Liver Disease Classification

Nabeel Khan

27-May-2020

Contents

1 Introduction

This project is part of the 'HarvardX: PH125.9x Data Science: Capstone' course. In this project, we chose a dataset of our choice and develop machine learning models to perform binary classification to diagnose liver disease.

1.1 Background

The liver plays a vital role in keeping us healthy. The liver's main job is to filter the blood coming from the digestive tract before passing it to the rest of the body. The liver also turns nutrients into chemicals our body needs, converts food into energy, and filters out poisons. The damage to the liver affects the whole body.

The patients with liver disease are on the rise because of excessive consumption of alcohol, inhale of harmful gases, or intake of contaminated food. The problems with the liver are not easily discovered in an early stage. Moreover, the diagnosis of liver damage is subjective and varies from doctor to doctor based on experience. The initial diagnosis of liver disease increases the survival rate of patients. The liver disease can be detected by analysing the levels of enzymes in the human blood [2, 3]. Therefore, a classification algorithm capable of automatically detecting the liver disease can assist the doctors in diagnosing liver disease. The classification techniques are commonly used in various automatic medical diagnoses tools[1].

1.2 Aim of Project

This project aims to develop a binary classifier, which can use blood enzymes information to diagnose liver disease. That information will assist doctors in conducting further testing of patients for additional verification and start treatment in time to save lives.

2 Dataset and Evaluation Metrics

We use the liver patient records, which are collected from North East of Andhra Pradesh, India. The data set contains:

1. 416 liver patient records and 167 non-liver patient records.

2.1 Download Data

The dataset is publically available online both at Kaggle and UCI repository. We download data from the website. Then, we split data into training and validation sets.

- 10% of the data is used for validation, and 90

```
################################
#  Install packages (if not installed)
################################
repos_path<- "http://cran.us.r-project.org"
if(!require(tidyverse)) install.packages("tidyverse", repos =repos_path)
```

```r
if(!require(caret)) install.packages("caret", repos = repos_path)
if(!require(data.table)) install.packages("data.table", repos =repos_path)
if(!require(lubridate)) install.packages("lubridate", repos = repos_path)
if(!require(dplyr)) install.packages("dplyr", repos = repos_path)
if(!require(sjmisc)) install.packages("dplyr", repos = repos_path)
if(!require(scales)) install.packages("scales", repos = repos_path)
if(!require(caret)) install.packages("caret", repos = repos_path)
if(!require(gam)) install.packages("gam", repos = repos_path)

################################
# Load libraries
################################
library(lubridate)
library(tidyverse)
library(dplyr)
library(lubridate)
library(sjmisc)
library(scales)
library(caret)
library(gam)
```

```r
################################
# Downloading data
################################
# Indian Live Patient Records :
 # https://www.kaggle.com/uciml/indian-liver-patient-records/
 # https://archive.ics.uci.edu/ml/machine-learning-databases/00225/Indian Liver Patient Dataset (ILPD).

url <- "https://archive.ics.uci.edu/ml/machine-learning-databases/00225/Indian Liver Patient Dataset (I

# Download csv
liverData <- read.csv(url)

# Rename columns of csv (follow the webpage for naming convention)
colnames(liverData)<- c("Age","Gender","Total_Bilirubin","Direct_Bilirubin", "Alkaline_Phosphotase","Ala

################################
# Creating training and validation sets
################################

# Validation set will be 10% of whole data
set.seed(1)
test_index <- createDataPartition(y = liverData$Dataset, times = 1, p = 0.1, list = FALSE)

training <- liverData[-test_index,]
validation <- liverData[test_index,]

 # Removing the objects from environment as no longer required
rm(liverData)
```

Both training and validation datasets have a distribution of patients with and with no disease.

```r
# Looking at distributions of liver disease
# 1 indicates that the liver is damage. While 2 means that the liver is healthy
print("Training Data")
```

```
## [1] "Training Data"
```

```r
table(training$Dataset)
```

```
##
##   1   2
## 374 149
```

```r
print("Validation Data")
```

```
## [1] "Validation Data"
```

```r
table(validation$Dataset)
```

```
##
##  1  2
## 41 18
```

2.2 Metrics

To evaluate the performance of classifiers, we will use the following metrics:

1. **Accuracy** It is the ratio of the number of correct predictions to the total number of input samples.

$$Accuracy = \frac{TruePositives + TrueNegatives}{TotalPredictions} \tag{1}$$

2. **Precision** It is defined as the proportion of the true positives against all the positive results.

$$Precision = \frac{NumberofTruePositives}{NumberofTruePositives + NumberofFalsePositives} \tag{2}$$

3. **Sensitivity** It is also referred as true positive rate or recall. It is the proportion of true positives that are correctly identified.

$$Sensitivity = \frac{NumberofTruePositives}{NumberofTruePositives + NumberofFalseNegatives} \tag{3}$$

4. **Specificity** It is the true negative rate. It is the proportion of true negatives that are correctly identified.

$$Specificity = \frac{NumberofTrueNegatives}{NumberofTrueNegatives + NumberofFalsePositives} \tag{4}$$

5. **F1 Score** F1 score is a function of Precision and Recall. F1 Score is a better measure to use if we need to seek a balance between Precision and Recall. And, there is an uneven class distribution.

$$F1Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{5}$$

6. **Cohen's Kappa** Cohen's Kappa (or simple Kappa) measures inter-rater reliability (sometimes called interobserver agreement). It is a measure of agreement between the two individuals. The kappa statistic measures the percentage of data values in the main diagonal of the table and then adjusts the values for the amount of agreement that could be expected due to chance alone.

The values of all metrics range from 0 to 1. Higher the value better is the metric.

3 Data Exploration

The dataset contains 11 variables, namely, "Age", "Gender'', "Total_Bilirubin", or "Alkaline_Phosphotase". The 'Dataset' variable indicates if the liver has a disease or not. For instance, a value of 1 means that the liver is damaged, while a value of 2 means that the liver is healthy.

All other variables except "Age", "Gender", and "Dataset" represent the amount of enzymes or proteins in the blood. These variables (or a subset) will be used to train our machine learning models to make diagnoses.

```
head(training)
```

Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Ar
62	Male	10.9	5.5	699	64	
62	Male	7.3	4.1	490	60	
58	Male	1.0	0.4	182	14	
72	Male	3.9	2.0	195	27	
46	Male	1.8	0.7	208	19	
26	Female	0.9	0.2	154	16	

The training dataset has 523 records. We can see that the "Albumin_and_Globulin_Ratio" variable has 4 null values. The remaining variables do not contain any null values.

- The validation data also has no null values (confirmed via summary).

```
sprintf("Rows of training dataset = %d", nrow(training))
```

```
## [1] "Rows of training dataset = 523"
```

```
print("=========================")
```

```
## [1] "========================="
```

```
summary(training)
```

```
##       Age           Gender    Total_Bilirubin  Direct_Bilirubin
##  Min.   : 4.00   Female:123   Min.   : 0.400   Min.   : 0.100
##  1st Qu.:33.00   Male  :400   1st Qu.: 0.800   1st Qu.: 0.200
##  Median :45.00                Median : 1.000   Median : 0.300
##  Mean   :44.68                Mean   : 3.314   Mean   : 1.492
##  3rd Qu.:57.50                3rd Qu.: 2.550   3rd Qu.: 1.200
```

```
##  Max.   :90.00                Max.   :75.000   Max.   :19.700
##
##  Alkaline_Phosphotase Alamine_Aminotransferase Aspartate_Aminotransferase
##  Min.   :  63.0       Min.   :  10.00          Min.   :  11.0
##  1st Qu.: 175.5       1st Qu.:  24.00          1st Qu.:  25.0
##  Median : 209.0       Median :  35.00          Median :  41.0
##  Mean   : 295.0       Mean   :  83.48          Mean   : 113.8
##  3rd Qu.: 298.0       3rd Qu.:  61.00          3rd Qu.:  87.0
##  Max.   :2110.0       Max.   :2000.00          Max.   :4929.0
##
##  Total_Protiens    Albumin      Albumin_and_Globulin_Ratio    Dataset
##  Min.   :2.70   Min.   :0.900   Min.   :0.3000             Min.   :1.000
##  1st Qu.:5.80   1st Qu.:2.600   1st Qu.:0.7000             1st Qu.:1.000
##  Median :6.60   Median :3.100   Median :0.9600             Median :1.000
##  Mean   :6.49   Mean   :3.147   Mean   :0.9499             Mean   :1.285
##  3rd Qu.:7.20   3rd Qu.:3.800   3rd Qu.:1.1000             3rd Qu.:2.000
##  Max.   :9.60   Max.   :5.500   Max.   :2.8000             Max.   :2.000
##                                 NA's   :4
```

```
print("Validation Dataset")
```

```
## [1] "Validation Dataset"
```

```
summary(validation)
```

```
##       Age           Gender   Total_Bilirubin  Direct_Bilirubin
##  Min.   :10.00   Female:18   Min.   : 0.500   Min.   : 0.100
##  1st Qu.:32.50   Male  :41   1st Qu.: 0.700   1st Qu.: 0.200
##  Median :46.00               Median : 1.000   Median : 0.300
##  Mean   :44.98               Mean   : 3.208   Mean   : 1.454
##  3rd Qu.:57.00               3rd Qu.: 3.000   3rd Qu.: 1.600
##  Max.   :84.00               Max.   :22.700   Max.   :10.200
##  Alkaline_Phosphotase Alamine_Aminotransferase Aspartate_Aminotransferase
##  Min.   :123.0        Min.   : 11.00           Min.   : 10.00
##  1st Qu.:177.0        1st Qu.: 21.50           1st Qu.: 28.00
##  Median :206.0        Median : 34.00           Median : 43.00
##  Mean   :253.3        Mean   : 57.25           Mean   : 77.12
##  3rd Qu.:265.5        3rd Qu.: 56.50           3rd Qu.: 89.00
##  Max.   :850.0        Max.   :322.00           Max.   :540.00
##  Total_Protiens     Albumin      Albumin_and_Globulin_Ratio
##  Min.   :4.000   Min.   :1.600   Min.   :0.4000
##  1st Qu.:5.650   1st Qu.:2.600   1st Qu.:0.8000
##  Median :6.400   Median :3.000   Median :0.9000
##  Mean   :6.414   Mean   :3.095   Mean   :0.9229
##  3rd Qu.:7.100   3rd Qu.:3.550   3rd Qu.:1.0400
##  Max.   :9.500   Max.   :4.900   Max.   :1.7000
##     Dataset
##  Min.   :1.000
##  1st Qu.:1.000
##  Median :1.000
##  Mean   :1.305
##  3rd Qu.:2.000
##  Max.   :2.000
```

3.1 Data Wrangling

3.1.1 Remove null values

The variable "Albumin_and_Globulin_Ratio" has four null values. We use the traditional data science approach and replace null values with the mean of the variable.

```
# Replace null values with the mean of the variable
training$Albumin_and_Globulin_Ratio[is.na(training$Albumin_and_Globulin_Ratio)] <- mean(training$Albumi
```

3.1.2 Create LiverDisease Variable

To improve readability, we create a new variable, namely, "LiverDisease", which will have one of the following values:

1. Malignant (M) indicating that the patient has liver disease.
2. Benign (B) indicating that the patient has no liver disease.

We further delete the "Dataset" variable as it is no longer needed. We apply these operations to both training and validation datasets.

```
# Adding a new column, which will contain the disease information
# M -> Malignant
# B -> Benign
training <- transform(training, LiverDisease= ifelse(Dataset==1, "M","B"))
validation <- transform(validation, LiverDisease= ifelse(Dataset==1, "M","B"))

# Deleting the column 'Dataset' as no longer required
training<-within(training, rm(Dataset))
validation<-within(validation, rm(Dataset))

# Displaying the first six rows
head(training)
```

Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Ar
62	Male	10.9	5.5	699	64	
62	Male	7.3	4.1	490	60	
58	Male	1.0	0.4	182	14	
72	Male	3.9	2.0	195	27	
46	Male	1.8	0.7	208	19	
26	Female	0.9	0.2	154	16	

The training data has 28% of the patient records, which has no liver damage or dieases. The rest of the patients have a liver damage.

```
summary(training$LiverDisease)/nrow(training)*100.0
```

```
##        B        M
## 28.48948 71.51052
```

4 Data Analysis

In this section, we extract insights from all variables to get an in-depth understanding before using them to train the machine learning models.

4.1 Age

The dataset consists of patients with varying ages ranging from 4 to 90. The distribution of ages shows a nice spread and indicates that the dataset is unbiased towards a specific age group.

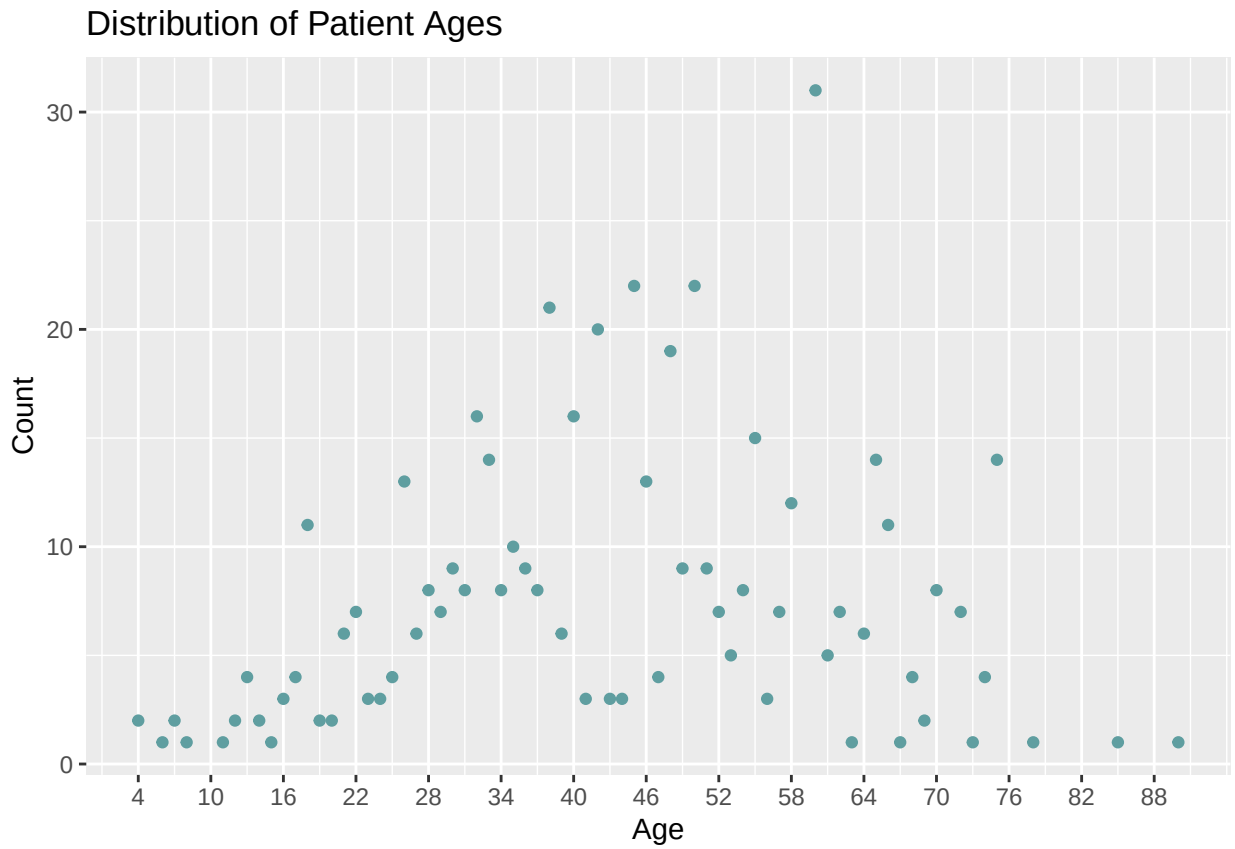
```
sprintf("Minimum age = %d",min(training$Age))
```

```
## [1] "Minimum age = 4"
```

```
sprintf("Maximum age = %d",max(training$Age))
```

```
## [1] "Maximum age = 90"
```

```
# Extracting frequency of patient ages
age_stats <-as.data.frame(table(training$Age))
names(age_stats)<- c("Age","Count")

# Removing the factor
age_stats$Age<-as.numeric(levels(age_stats$Age))

# Plotting distribution of ages
age_stats %>% ggplot(aes(Age, Count)) +
  geom_point(color="cadetblue") +
  scale_x_continuous(breaks = round(seq(min(age_stats$Age),
                                        max(age_stats$Age), by = 6),1)) +
  ggtitle("Distribution of Patient Ages")
```



Now, we breakdown the distribution of ages to the presence or absence of liver diseases. We notice a good spread of age group for both scenarios.

```
# Plotting distributions of ages based on liver diseases
training %>%
  ggplot(aes(as.numeric(row.names(training)),Age, color=LiverDisease)) +
  geom_point() +
  labs(y="Age", x = "Number of patients")+
  facet_wrap( ~ LiverDisease) +
  ggtitle("Distribution of ages based on liver disease")
```

4.2 Gender

76% of the patient records are of males. It would be good to have a more and less equal distribution of records for both genders, although we do not expect it to make any difference in the performance of our models.

Both "Gender" and "Age" variables are not used to train the model. These variables provide descriptive information.

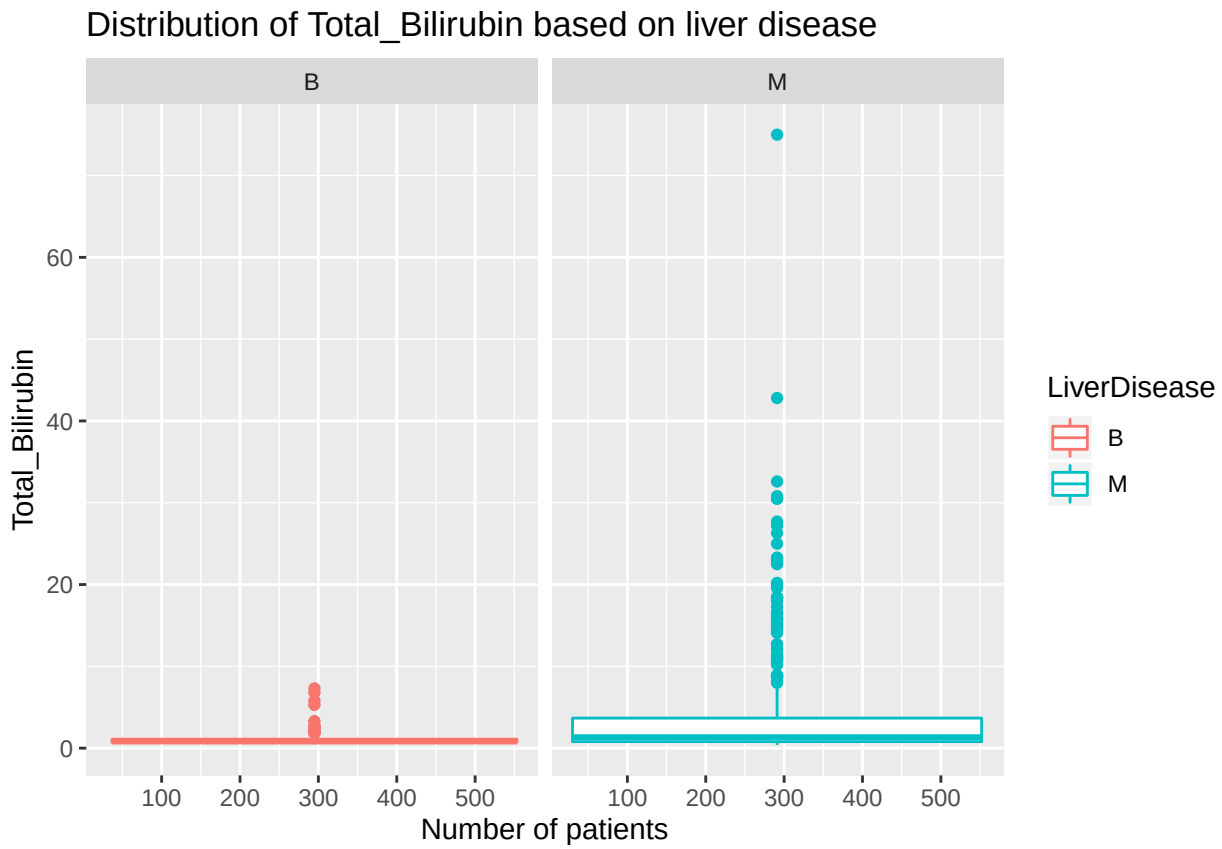
```
# Getting summary of genders
summary(training$Gender)
```

```
## Female   Male
##    123    400
```

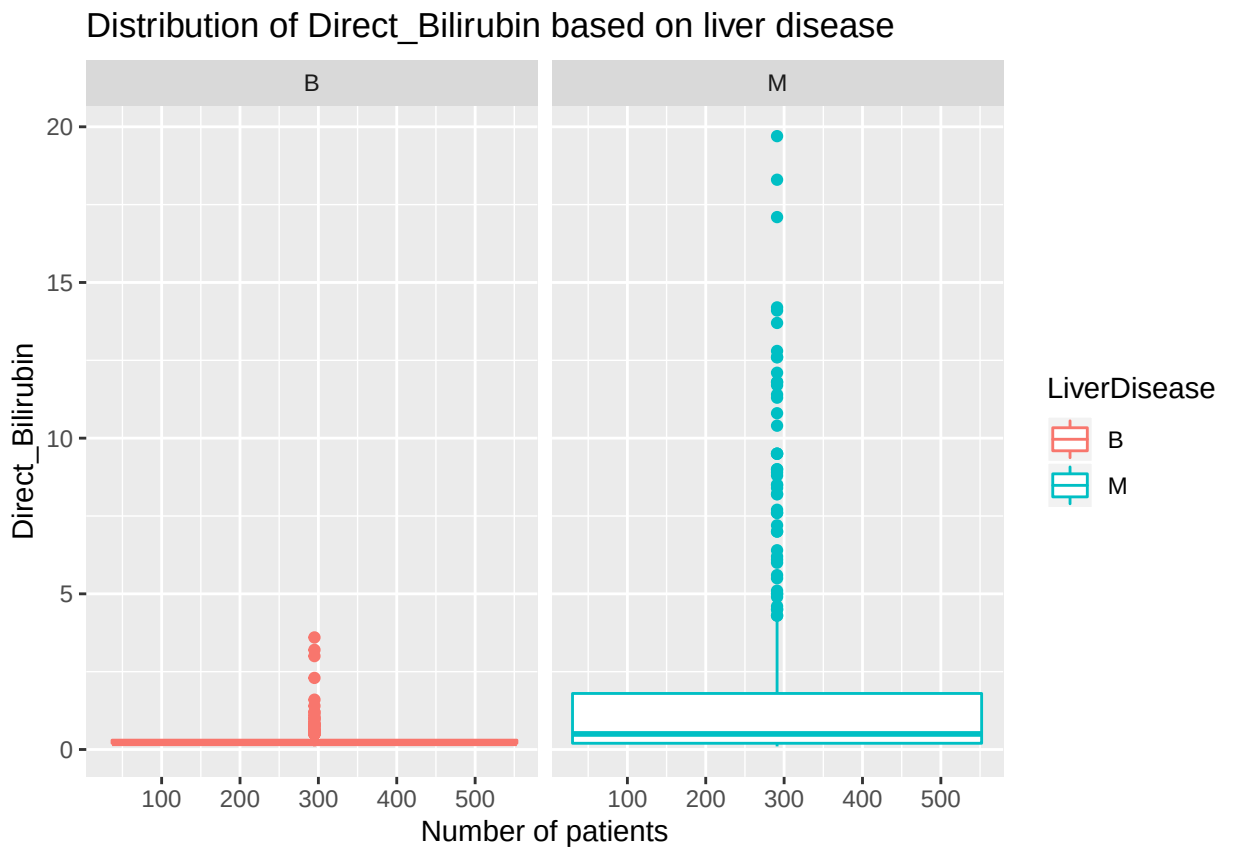
\subsection{Total_Bilirubin and Direct_Bilirubin} Bilirubin refers to any form of a yellowish pigment made in the liver when red blood cells are broken down. The elevated levels of bilirubin indicate that the liver is damaged. We find a similar trend with these variables that levels of bilirubin are high for patients with liver diseases.

```
# Plotting distributions of Total_Bilirubin based on liver diseases
training %>%
  ggplot(aes(as.numeric(row.names(training)),Total_Bilirubin, color=LiverDisease)) +
  geom_boxplot() +
  labs(y="Total_Bilirubin", x = "Number of patients")+
```

```
  facet_wrap( ~ LiverDisease) +
  ggtitle("Distribution of Total_Bilirubin based on liver disease")
```




```
# Plotting distributions of Direct_Bilirubin based on liver disease
training %>%
  ggplot(aes(as.numeric(row.names(training)),Direct_Bilirubin, color=LiverDisease)) +
  geom_boxplot() +
  labs(y="Direct_Bilirubin", x = "Number of patients")+
  facet_wrap( ~ LiverDisease) +
  ggtitle("Distribution of Direct_Bilirubin based on liver disease")
```



Now, we look at the correlations of bilirubins. We observe that both bilirubins are weakly correlated with liver disease. However, both bilirubins are highly correlated, and we can also use one of them to train the model (if required).

```
# Making a subset of data
subset_train <- training[c("Total_Bilirubin","Direct_Bilirubin","LiverDisease")]
# Converting disease variable to numeric format
subset_train <- transform(subset_train, LiverDisease= ifelse(subset_train$LiverDisease=="M", 1,0))
# Looking at the correlations
cor(subset_train)
```

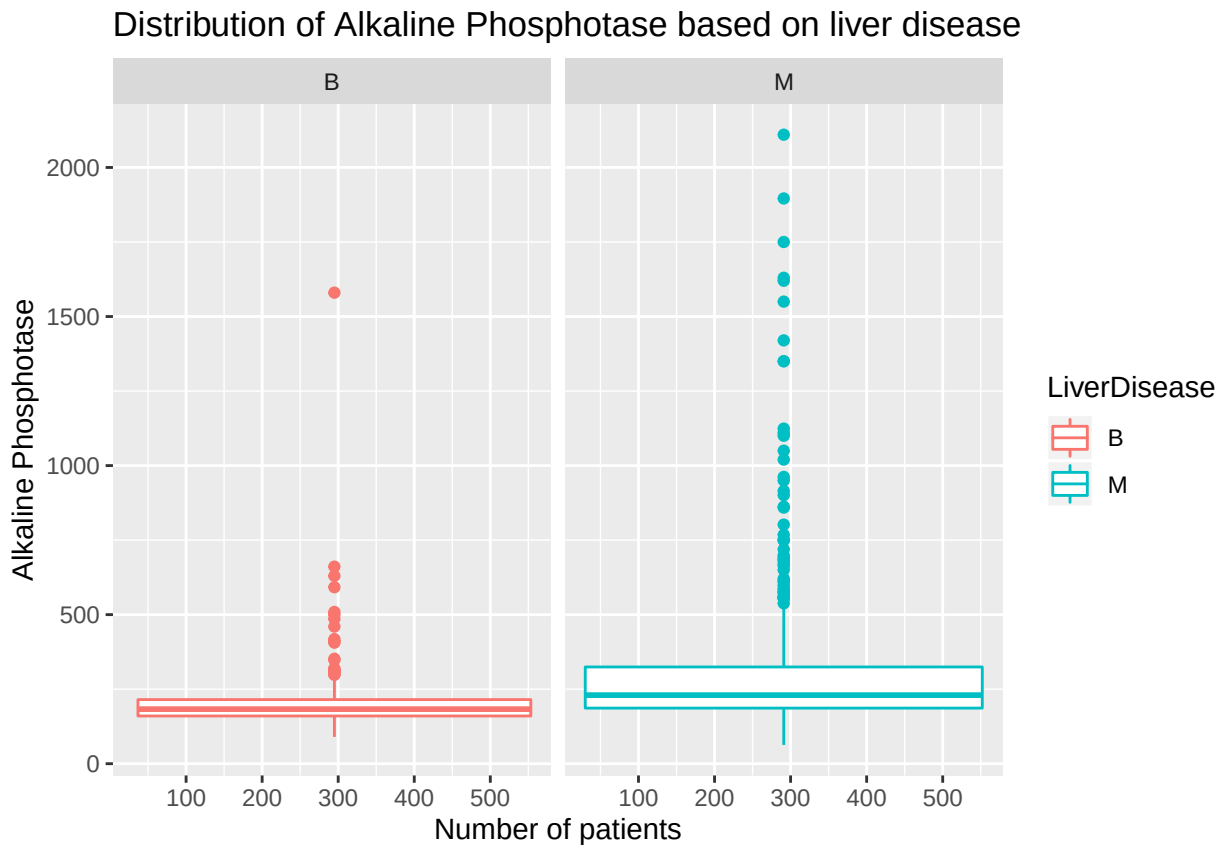
```
##                 Total_Bilirubin Direct_Bilirubin LiverDisease
## Total_Bilirubin       1.0000000        0.8655131    0.2165775
## Direct_Bilirubin      0.8655131        1.0000000    0.2429118
## LiverDisease          0.2165775        0.2429118    1.0000000
```

4.3 Alkaline Phosphotase

Alkaline phosphatase (ALP) is an enzyme in a person's blood that helps break down proteins. The elevated levels indicate that the liver has a disease, and we notice a similar trend in our dataset.

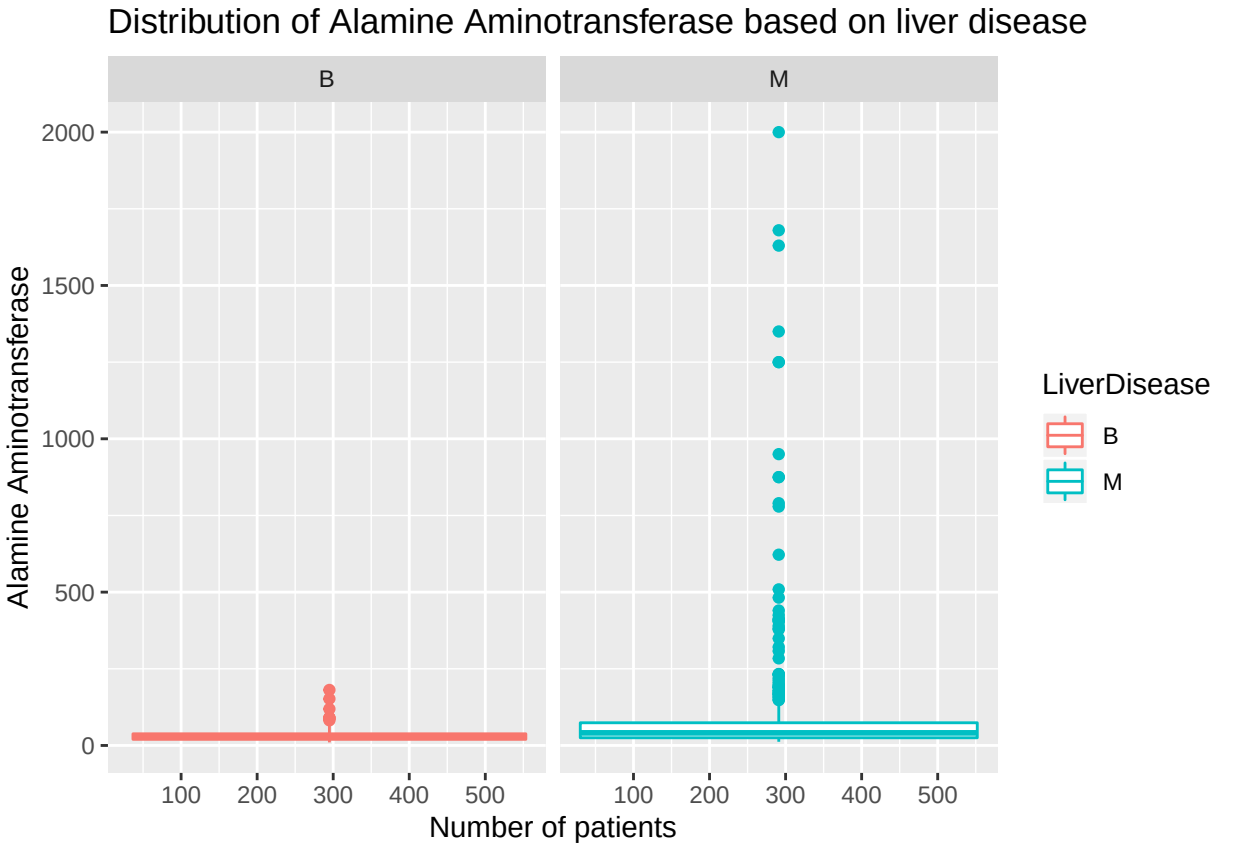
```
# Plotting distributions of Alkaline Phosphotase based on liver diseases
training %>%
  ggplot(aes(as.numeric(row.names(training)),Alkaline_Phosphotase, color=LiverDisease)) +
  geom_boxplot() +
```

```
  labs(y="Alkaline Phosphotase", x = "Number of patients")+
  facet_wrap( ~ LiverDisease) +
  ggtitle("Distribution of Alkaline Phosphotase based on liver disease")
```

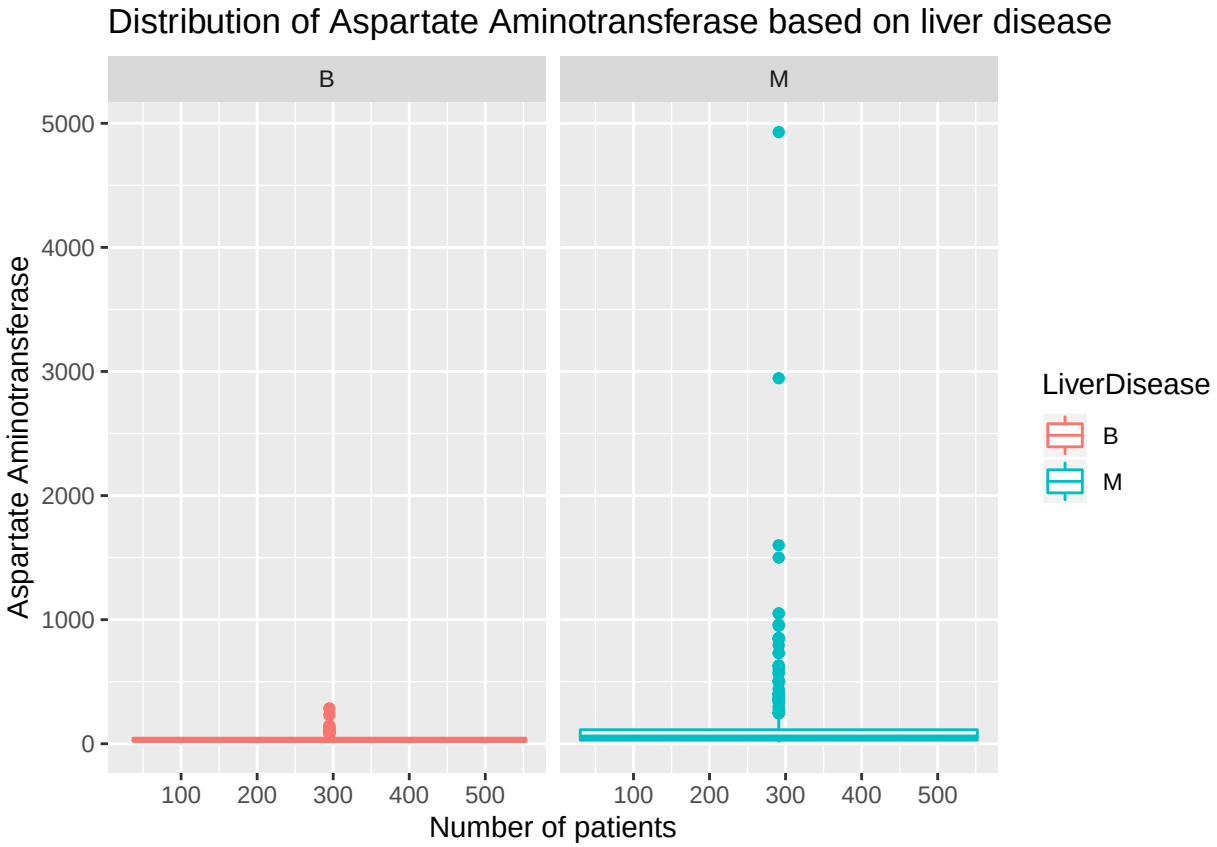


\subsection{Alamine_Aminotransferase and Aspartate_Aminotransferase} Aminotransferases are enzymes that are important in the synthesis of amino acids, which form proteins. Alanine aminotransferase (ALT) and Aspartate aminotransferase (AST) are found primarily in the liver and kidney. High levels of ALT and AST are expected for patients with liver diseases. We also notice slightly elevated levels of these enzymes for patients with liver diseases in our dataset.

```
# Plotting distributions of Alamine Aminotransferase based on liver diseases
training %>%
  ggplot(aes(as.numeric(row.names(training)),Alamine_Aminotransferase, color=LiverDisease)) +
  geom_boxplot() +
  labs(y="Alamine Aminotransferase", x = "Number of patients")+
  facet_wrap( ~ LiverDisease) +
  ggtitle("Distribution of Alamine Aminotransferase based on liver disease")
```




```
# Plotting distributions of Aspartate_Aminotransferase based on liver diseases
training %>%
  ggplot(aes(as.numeric(row.names(training)),Aspartate_Aminotransferase, color=LiverDisease)) +
  geom_boxplot() +
  labs(y="Aspartate Aminotransferase", x = "Number of patients")+
  facet_wrap( ~ LiverDisease) +
  ggtitle("Distribution of Aspartate Aminotransferase based on liver disease")
```



Contrary to bilirubins, there exists a weak correlation between both aminotransferases.

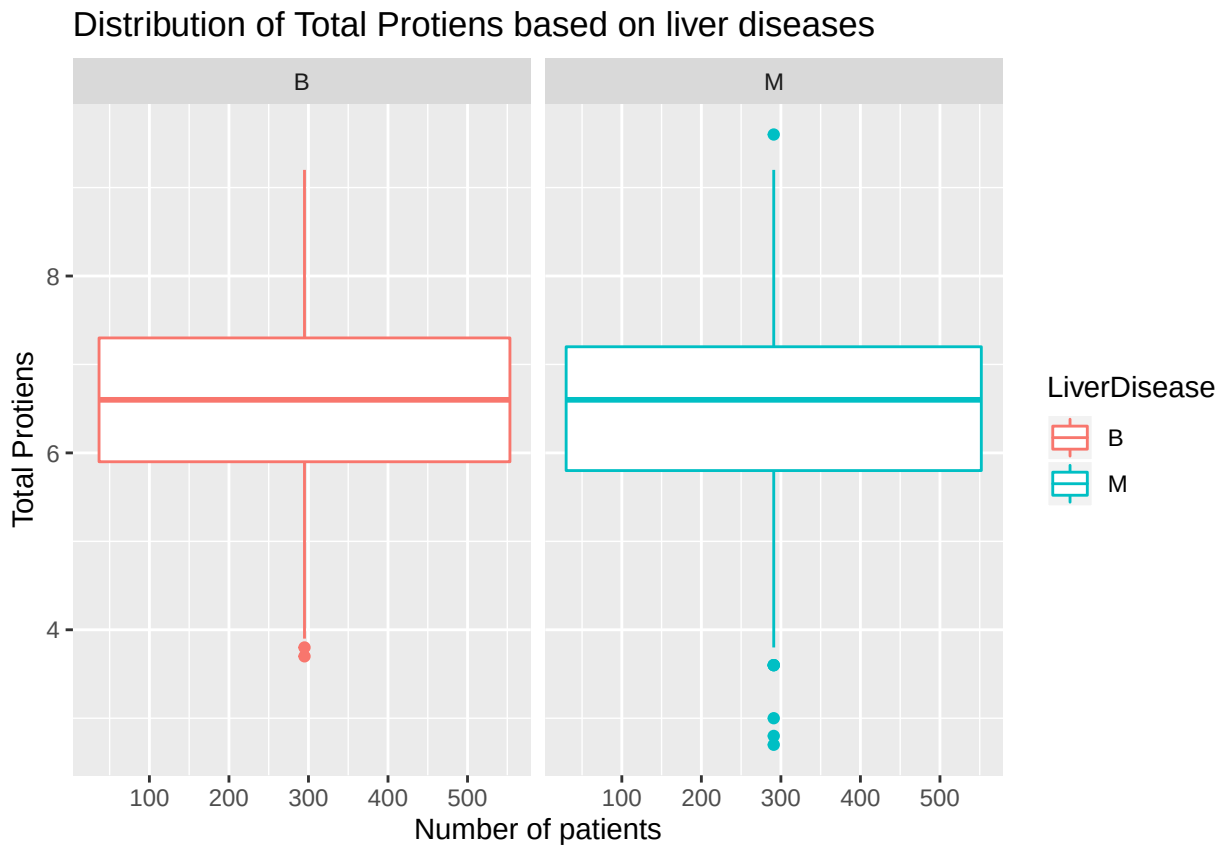
```
# Making a subset of data
subset_train <- training[c("Alkaline_Phosphotase","Aspartate_Aminotransferase","LiverDisease")]

# Converting disease variable to numeric format
subset_train <- transform(subset_train, LiverDisease= ifelse(subset_train$LiverDisease=="M", 1,0))

# Looking at the coorelations
cor(subset_train)
```

```
##                          Alkaline_Phosphotase Aspartate_Aminotransferase
## Alkaline_Phosphotase                1.0000000                  0.1644067
## Aspartate_Aminotransferase          0.1644067                  1.0000000
## LiverDisease                        0.1848371                  0.1524066
##                          LiverDisease
## Alkaline_Phosphotase        0.1848371
## Aspartate_Aminotransferase  0.1524066
## LiverDisease                1.0000000
```

4.4 Total Protiens

The total protein test measures the total amount of protein in your body. The distributions indicate that this variable cannot be used to diagnose liver disease. We do not see any pattern which we can use for classification.

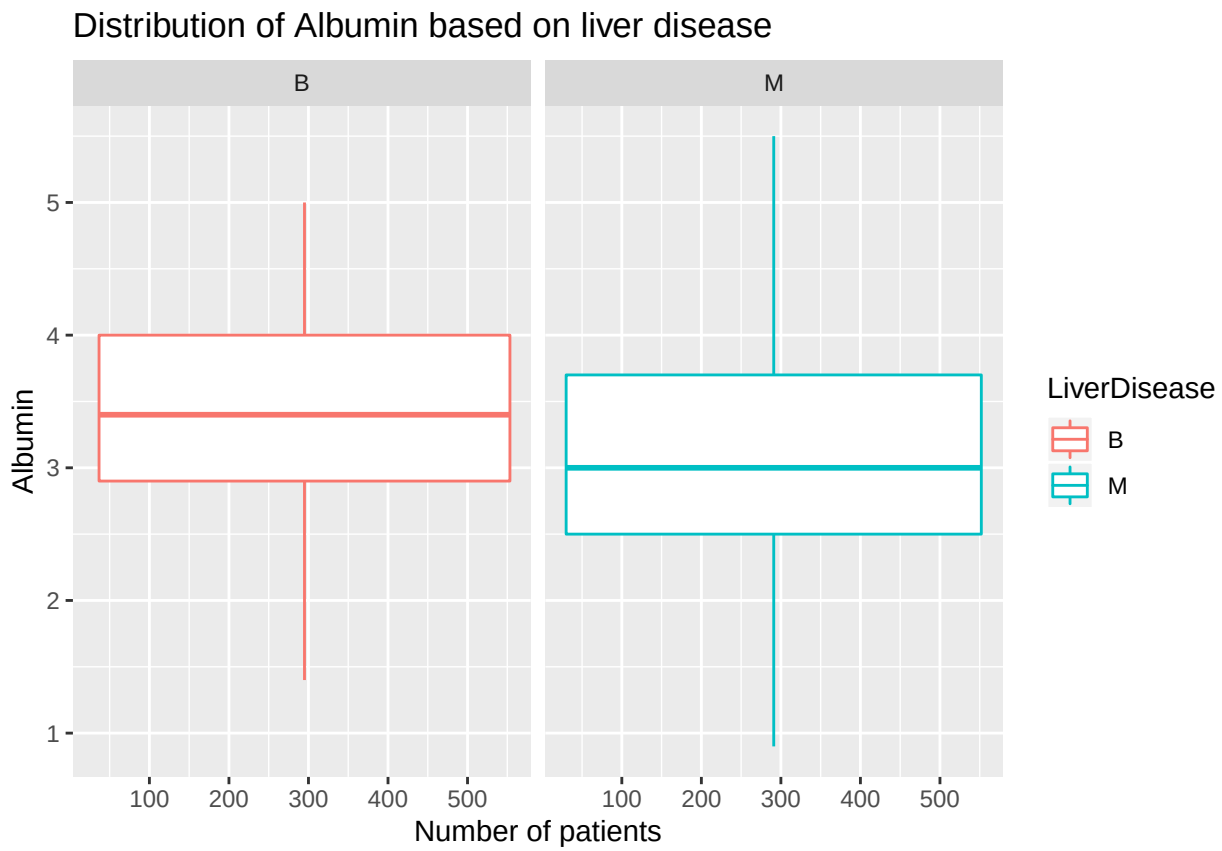
```r
# Plotting distributions of Total Protiens based on liver diseases
training %>%
  ggplot(aes(as.numeric(row.names(training)),Total_Protiens, color=LiverDisease)) +
  geom_boxplot() +
  labs(y="Total Protiens", x = "Number of patients")+
  facet_wrap( ~ LiverDisease) +
  ggtitle("Distribution of Total Protiens based on liver diseases")
```



4.5 Albumin

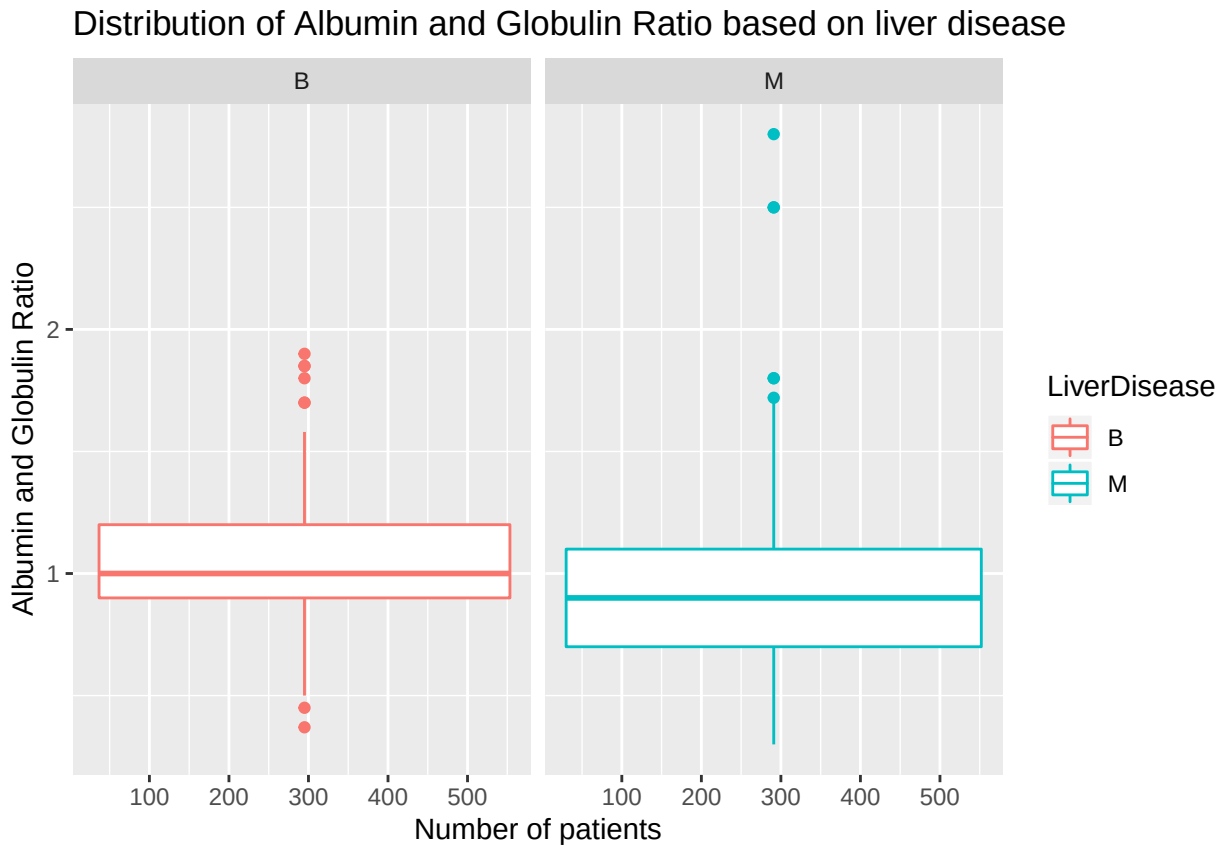
Albumin is a protein made by the liver to keep fluid in the bloodstream. The low levels of albumin indicate a problem with the liver, and we notice a similar trend in our dataset.

```r
# Plotting distributions of Albumin based on liver diseases
training %>%
  ggplot(aes(as.numeric(row.names(training)),Albumin, color=LiverDisease)) +
  geom_boxplot() +
  labs(y="Albumin", x = "Number of patients")+
  facet_wrap( ~ LiverDisease) +
  ggtitle("Distribution of Albumin based on liver disease")
```



\subsection{Albumin_and_Globulin_Ratio (AG)} These proteins are crucial for body growth, development, and health. They form the structural part of most organs and makeup enzymes that regulate body functions. The low ratios of AG refer to liver issues, and we notice the same trend from distributions plot.

```
# Plotting distributions of Albumin based on liver diseases
training %>%
  ggplot(aes(as.numeric(row.names(training)),Albumin_and_Globulin_Ratio, color=LiverDisease)) +
  geom_boxplot() +
  labs(y="Albumin and Globulin Ratio", x = "Number of patients")+
  facet_wrap( ~ LiverDisease) +
  ggtitle("Distribution of Albumin and Globulin Ratio based on liver disease")
```



5 Methods

Based on the discussion in Section 4, we will not use "Age" and "Total Protein" variables to train the machine learning models. So, we remove these variables from both training and validation datasets. Then, we predict the liver disease using all the remaining variables of the dataset.

```
# Removing the variables 'Age' and 'Dataset'
training<-within(training, rm(Age,Total_Protiens))
validation<-within(validation, rm(Age,Total_Protiens))
```

For data pre-processing, we remove zero-variance predictors and then center and scale all those remaining using the preProc argument. Feature scaling is one of the most critical steps during the pre-processing of data before creating a machine learning model. It can make a significant difference between a weak machine learning model and a better one.

5.1 Logistic Regression

We use *glm* method with cross-validation of 10 folds to train the model.

```
# Defining a cross-validation (10 K folds )
control <- trainControl(method = "repeatedcv", number =10,repeats = 5)

# Train logistic regression model
```

```r
train_glm <- train(LiverDisease ~.,
                   method = "glm",
                   preProc = c("zv","center", "scale"),
                   data = training,
                   trControl=control)

# Showing the accuracy
sprintf("The accuracy of GLM = %f",train_glm$results$Accuracy)
```

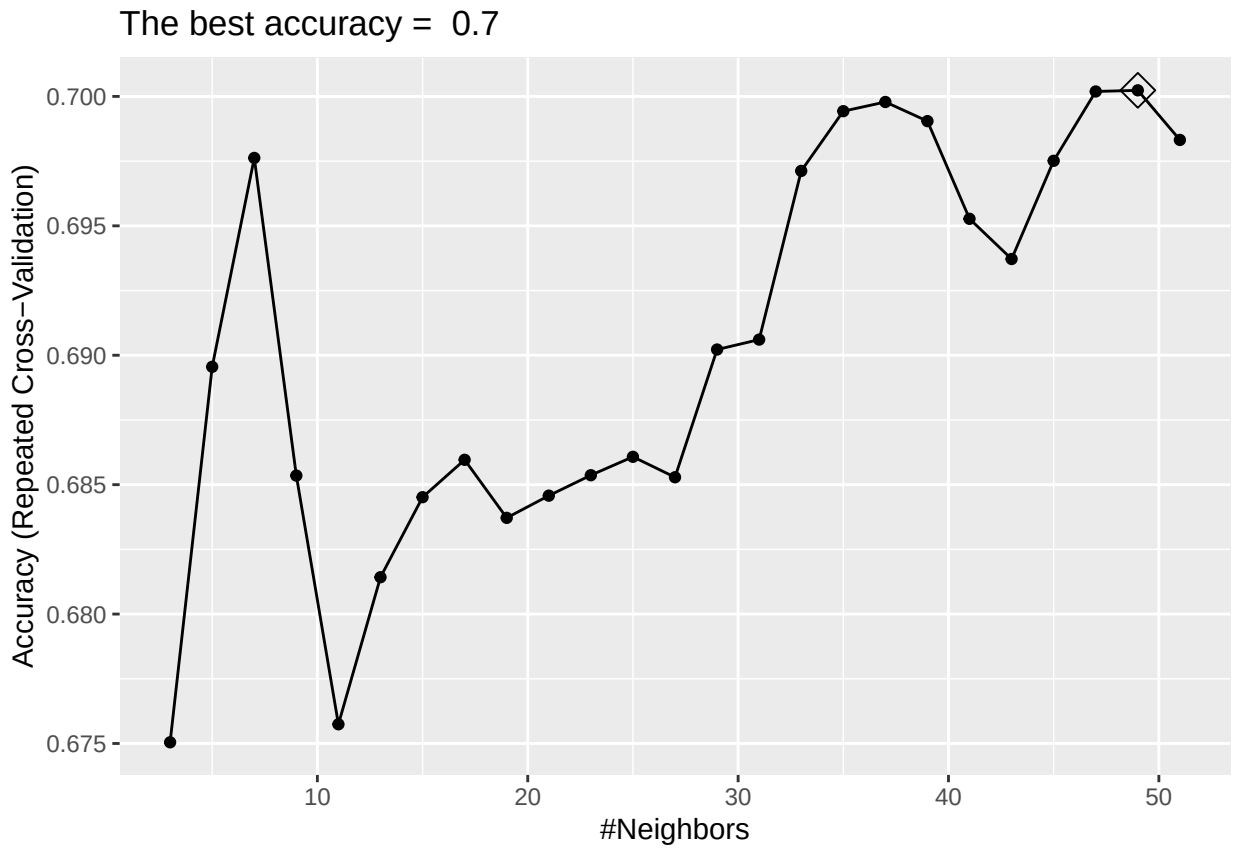
```
## [1] "The accuracy of GLM = 0.702495"
```

```r
# Storing the results
model_results <- data_frame(method = "glm", Accuracy = train_glm$results$Accuracy)
```

5.2 K-nearest neigbors (knn)

We use *knn* method with cross-validation of 10 folds to train the model. We tune the model with several values of k, ranging from 3 to 51, to optimize the performance.

```r
set.seed(1)
# Defining a cross validation (10 K folds )
control <- trainControl(method = "repeatedcv", number = 10, repeats=5)

# Train knn model
train_knn <- train(LiverDisease~ .,
                   method = "knn",
                   preProc = c("zv","center", "scale"),
                   data = training,
                   tuneGrid = data.frame(k = seq(3, 51, 2)),
                   trControl = control)

# Plot the model and highlight the best result
ggplot(train_knn, highlight = TRUE) +
  ggtitle(paste("The best accuracy = ",round(max(train_knn$results$Accuracy),3)))
```



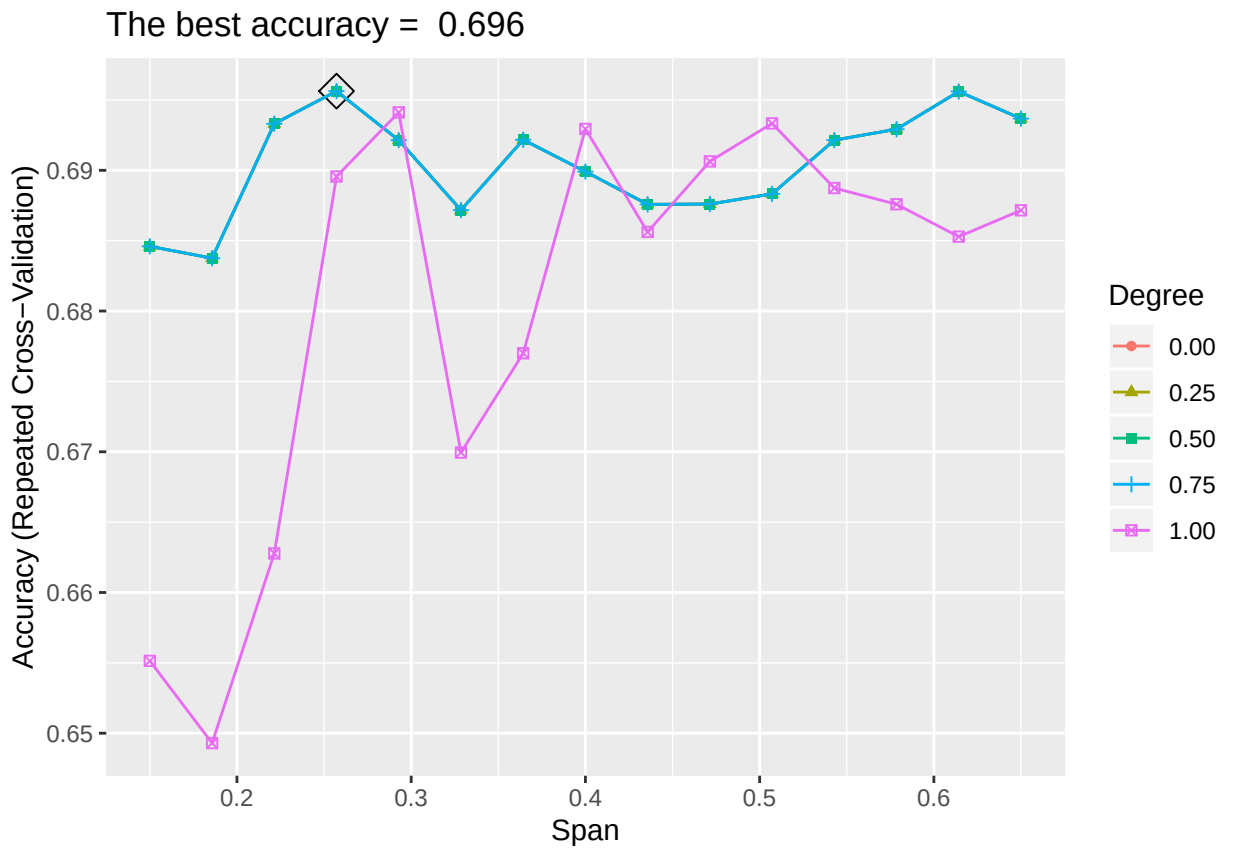

```
# Storing the results
model_results <- bind_rows(model_results,data_frame(method="knn",
                                    Accuracy = max(train_knn$results$Accuracy) ))
```

5.3 Local Regression

We use *loess* method with cross-validation of 10 folds to train the model. We tune the **span** and **degree** parameters to optimize the performance of the model.

```
set.seed(1)

# Defining a cross validation (10 K folds )
control <- trainControl(method = "repeatedcv", number = 10, repeats=5)

# Define tuning parameters
tune_grid <- expand.grid(span = seq(0.15, 0.65, len = 15), degree = seq(0,1,0.25))

# Train the model
train_loess <- train(LiverDisease~.,
                     method = "gamLoess",
                     preProc = c("zv","center", "scale"),
                     data = training,
                     tuneGrid= tune_grid,
                     trControl = control)
```

```
# Plot the model and highlight the best result
ggplot(train_loess, highlight = TRUE) +
  ggtitle(paste("The best accuracy = ",round(max(train_loess$results$Accuracy),3)))
```



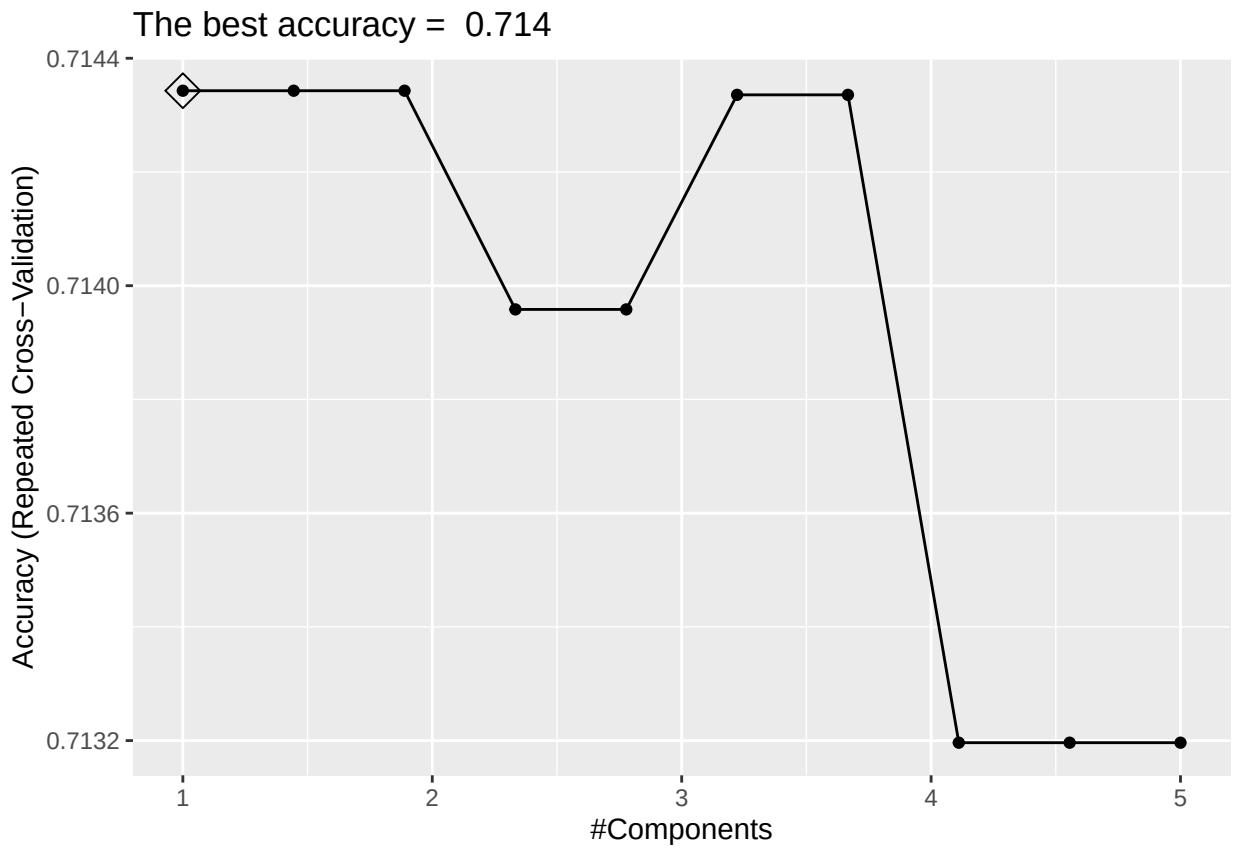

```
# Storing the results
model_results <- bind_rows(model_results,data_frame(method="loess",
                                    Accuracy = max(train_loess$results$Accuracy) ))
```

5.4 Partial Least Squares (PLS)

We use *pls* method with cross-validation of 10 folds to train the model. We tune the **ncomp** parameter to optimize the performance of the model.

```
set.seed(1)

# Define tuning parameters
tune_grid <- expand.grid(ncomp = seq(1,5, len = 10))

# Defining a cross validation (10 K folds )
control <- trainControl(method = "repeatedcv", number = 10, repeats=5)

# Train the model
train_pls <- train(LiverDisease~.,
                   method = "pls",
```

```
                  preProc = c("zv","center", "scale"),
                  data = training,
                  tuneGrid= tune_grid,
                  trControl = control)

# Plot the model and highlight the best result
ggplot(train_pls, highlight = TRUE) +
  ggtitle(paste("The best accuracy = ",round(max(train_pls$results$Accuracy),3)))
```




```
# Storing the results
model_results <- bind_rows(model_results,data_frame(method="pls",
                                    Accuracy = max(train_pls$results$Accuracy) ))
```

5.5 Linear Discriminant Analysis (LDA)

The *lda* is a statistical classifier, and we use this method with cross-validation of 10 folds for training. There is no parameter to tune for this model.

```
set.seed(1)
# Defining a cross validation (10 K folds )
control <- trainControl(method = "repeatedcv", number=10, repeats=5)

# Train the model
train_lda <- train(LiverDisease~.,
```

```
                    method = "lda",
                    preProc = c("zv","center", "scale"),
                    data = training,
                    trControl = control)

sprintf("The accuracy of lda = %f",max(train_lda$results$Accuracy))
```

```
## [1] "The accuracy of lda = 0.710134"
```

```
# Storing the results
model_results <- bind_rows(model_results,data_frame(method="lda",
                                      Accuracy = max(train_lda$results$Accuracy) ))
```

5.6 Quadratic Discriminant Analysis (QDA)

The *qda* is a statistical classifier, and we use the method with cross-validation of 10 folds for training. There is no parameter to tune for this model.

```
set.seed(1)
# Defining a cross validation (10 K folds )
control <- trainControl(method = "repeatedcv", number=10, repeats=5)

# Train the model
train_qda <- train(LiverDisease~.,
                    method = "qda",
                    preProc = c("zv","center", "scale"),
                    data = training,
                    trControl = control)

sprintf("The accuracy of qda = %f",max(train_qda$results$Accuracy))
```

```
## [1] "The accuracy of qda = 0.557171"
```

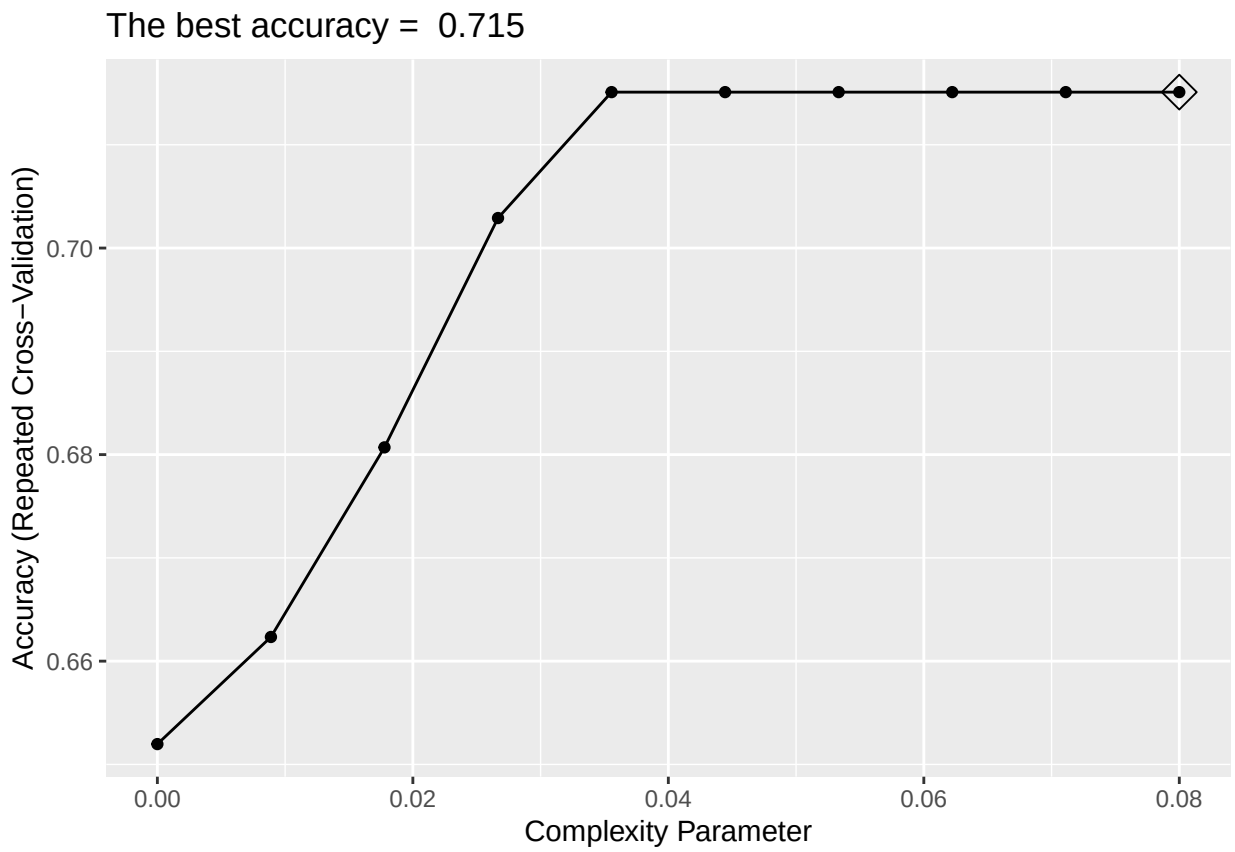
```
# Storing the results
model_results <- bind_rows(model_results,data_frame(method="qda",
                                      Accuracy = max(train_qda$results$Accuracy) ))
```

5.7 Decision Tress

We use *raprt* method with cross-validation of 10 folds for training. We tune the *cp* parameter to optimize the performance of the model.

```
set.seed(1)

# Defining a cross validation (10 K folds )
control <- trainControl(method = "repeatedcv", number = 10, repeats=5)

# Train the model
train_rpart <- train(LiverDisease~.,
                      method = "rpart",
```

```
                      preProc = c("zv","center", "scale"),
                      data = training,
                      trControl = control,
                      tuneGrid = data.frame(cp = seq(0, 0.08, len = 10)))

# Plot the model and highlight the best result
ggplot(train_rpart, highlight = TRUE) +
  ggtitle(paste("The best accuracy = ",round(max(train_rpart$results$Accuracy),3)))
```



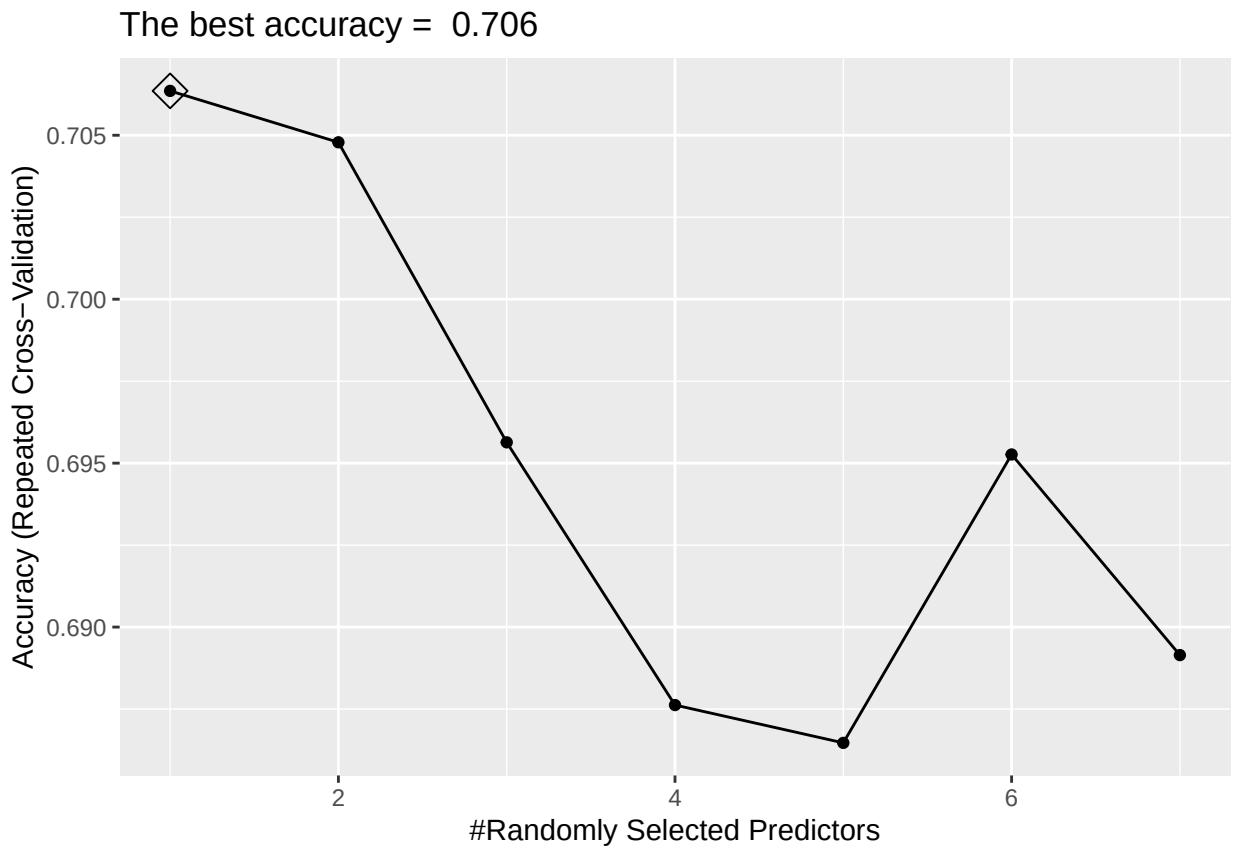

```
# Storing the results
model_results <- bind_rows(model_results,data_frame(method="rpart",
                                      Accuracy = max(train_rpart$results$Accuracy) ))
```

5.8 Random Forests

We use *rf* method with cross-validation of 10 folds for training. We tune the *mtry* parameter to optimize the performance of the model.

```
set.seed(1)

# Defining a cross validation (10 K folds )
control <- trainControl(method = "repeatedcv", number = 10, repeats=5)

# Train the model
```

```
train_rf <- train(LiverDisease~.,
                  method = "rf",
                  preProc = c("zv","center", "scale"),
                  data = training,
                  trControl = control,
                  tuneGrid = data.frame(mtry=seq(1,7)),
                  ntree=100)

# Plot the model and highlight the best result
ggplot(train_rf, highlight = TRUE) +
  ggtitle(paste("The best accuracy = ",round(max(train_rf$results$Accuracy),3)))
```




```
# Storing the results
model_results <- bind_rows(model_results,data_frame(method="rf",
                                    Accuracy = max(train_rf$results$Accuracy) ))
```

5.9 Support Vector Machine

We use support vector machine with cross-validation of 10 folds for training. We tune the *tau* parameter to optimize the performance of the model.

```
set.seed(1)

# Defining a cross validation (10 K folds )
```

```
control <- trainControl(method = "repeatedcv", number = 10, repeats = 5)

# Train the model
train_svm <- train(LiverDisease~.,
                   preProc = c("zv","center", "scale"),
                   data = training,
                   method = "svmLinear",
                   tune_grid= data.frame(tau=seq(1,10)),
                   trControl = control)

sprintf("The accuracy of svm = %f",max(train_svm$results$Accuracy))
```

```
## [1] "The accuracy of svm = 0.715098"
```

```
# Storing the results
model_results <- bind_rows(model_results,data_frame(method="svm",
                                     Accuracy = max(train_svm$results$Accuracy) ))
```

5.10 Adaptive Boosting (Adaboost)

AdaBoost is a machine learning meta-algorithm for classification. We train the model with cross-validation of 10 folds.

```
set.seed(1)

# Defining a cross validation (10 K folds )
control <- trainControl(method = "repeatedcv", number = 10, repeats = 5)

# Tuning the model
tune_grid  = expand.grid(method = c("Adaboost.M1", "Real adaboost"))

train_ada <- train(LiverDisease~.,
                   preProc = c("zv","center", "scale"),
                   data = training,
                   method = "ada",
                   trControl = control)

sprintf("The accuracy of adaboost = %f",max(train_ada$results$Accuracy))
```

```
## [1] "The accuracy of adaboost = 0.713581"
```

```
# Stroing the results
model_results <- bind_rows(model_results,data_frame(method="ada",
                                     Accuracy = max(train_ada$results$Accuracy) ))
```

5.11 Random Forest with PCA

We apply principal component analysis on data and then use *rf* method to train the model with cross-validation of 10 folds. We utilize the same tuning, which was used before with the random forest model.

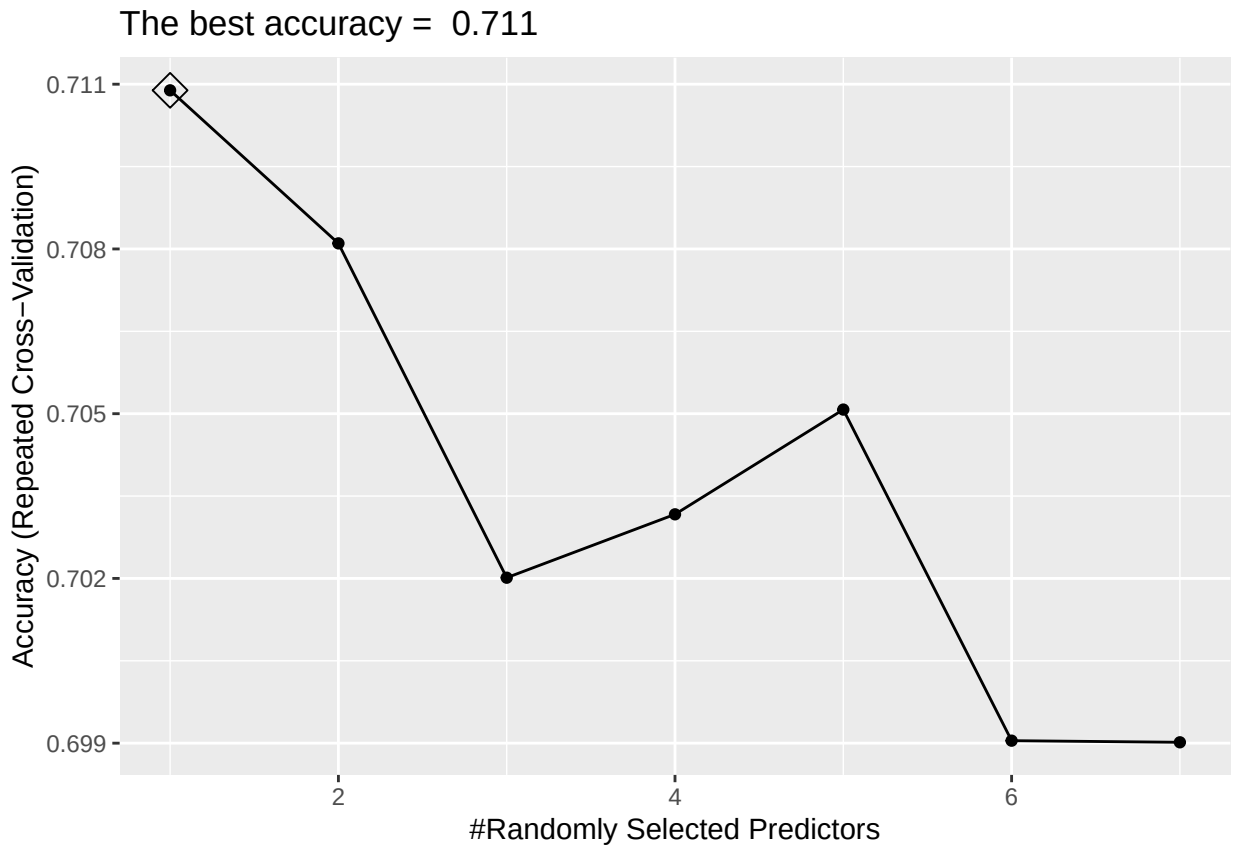
```r
set.seed(1)

# Defining a cross validation (10 K folds )
control <- trainControl(method = "repeatedcv", number = 10, repeats=5)
# Train the model
train_rf_pca <- train(LiverDisease~.,
                      method = "rf",
                      preProc = c("zv","center", "scale"),
                      data = training,
                      trControl = control,
                      tuneGrid = data.frame(mtry=seq(1,7)),
                      preProcess=c("pca"),
                      ntree=100)
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
# Plot the model and highlight the best result
ggplot(train_rf_pca, highlight = TRUE) +
  ggtitle(paste("The best accuracy = ",round(max(train_rf_pca$results$Accuracy),3)))
```




```
# Storing the results
model_results <- bind_rows(model_results,data_frame(method="rf_pca",
                                    Accuracy = max(train_rf_pca$results$Accuracy) ))
```

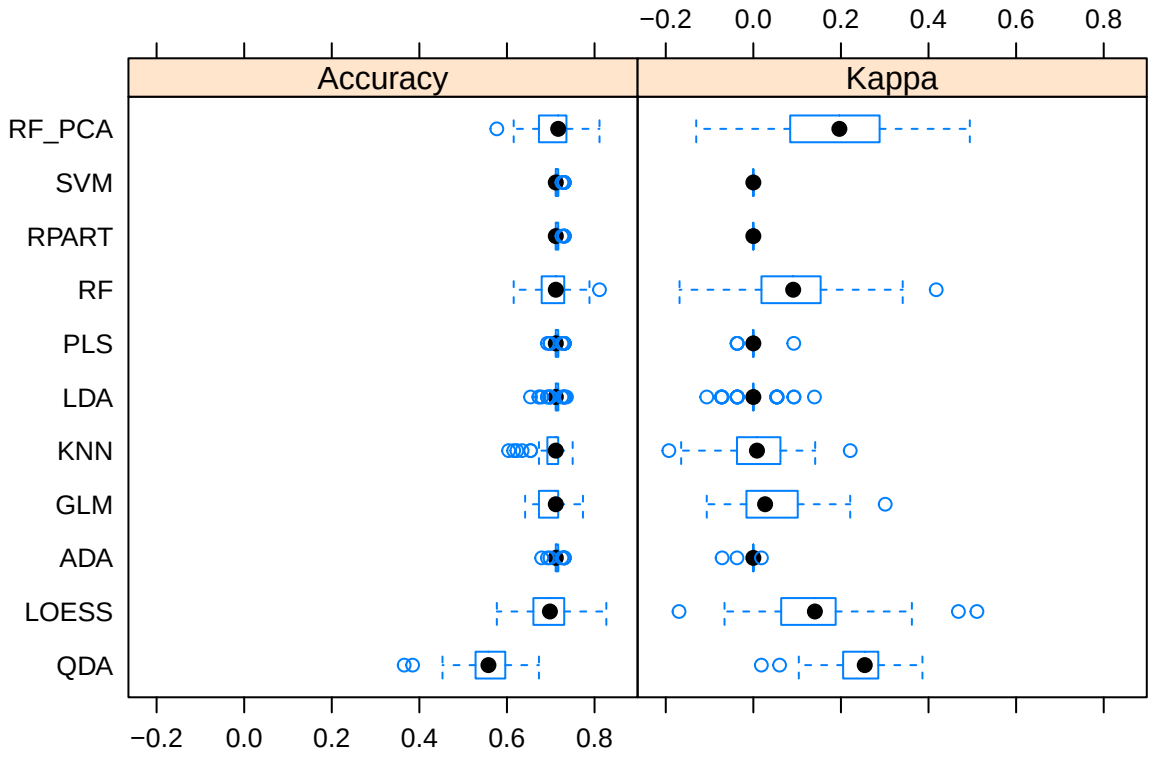
The reported accuracies and kappas for all models across the training dataset are shown in the following table and graph. The results show that *qda* model performs the worse. All other models provide an accuracy of around 0.70. The random forest, along with the principal component analysis, gives the best performance.

```
model_results
```

method	Accuracy
glm	0.7024949
knn	0.7002355
loess	0.6956271
pls	0.7143429
lda	0.7101339
qda	0.5571708
rpart	0.7150976
rf	0.7063528
svm	0.7150976

method	Accuracy
ada	0.7135809
rf_pca	0.7108886

```
# collect resamples
results <- resamples(list(GLM=train_glm,
                          KNN = train_knn,
                          LOESS=train_loess,
                          PLS= train_pls,
                          LDA = train_lda,
                          QDA = train_qda,
                          RPART = train_rpart,
                          RF = train_rf,
                          SVM = train_svm,
                          ADA= train_ada,
                          RF_PCA = train_rf_pca))
# boxplots of results
bwplot(results)
```



6 Results

The statistical measurements of accuracy and precision reveal the necessary reliability of a test. Specificity is the ability of a test to exclude individuals who do not have a given disease correctly, and sensitivity is

the ability of a test to identify people who have a given disease accurately. On the other hand, the F1 score is the harmonic mean of Precision and Recall and gives a better measure of the incorrectly classified cases than the accuracy metric. And, the kappa metric measures the inter-rater reliability.

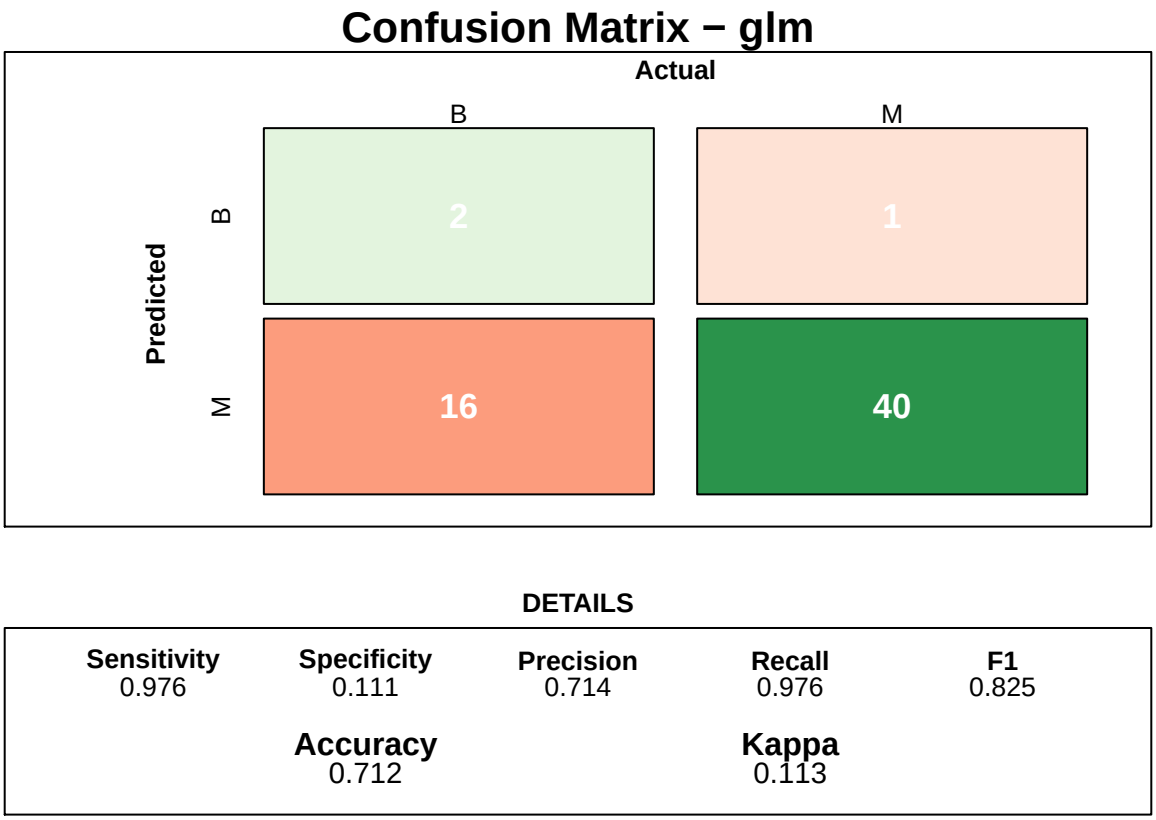
In Section 5, we trained several models using training data. Now, we will evaluate the performance of these models using validation data. We have not used **qda** model due to poor performance on training data. The results show that **rf** and \emph{rf_pca} models performs best in terms of precision and recall. However, the models have slightly lower sensitivity and specificity compared to other models. But, the \textbf{rf_pca} model gives the best kappa value and is significantly better than the remaining ones. So, we can deduce that overall, \textbf{rf_pca} performs the best compared to other models.

```
# Function to display a confusion matrix
# Code Source:
# https://stackoverflow.com/questions/23891140/r-how-to-visualize-confusion-matrix-using-the-caret-pack

draw_confusion_matrix <- function(cm, title) {

  total <- sum(cm$table)
  res <- as.numeric(cm$table)

  # Generate color gradients. Palettes come from RColorBrewer.
  greenPalette <- c("#F7FCF5","#E5F5E0","#C7E9C0","#A1D99B","#74C476","#41AB5D","#238B45","#006D2C","#0
  redPalette <- c("#FFF5F0","#FEE0D2","#FCBBA1","#FC9272","#FB6A4A","#EF3B2C","#CB181D","#A50F15","#670
  getColor <- function (greenOrRed = "green", amount = 0) {
    if (amount == 0)
      return("#FFFFFF")
    palette <- greenPalette
    if (greenOrRed == "red")
      palette <- redPalette
    colorRampPalette(palette)(100)[10 + ceiling(90 * amount / total)]
  }

  # set the basic layout
  layout(matrix(c(1,1,2)))
  par(mar=c(2,2,2,2))
  plot(c(100, 345), c(300, 450), type = "n", xlab="", ylab="", xaxt='n', yaxt='n')
  title(title, cex.main=2)

  # create the matrix
  classes = colnames(cm$table)
  rect(150, 430, 240, 370, col=getColor("green", res[1]))
  text(195, 435, classes[1], cex=1.2)
  rect(250, 430, 340, 370, col=getColor("red", res[3]))
  text(295, 435, classes[2], cex=1.2)
  text(125, 370, 'Predicted', cex=1.3, srt=90, font=2)
  text(245, 450, 'Actual', cex=1.3, font=2)
  rect(150, 305, 240, 365, col=getColor("red", res[2]))
  rect(250, 305, 340, 365, col=getColor("green", res[4]))
  text(140, 400, classes[1], cex=1.2, srt=90)
  text(140, 335, classes[2], cex=1.2, srt=90)

  # add in the cm results
  text(195, 400, res[1], cex=1.6, font=2, col='white')
  text(195, 335, res[2], cex=1.6, font=2, col='white')
  text(295, 400, res[3], cex=1.6, font=2, col='white')
```

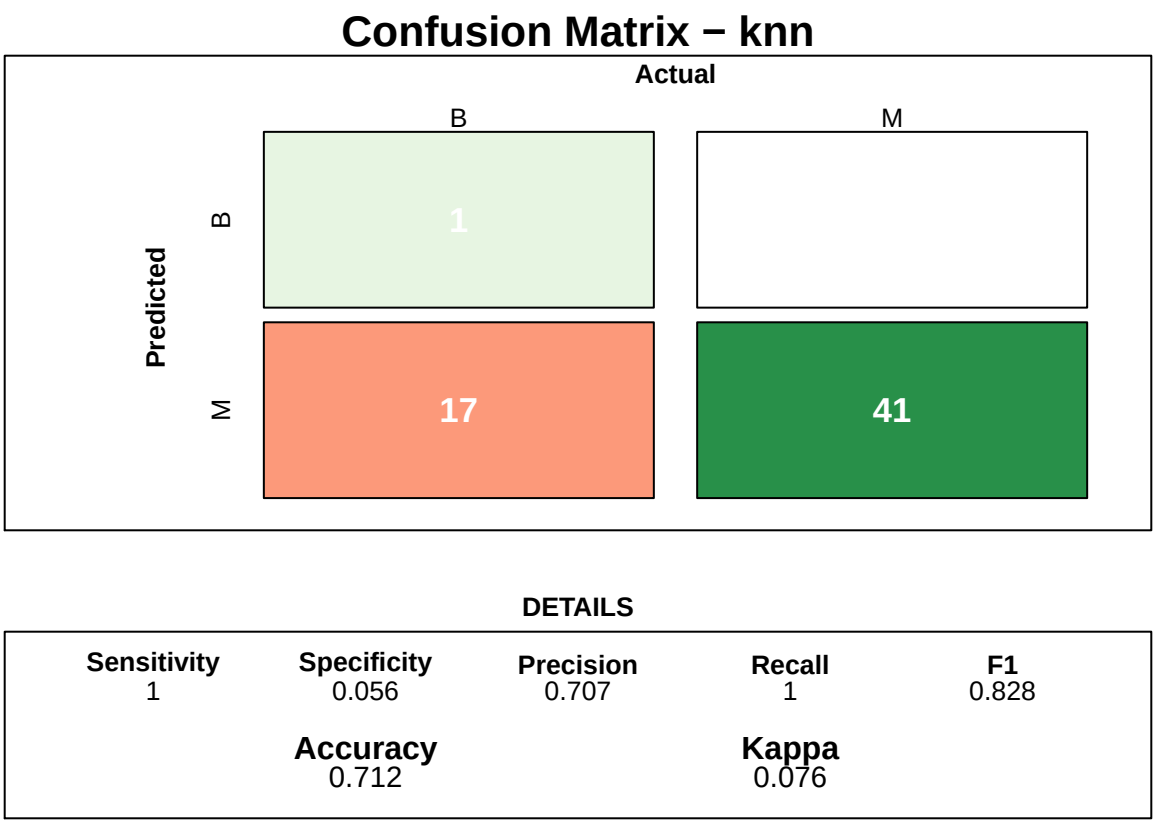
```
  text(295, 335, res[4], cex=1.6, font=2, col='white')

  # add in the specifics
  plot(c(100, 0), c(100, 0), type = "n", xlab="", ylab="", main = "DETAILS", xaxt='n', yaxt='n')
  text(10, 85, names(cm$byClass[1]), cex=1.2, font=2)
  text(10, 70, round(as.numeric(cm$byClass[1]), 3), cex=1.2)
  text(30, 85, names(cm$byClass[2]), cex=1.2, font=2)
  text(30, 70, round(as.numeric(cm$byClass[2]), 3), cex=1.2)
  text(50, 85, names(cm$byClass[5]), cex=1.2, font=2)
  text(50, 70, round(as.numeric(cm$byClass[5]), 3), cex=1.2)
  text(70, 85, names(cm$byClass[6]), cex=1.2, font=2)
  text(70, 70, round(as.numeric(cm$byClass[6]), 3), cex=1.2)
  text(90, 85, names(cm$byClass[7]), cex=1.2, font=2)
  text(90, 70, round(as.numeric(cm$byClass[7]), 3), cex=1.2)

  # add in the accuracy information
  text(30, 35, names(cm$overall[1]), cex=1.5, font=2)
  text(30, 20, round(as.numeric(cm$overall[1]), 3), cex=1.4)
  text(70, 35, names(cm$overall[2]), cex=1.5, font=2)
  text(70, 20, round(as.numeric(cm$overall[2]), 3), cex=1.4)
}
```

```
# Creating an empty data frame to hold the results of models
# across validation dataset
ml_results<-data_frame()

# Function to compute all stats from models
evaluate_performance <- function(model_name, model, validation,model_results,title)
{
  # Generating predictions
  predictions<-predict(model, validation)

   # Draw the confusion matrix
  cm<-confusionMatrix(predictions,validation$LiverDisease,positive="M")

  draw_confusion_matrix(cm,title)

  # Generate metrics
  sensitivty<-as.numeric(cm$byClass[1])
  specificity<-as.numeric(cm$byClass[2])
  precision<-as.numeric(cm$byClass[5])
  recall<-as.numeric(cm$byClass[6])
  f1_score<-as.numeric(cm$byClass[7])
  accuracy<-as.numeric(cm$overall[1])
  kappa <-as.numeric(cm$overall[2])

  # Store metrics to a data frame
  ml_results <- bind_rows(ml_results, data_frame(Models = model_name,
              Accuracy = accuracy,
              Precision= precision,
              Sensitivty=sensitivty,
              Specificity=specificity,
              F1_Score = f1_score,
```

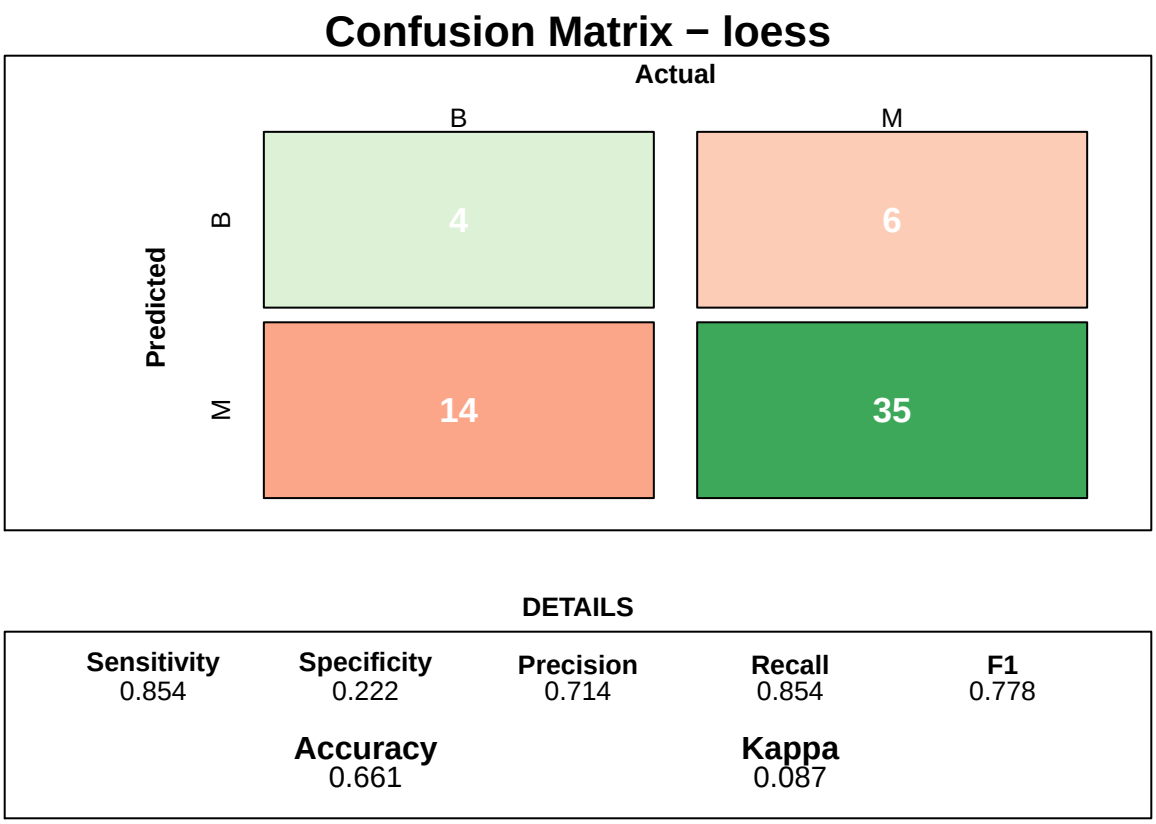
```
            Kappa= kappa))
}

# Evaluating the performance of models
ml_results<-evaluate_performance("glm",train_glm,validation,ml_results,"Confusion Matrix - glm")
```



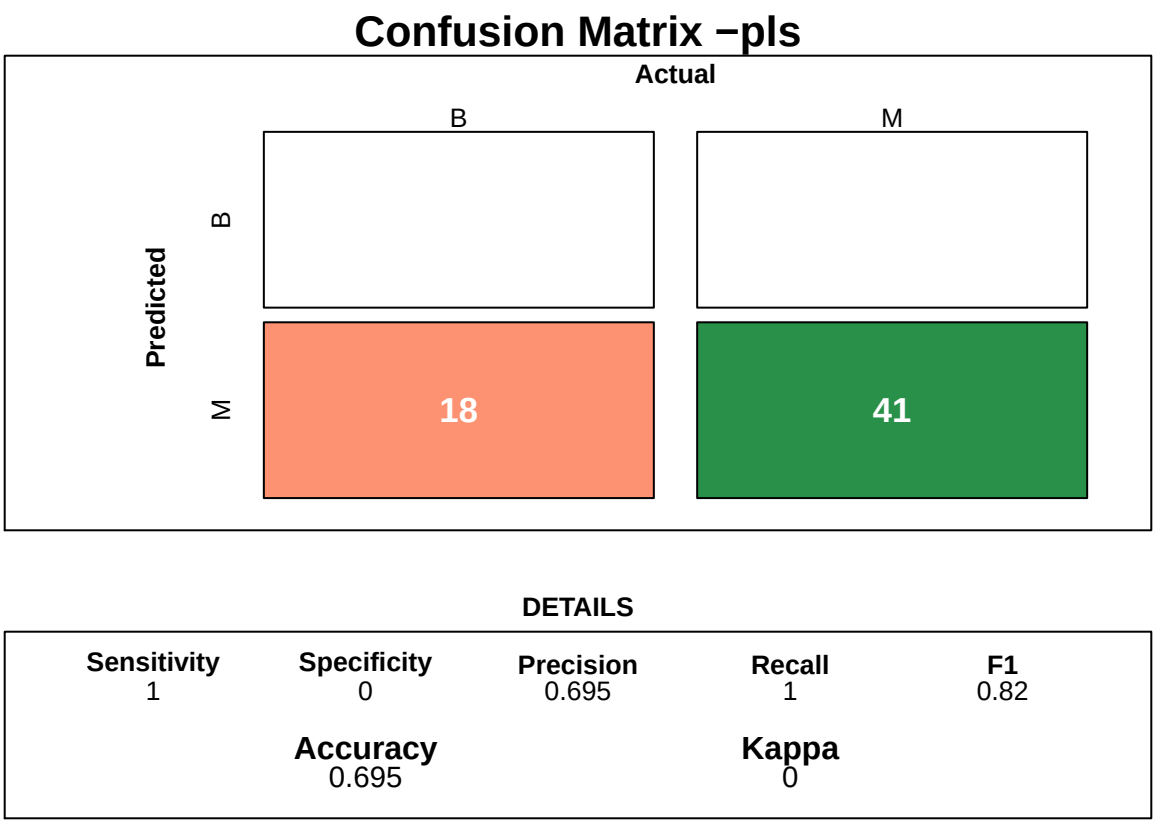

```
# Evaluate knn model
ml_results<-evaluate_performance("knn",train_knn,validation,ml_results,"Confusion Matrix - knn")
```



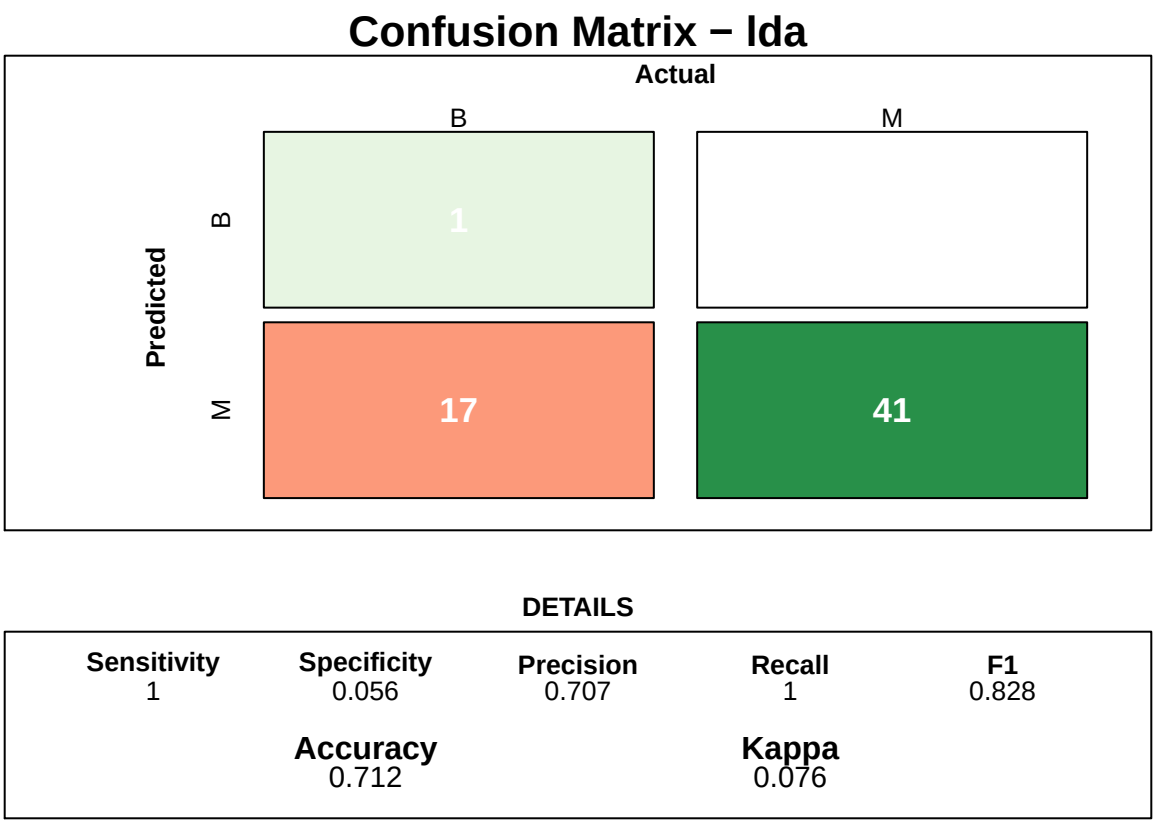

```
#Evaluate loess model
ml_results<-evaluate_performance("loess",train_loess,validation,ml_results,"Confusion Matrix - loess")
```



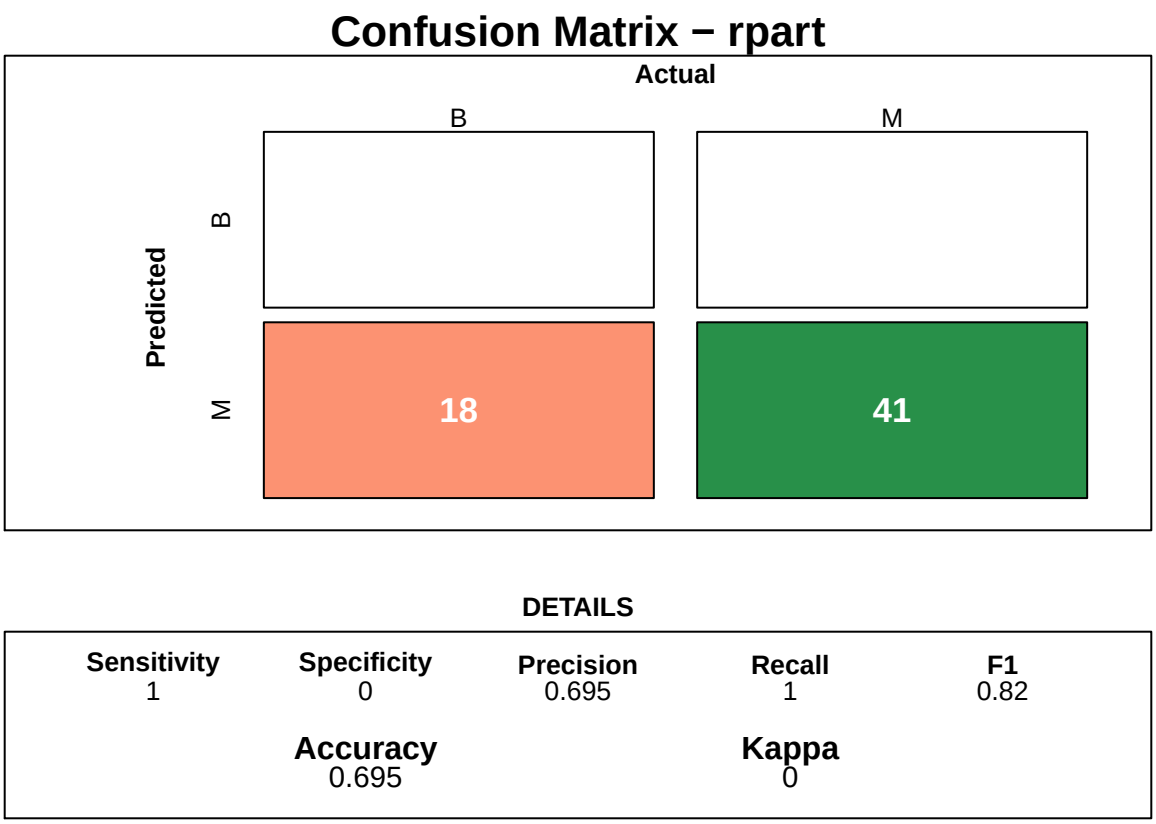

```
ml_results<-evaluate_performance("pls",train_pls,validation,ml_results,"Confusion Matrix -pls")
```




```
ml_results<-evaluate_performance("lda",train_lda,validation,ml_results, "Confusion Matrix - lda")
```




```
ml_results<-evaluate_performance("rpart",train_rpart,validation,ml_results, "Confusion Matrix - rpart")
```

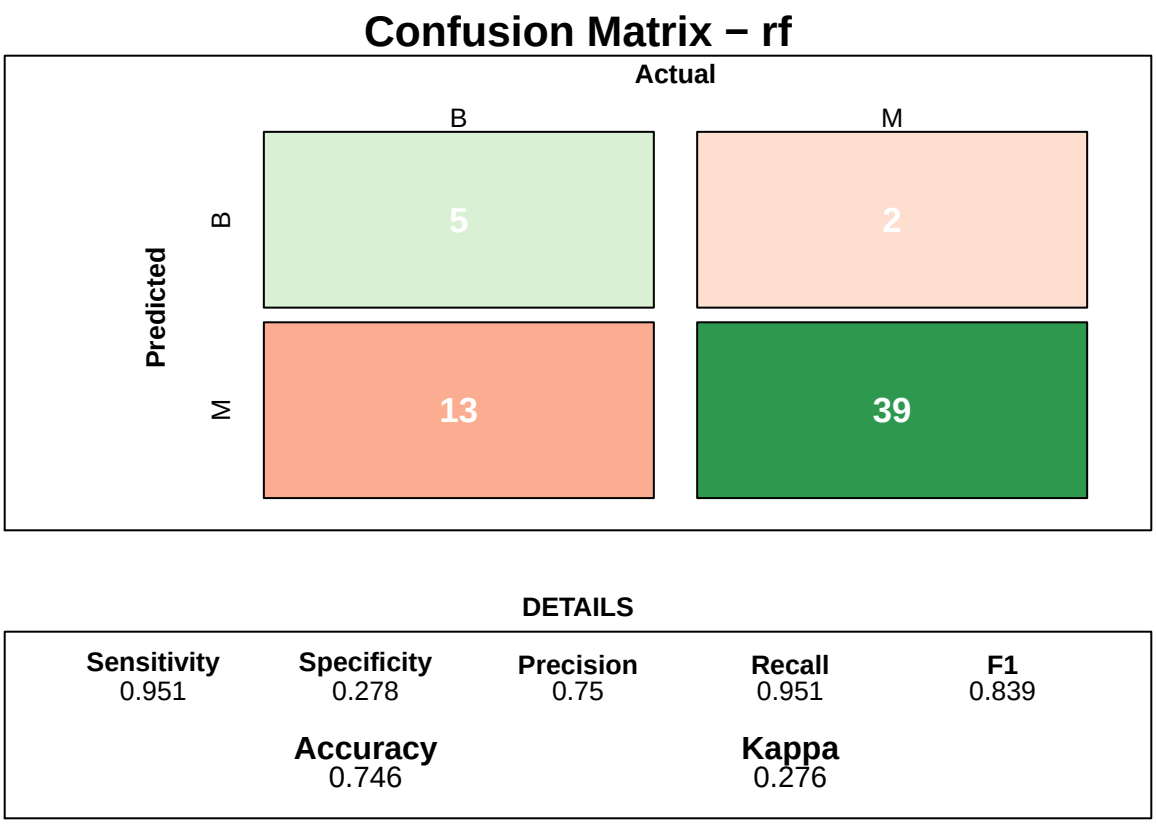


DETAILS

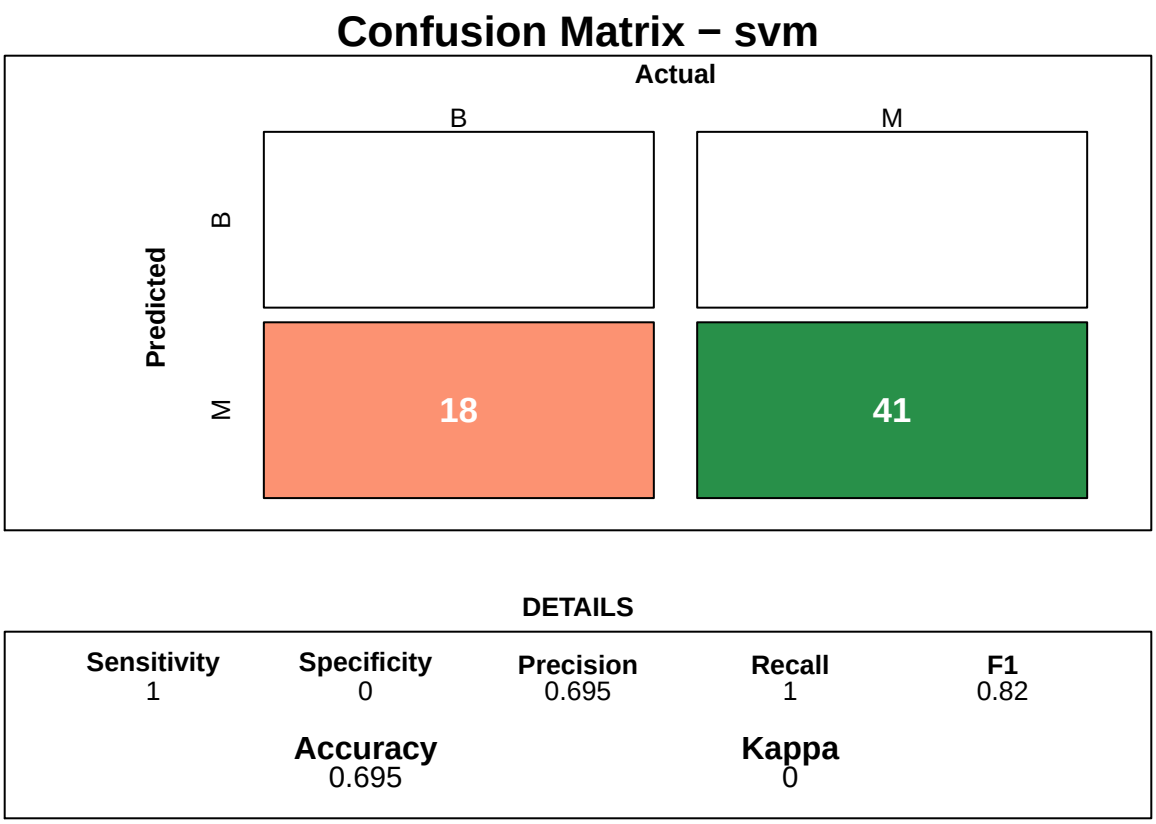
Sensitivity	Specificity	Precision	Recall	F1
1	0	0.695	1	0.82

Accuracy	Kappa
0.695	0

```
ml_results<-evaluate_performance("rf",train_rf,validation,ml_results,"Confusion Matrix - rf")
```




```
ml_results<-evaluate_performance("svmlinear",train_svm,validation,ml_results, "Confusion Matrix - svm")
```

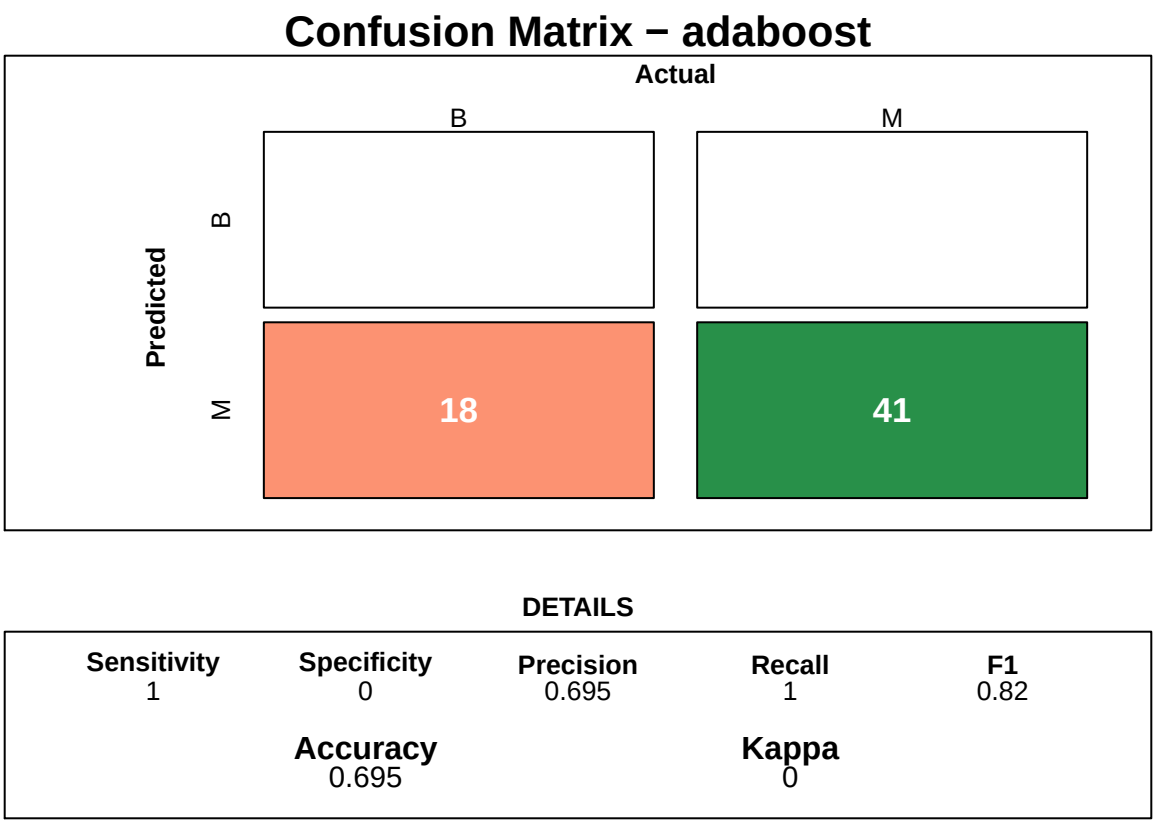


DETAILS

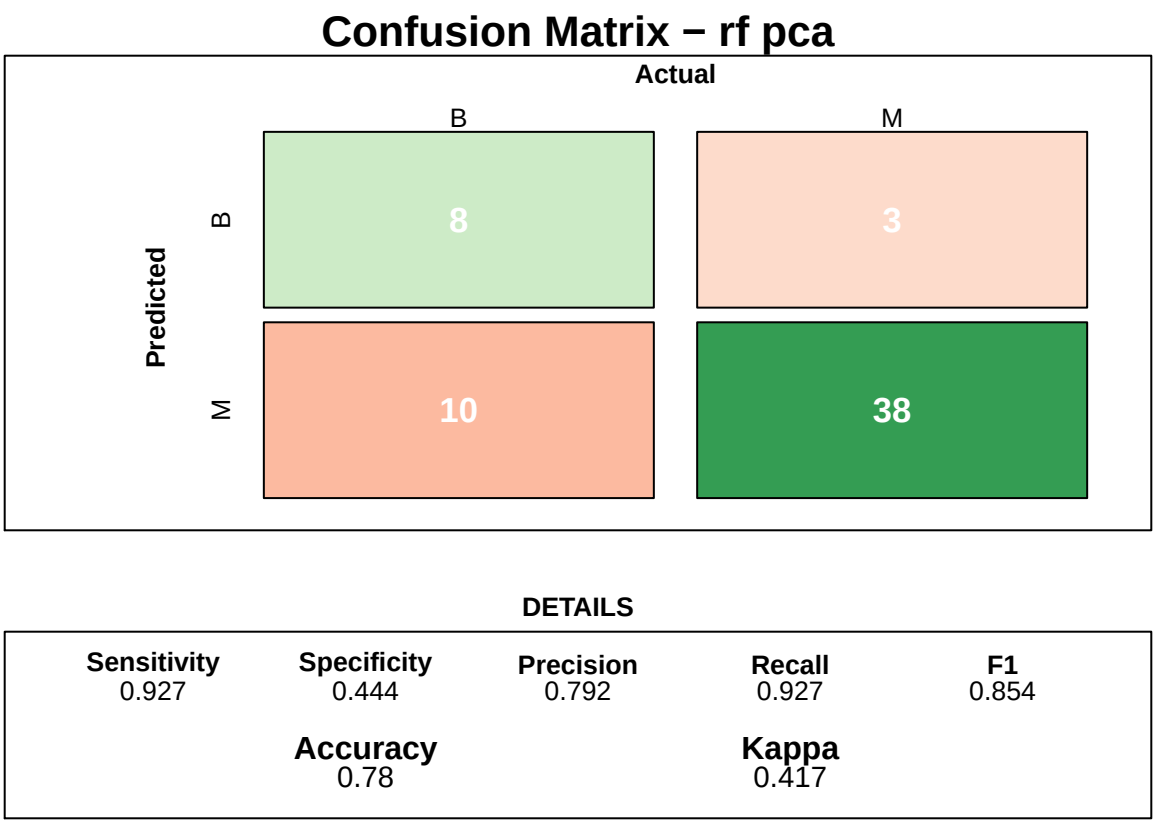
Sensitivity	Specificity	Precision	Recall	F1
1	0	0.695	1	0.82

Accuracy	Kappa
0.695	0

```
ml_results<-evaluate_performance("ada",train_ada,validation,ml_results,"Confusion Matrix - adaboost")
```



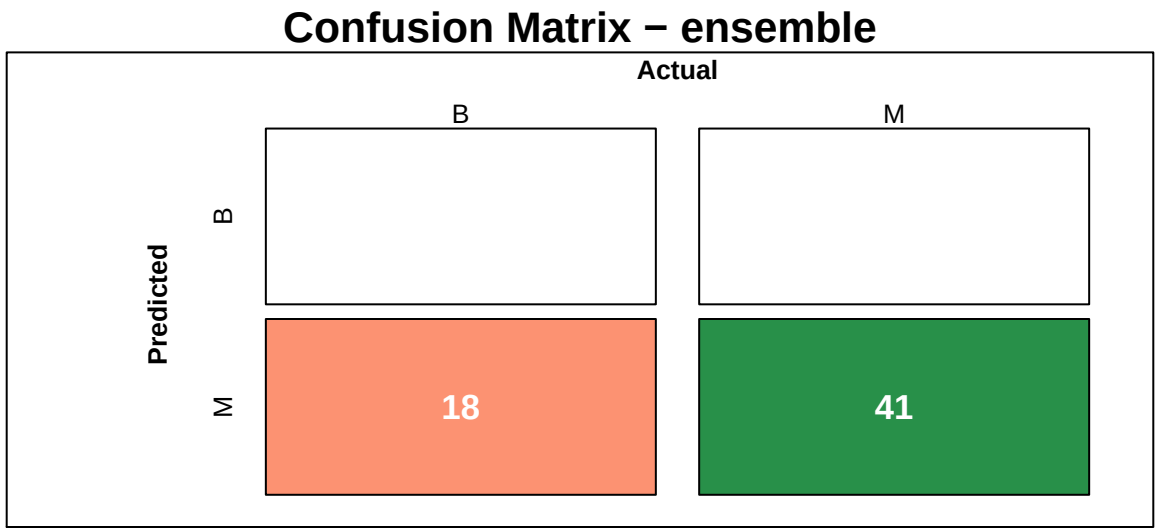

```
ml_results<-evaluate_performance("rf_pca",train_rf_pca,validation,ml_results,"Confusion Matrix - rf pca
```



Now, we do an experiment to combine the predictions of multiple models, i.e., ensemble model. The idea is to diagnose a liver disease only if 50% of the predictions from different models vote that the liver has a disease. The accuracy of the model and the resulting confusion matrix is not good to consider it for further analysis.

```
# Generating prediction of all models
glm_predictions<-predict(train_glm, validation)
knn_predictions<-predict(train_knn, validation)
loess_predictions<-predict(train_loess, validation)
pls_predictions<-predict(train_pls, validation)
lda_predictions<-predict(train_lda, validation)
rpart_predictions<-predict(train_rpart, validation)
rf_predictions<-predict(train_rf, validation)
svmlinear_predictions<-predict(train_svm, validation)
ada_predictions<-predict(train_ada, validation)
rf_pca_predictions<-predict(train_rf_pca, validation)

# Generate outputs fpr ensemble model
ensemble_pred<-data.frame(glm_predictions,
                          knn_predictions,
                          loess_predictions,
                          pls_predictions,
                          lda_predictions,
                          rpart_predictions,
                          rf_predictions,
                          svmlinear_predictions,
```

```
                          ada_predictions,
                          rf_pca_predictions)

# If 50% of the predictions say disease then we pick it a disease
votes <- rowMeans(ensemble_pred=="M")
ensemble_predictions <- ifelse(votes > 0.5, "M", "B") %>% factor()

# Generate metrics
cm<-confusionMatrix(ensemble_predictions,validation$LiverDisease,positive="M")
draw_confusion_matrix(cm,"Confusion Matrix - ensemble")
```



DETAILS

Sensitivity	Specificity	Precision	Recall	F1
1	0	0.695	1	0.82

Accuracy	Kappa
0.695	0

```
sensitivty<-as.numeric(cm$byClass[1])
specificity<-as.numeric(cm$byClass[2])
precision<-as.numeric(cm$byClass[5])
recall<-as.numeric(cm$byClass[6])
f1_score<-as.numeric(cm$byClass[7])
accuracy<-as.numeric(cm$overall[1])
kappa <-as.numeric(cm$overall[2])

# Store metrics to a data frame
ml_results <- bind_rows(ml_results, data_frame(Models = "Ensemble",
              Accuracy = accuracy,
              Precision= precision,
              Sensitivty=sensitivty,
              Specificity=specificity,
```

```
            F1_Score = f1_score,
            Kappa= kappa))
```

From the results, we can see that \emph{rf_pca} performs the best compared to the other models. We saw a significant improvement in *kappa* and *Specifivity* measures compared to the other models. Though, the model has a slightly lower sensitivity compared to the other models.

```
ml_results
```

Models	Accuracy	Precision	Sensitivty	Specificity	F1_Score	Kappa
glm	0.7118644	0.7142857	0.9756098	0.1111111	0.8247423	0.1131742
knn	0.7118644	0.7068966	1.0000000	0.0555556	0.8282828	0.0755760
loess	0.6610169	0.7142857	0.8536585	0.2222222	0.7777778	0.0866873
pls	0.6949153	0.6949153	1.0000000	0.0000000	0.8200000	0.0000000
lda	0.7118644	0.7068966	1.0000000	0.0555556	0.8282828	0.0755760
rpart	0.6949153	0.6949153	1.0000000	0.0000000	0.8200000	0.0000000
rf	0.7457627	0.7500000	0.9512195	0.2777778	0.8387097	0.2763696
svmlinear	0.6949153	0.6949153	1.0000000	0.0000000	0.8200000	0.0000000
ada	0.6949153	0.6949153	1.0000000	0.0000000	0.8200000	0.0000000
rf_pca	0.7796610	0.7916667	0.9268293	0.4444444	0.8539326	0.4167300
Ensemble	0.6949153	0.6949153	1.0000000	0.0000000	0.8200000	0.0000000

7 Conclusion

In this project, we developed machine learning models to diagnose liver disease by analysing protein levels in the blood. We used patients' liver records collected from India. We found out that some variables did not correlate with the presence or absence of liver disease. So, we ignored those variables and used the remaining variables to train the model.

The results show that *rf+pca* performed the best on the validation dataset. We tried to further improve the results by combining the outputs of several models. The idea was to use votes to decide if the liver has a disease. If more than 50% of the models predict a disease, then we consider that the liver is damaged. But, the resulting model did not perform well.

All models seem to overfit the data, and we end up getting more "Malignant" cases. The \emph{rf_pca} performs better, and we can correctly predict some "Benign" cases. The only reason we can think of is data is imbalanced. Ideally, both classes should have an equal distribution for patient records to train robust models. But, only 28% of the data belong to patients with liver disease. That's why the trained models are more biased in wrongly categorizing healthy patients. The ideal solution is to get more data to produce robust models. Alternatively, we can class weights to solve the problem of imbalanced data. We will explore the concept of class weights in future work.

References

[1] Ethan Du-Crowa, Lucy Warrenb, Susan M Astleya and Johan Hullemanc,"Is there a safety-net effect with Computer-Aided Detection (CAD)?", Medical Imaging 2019.

[2] Eugene, R., Sorrell, Michael F.; Maddrey, Willis C., "Schiff's Diseases of the Liver", 10th Edition, Lippincott Williams & Wilkins by Schiff.

[3] Bendi, Venkata . R, M. S. Prasad Babu, and N. B. Venkateswarlu, "Critical Comparative Study of Liver Patients from USA and INDIA: An Exploratory Analysis", International Journal of Computer Science Issues, May 2012.