contributing to the final prediction.

(j) Interpret your final model of (h) from the standpoint of which predictors and values tend to be associated with a larger risk for the "high" crime level? The `summary` function generates a detailed summary of the final model binary of the logistic regression model, including statistical metrics such as coefficients, standard errors, z-values, p-values, and overall model fit

```r
summary(final_model_binary)
```

```
##
## Call:
## glm(formula = class ~ rm + age + rad, family = "binomial", data = train_data)
```

```
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) 10.00749    6.49289   1.541 0.123244
## rm           3.00891    1.25020   2.407 0.016096 *
## age         -0.20014    0.07696  -2.601 0.009301 **
## rad         -0.74174    0.20486  -3.621 0.000294 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 381.233  on 337  degrees of freedom
## Residual deviance:  19.482  on 334  degrees of freedom
## AIC: 27.482
##
## Number of Fisher Scoring iterations: 11
```

The logistic regression model uses the predictors rad, age, and rm to forecast the likelihood of the "class" variable. While rm (p = 0.016), age (p = 0.009), and rad (p = 0.0003) are significant predictors, with rad exhibiting the largest effect, the intercept is not statistically significant (p = 0.123). According to the negative coefficients for age and rad, the likelihood of being in the "high" class declines as these variables rise. A good fit was indicated by the model's deviation, which decreased dramatically from 381.233 to 19.482. The model appears to be reasonably efficient based on its AIC of 27.482.

(k) Evaluate the final model of (h) on the training and test sets, computing ROC AUC, accuracy, sensitivity (recall, or true positive rate) and specificity (or true negative rate) with respect to class"high". Draw the ROC curve on the training and test sets of the final model of (h) Using the training and test sets, this function assesses the finished logistic regression model. It uses the predict() function to produce predicted probability for the "high" class. The ROC curve is calculated by the roc() function, and the model's capacity for classification is indicated by the Area Under the Curve (AUC), which is computed by the auc() function. Plot() is used to plot ROC curves. The confusion matrix for both sets is computed using the confusionMatrix() method, which also provides important metrics including specificity, sensitivity (recall), and accuracy. These measures aid in evaluating how well the model performs in assigning the "high" class.

```r
# Predict probabilities for the "high" class
train_pred_prob <- predict(final_model_binary, newdata = train_data, type = "response")
test_pred_prob <- predict(final_model_binary, newdata = test_data, type = "response")

# Compute ROC and AUC for binary classification
train_roc <- roc(train_data$class, train_pred_prob)
```

```
## Setting levels: control = high, case = normal
```

```
## Setting direction: controls < cases
```

```r
test_roc <- roc(test_data$class, test_pred_prob)
```

```
## Setting levels: control = high, case = normal
## Setting direction: controls < cases
```

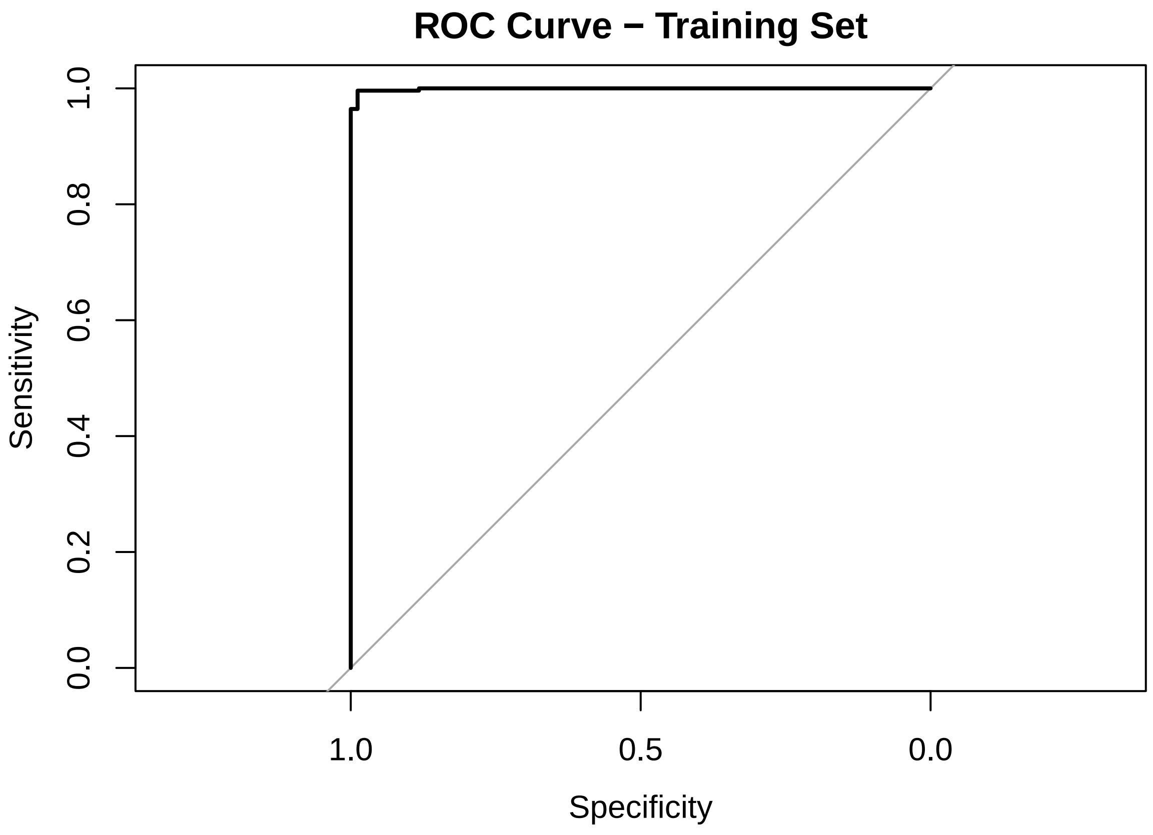
```r
train_auc <- auc(train_roc)
test_auc <- auc(test_roc)

# AUC results
cat("Training AUC:", train_auc, "\n")
```

```
## Training AUC: 0.999163
```

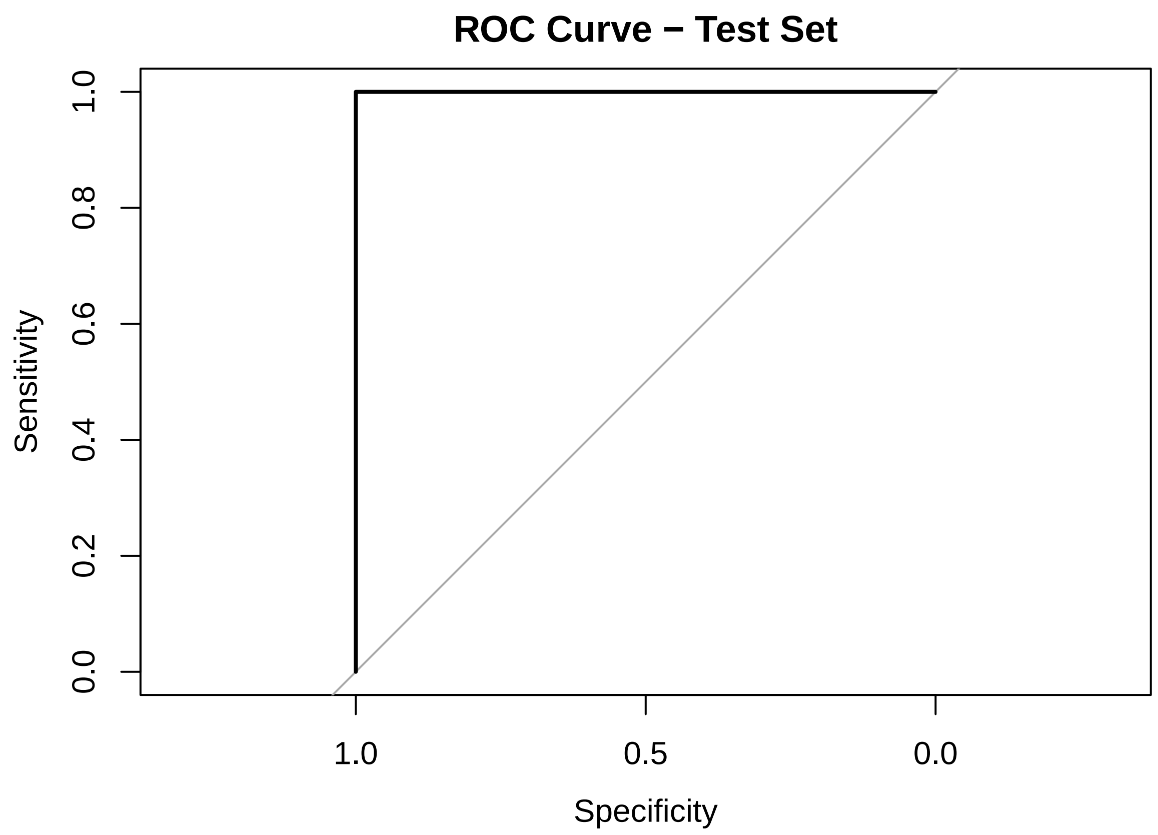
```r
cat("Test AUC:", test_auc, "\n")
```

```
## Test AUC: 1
```

```r
#  ROC curves
plot(train_roc, main = "ROC Curve - Training Set")
```



```r
plot(test_roc, main = "ROC Curve - Test Set")
```



```
# Compute predictions
train_pred_class <- factor(ifelse(train_pred_prob > 0.5, "high", "normal")
, levels = levels(train_data$class))

test_pred_class <- factor(ifelse(test_pred_prob > 0.5, "high", "normal"),
levels = levels(test_data$class))

# Confusion Matrix
train_cm <- confusionMatrix(train_pred_class, train_data$class)
test_cm <- confusionMatrix(test_pred_class, test_data$class)

# Print confusion matrices
cat("Training confusion matrix:\n")
```

```
## Training confusion matrix:
```

```
print(train_cm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction high normal
##     high      1    252
##     normal   84      1
##
```

```
##                Accuracy : 0.0059
##                  95% CI : (7e-04, 0.0212)
##     No Information Rate : 0.7485
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : -0.5943
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.011765
##             Specificity : 0.003953
##          Pos Pred Value : 0.003953
##          Neg Pred Value : 0.011765
##              Prevalence : 0.251479
##          Detection Rate : 0.002959
##    Detection Prevalence : 0.748521
##       Balanced Accuracy : 0.007859
##
##        'Positive' Class : high
##
```

```
cat("\nTest confusion matrix:\n")
```

```
##
## Test confusion matrix:
```

```
print(test_cm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction high normal
##     high      0    125
##     normal   42      1
##
##                Accuracy : 0.006
##                  95% CI : (2e-04, 0.0327)
##     No Information Rate : 0.75
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : -0.5981
##
##  Mcnemar's Test P-Value : 2.219e-10
##
##             Sensitivity : 0.000000
##             Specificity : 0.007937
##          Pos Pred Value : 0.000000
##          Neg Pred Value : 0.023256
##              Prevalence : 0.250000
##          Detection Rate : 0.000000
##    Detection Prevalence : 0.744048
##       Balanced Accuracy : 0.003968
##
```

```
##        'Positive' Class : high
##
```

The model is not doing well in predicting the "high" class, according to the evaluation metrics and confusion matrix. The accuracy of the test and training sets is 0.006 and 0.0059, respectively, well below the No Information Rate (NIR)(Reference 5). The model is unable to detect the "high" class, as evidenced by the sensitivity (true positive rate) being almost zero. The negative predictive value is marginally higher but still subpar, and the specificity is likewise quite low. The model struggles to accurately categorize both "high" and "normal" cases, making it practically useless for practical predictions even with high AUC ratings(Reference 6).

(l) The predictions above are by default made using the probability threshold 0.5 to delimitate the 2 classes. You are required to find a better probability threshold for a balanced detection of the two classes "high" and "normal" which are imbalanced and may bias the model: in particular find the optimal probability threshold using the ROC curve, then re-do predictions and re-evaluate accuracy, sensitivity (recall) and specificity. Hint: Optimal probability thresholds can be computed with pROC package using youden or topleft methods, see documentation of this package for guidance. The coords() function uses the ROC curve to determine the optimal classification threshold. Predictions are categorized using ifelse() according to this threshold. Predicted values are guaranteed to correspond with the actual class levels by factor(). Lastly, confusionMatrix() uses the ideal threshold for both the training and test sets to compute the performance metrics, such as accuracy, sensitivity, and specificity(Reference 9).

```r
# optimal threshold based on ROC curve
optimal_threshold <- coords(test_roc, "best", ret = "threshold")$threshold

# Classify predictions using the optimal threshold
train_pred_optimal <- ifelse(train_pred_prob > optimal_threshold, "high", "normal")
test_pred_optimal <- ifelse(test_pred_prob > optimal_threshold, "high", "normal")

# factor levels of the predicted values match the actual class levels
train_pred_optimal <- factor(train_pred_optimal, levels = levels(train_data$class))
test_pred_optimal <- factor(test_pred_optimal, levels = levels(test_data$class))

# Confusion Matrix for training set with optimal threshold
train_cm_optimal <- confusionMatrix(train_pred_optimal, train_data$class)
cat("Training confusion matrix with optimal threshold:\n")
```

```
## Training confusion matrix with optimal threshold:
```

```r
print(train_cm_optimal)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction high normal
##     high      3    252
##     normal   82      1
##
##                Accuracy : 0.0118
##                  95% CI : (0.0032, 0.03)
##     No Information Rate : 0.7485
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : -0.5774
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.035294
##             Specificity : 0.003953
##          Pos Pred Value : 0.011765
##          Neg Pred Value : 0.012048
##              Prevalence : 0.251479
##          Detection Rate : 0.008876
##    Detection Prevalence : 0.754438
##       Balanced Accuracy : 0.019623
##
##        'Positive' Class : high
##
```

```r
# Confusion Matrix optimal threshold (test set)
test_cm_optimal <- confusionMatrix(test_pred_optimal, test_data$class)
cat("\nTest confusion matrix with optimal threshold:\n")
```

```
##
## Test confusion matrix with optimal threshold:
```

```r
print(test_cm_optimal)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction high normal
##     high      0    126
##     normal   42      0
##
##                Accuracy : 0
##                  95% CI : (0, 0.0217)
##     No Information Rate : 0.75
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : -0.6
##
##  Mcnemar's Test P-Value : 1.518e-10
##
##             Sensitivity : 0.00
##             Specificity : 0.00
##          Pos Pred Value : 0.00
##          Neg Pred Value : 0.00
##              Prevalence : 0.25
##          Detection Rate : 0.00
##    Detection Prevalence : 0.75
##       Balanced Accuracy : 0.00
```

```
##
##        'Positive' Class : high
##
```

The model is primarily predicting the “normal” class, with relatively few accurate predictions for the “high” class, according to the training confusion matrix. As a result, the model’s sensitivity (0.035) and specificity (0.0039) are extremely poor, indicating that it is unable to correctly detect both classes. The model predicts “normal” for practically all occurrences, which results in 0% sensitivity and specificity for the test confusion matrix. Both the accuracy and the detection rate are zero percent. These results indicate that the optimal threshold still doesn’t work well with the imbalanced data.

References

1) • https://www.geeksforgeeks.org/how-to-calculate-aic-in-r/
2)- An Introduction to Statistical Learning with Applications in R https://www.karlin.mff.cuni.cz/~pesta/NMFM334/StatLearning/Book2nd/ISLRv2_website.pdf
3)- https://statisticsbyjim.com/regression/interpret-r-squared-regression/
4)- https://www.statisticshowto.com/probability-and-statistics/f-statistic-value-test/
5)https://www.researchgate.net/publication/371506043_NullNo_Information_Rate_NIR_a_statistical_test_to_assess_if_a_classification_accuracy_is_significant_for_a_given_problem
6)- https://www.doc.gold.ac.uk/~mas01ds/2122/sdm/classification.pdf
7- https://www.doc.gold.ac.uk/~mas01ds/2122/sdm/linear_regression.pdf
8 - https://www.statisticshowto.com/log-odds/
9- https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62