Project Report - Predicting Liver Disease

Bhathiya Maneendra Pilanawithana

2023-04-07

1. Introduction

The project is aimed at developing a predictive algorithm for liver disease using patient records from India, which are publicly available on Kaggle.com through the link "https://www.kaggle.com/datasets/uciml/indian-liver-patient-records"".

Liver disease is a significant health problem worldwide, and its early detection is crucial because it allows for prompt treatment, which can prevent the progression of the disease and potentially save lives. The liver is a vital organ that performs numerous functions in the body, such as filtering toxins from the blood, producing bile, and metabolizing drugs. Liver disease can develop over time and may not present any symptoms until it has progressed to an advanced stage. Early detection can help identify the disease before symptoms become severe and irreversible damage has occurred. Furthermore, some types of liver disease, such as viral hepatitis, can be highly contagious, making early detection even more important in preventing the spread of the disease to others. Additionally, early detection of liver disease can enable healthcare professionals to closely monitor and manage the condition, which can help reduce the risk of complications and improve overall health outcomes. This may involve lifestyle changes, medication, and in some cases, surgery or liver transplantation.

In this project, various machine learning will be applied and fine tuned to predict the likelihood of liver disease in patients.

2. Dataset Description

This is extracted from Kaggle.The data set contains 416 liver patient records and 167 non liver patient records collected from North East of Andhra Pradesh, India. Any patient whose age exceeded 89 is listed as being of age "90".

Columns:

1. Age of the patient
2. Gender of the patient
3. Total Bilirubin
4. Direct Bilirubin
5. Alkaline Phosphotase
6. Alamine Aminotransferase
7. Aspartate Aminotransferase
8. Total Protiens
9. Albumin
10. Albumin and Globulin Ratio
11. Dataset: field used to split the data into two sets (patient with liver disease, or no disease)

3. Data Wrangling

First, the csv file stored in "Raw-Dataset" subfolder is loaded in to a dataframe using following code.

```
dat <- read.csv("./Raw-Dataset/indian_liver_patient.csv")
```

It was observed that presence of liver disease is stated in the "Dataset" column of the Dataframe and the existence of liver disease is denoted by 1 and non-existence is denoted by 2. Since this notation is counter intuitive, "Dataset" Column is renamed to "Disease" and existence and non-existence of liver disease is denoted by 1 and 0 respectively. Apart from this, the "Gender" column of the Dataframe is a character vector with on "Male" and "Female" entries. This should also be converted to a factor vector. Both of these can be achieved by the following code chunk.

```
dat <- dat %>%
  mutate(Disease = as.factor(ifelse(Dataset==1,1,0)), Gender=as.factor(Gender)) %>%
  select(-Dataset)
```

Now, the existence of liver disease is noted in "Disease" column intuitively using 1 and 0.

By running the following code it was observed that there are missing values in the Dataframe and they are all in the "Albumin_and_Globulin_Ratio" column.

```
sum(is.na(dat %>% select(-Gender)))
```

```
## [1] 4
```

```
colSums(is.na(dat %>% select(-Gender)))
```

```
##                         Age            Total_Bilirubin
##                           0                          0
##           Direct_Bilirubin       Alkaline_Phosphotase
##                           0                          0
##   Alamine_Aminotransferase Aspartate_Aminotransferase
##                           0                          0
##             Total_Protiens                    Albumin
##                           0                          0
## Albumin_and_Globulin_Ratio                    Disease
##                           4                          0
```

Since there are 583 total observations, omission of the 4 rows with missing values was considered to have a negligible effect. The omission was carried out by the following code.

```
dat <- na.omit(dat)
```

The Dataframe is now considered ready for further statistical analysis to find out which variable to use for prediction of liver disease.

4. Analysis

We start with analyzing individual variables to find out any apparent patterns, variable ranges suggestive of log transformations. Next, correlations between variables are calculated, which may be suggestive of linear relationships among variables. The data is then preprocessed using identified patterns, transformations and to eliminate dependent variables. Cross validation then is used for model selection and hyper parameter tuning.

First, we will analyse the gender influence in liver disease. The portion of people with liver disease for male and female genders can be extracted from the following code.

```
mean((dat %>% filter(Gender=="Male"))$Disease==1)
```

```
## [1] 0.7357631
```

```
mean((dat %>% filter(Gender=="Female"))$Disease==1)
```

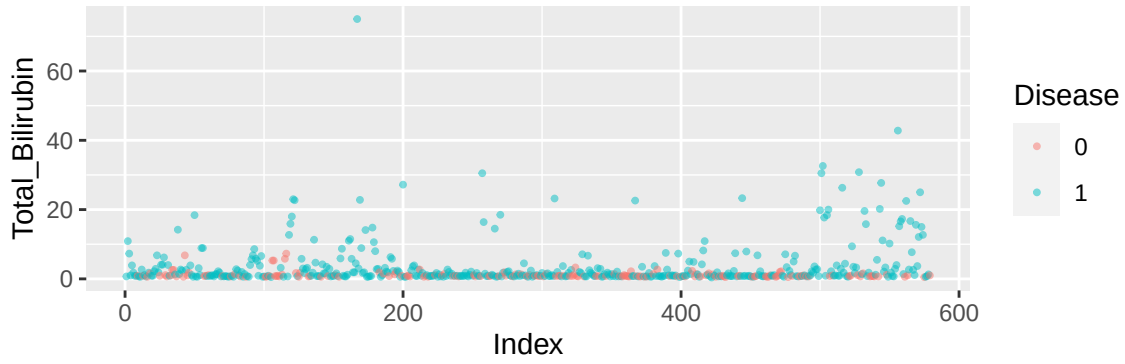
```
## [1] 0.65
```

It was observed, that according to the sample in the dataset both genders have more than 50% chance of having liver disease, which might not be case for the populations. However, gender male has higher likelyhood for liver disease, which suggests that the column gender may play an important part in liver disease prediction.

4.1 Checking for Outliers and Visible Patterns of Indivugual Numerical Variables

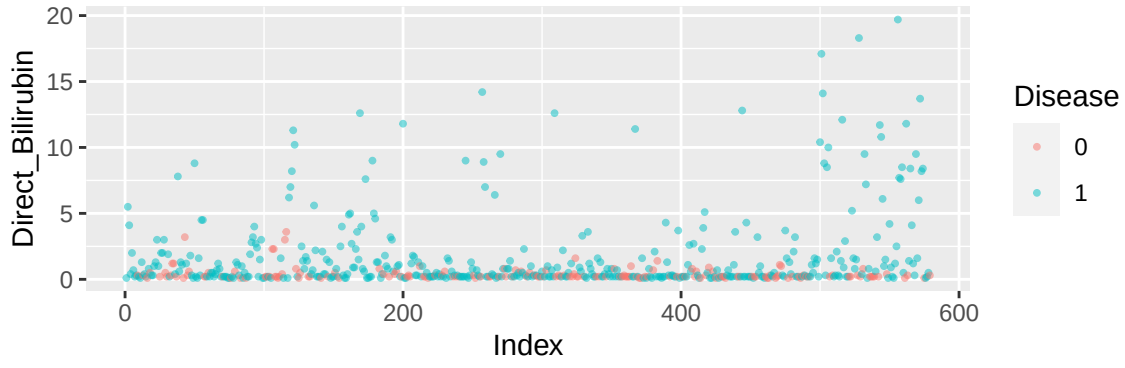
Each variable is plotted individually to identify any outliers.

```
dat %>% ggplot(aes(x = seq(1,nrow(dat),1), y = Total_Bilirubin)) +
  geom_point(aes(col=as.factor(Disease)), alpha = 0.5, size = 0.7) +
  guides(color = guide_legend(title = "Disease")) +
  xlab("Index")
```

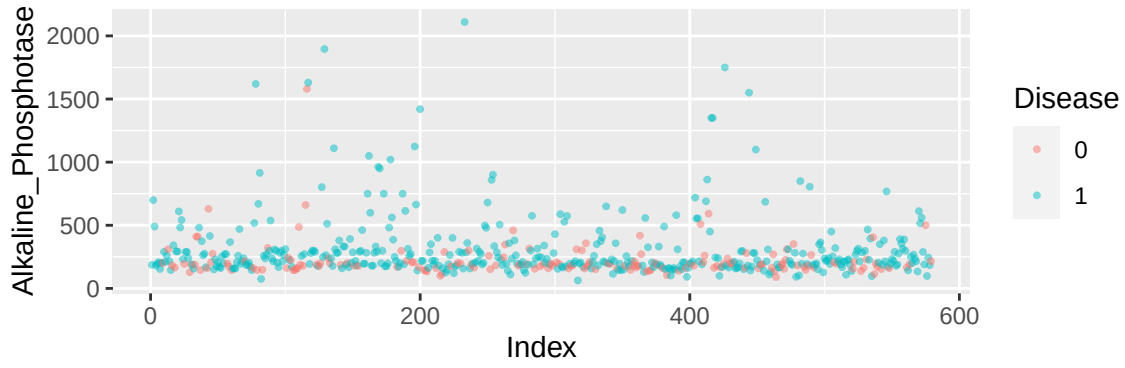


Although the plot of "Total Bilirubin" appears to have one element with very high value, it is not extreme enough to be considered an outlier.

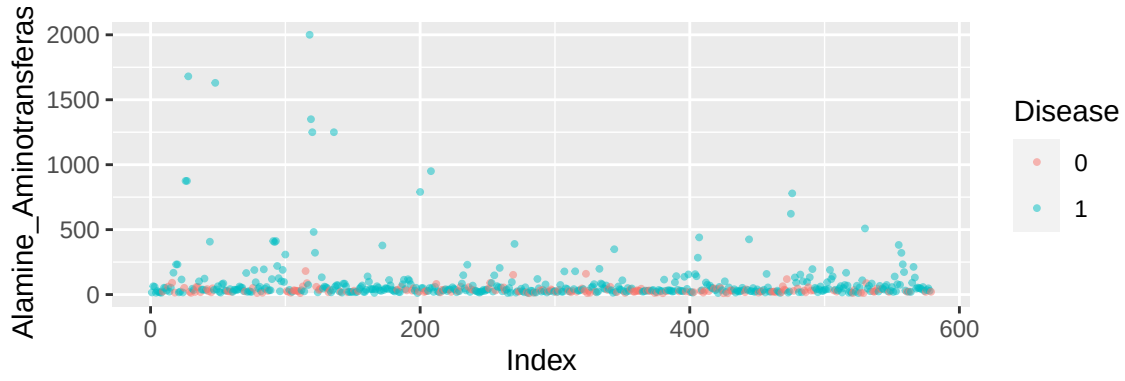
```
dat %>% ggplot(aes(x = seq(1,nrow(dat),1), y = Direct_Bilirubin)) +
  geom_point(aes(col=as.factor(Disease)), alpha = 0.5, size = 0.7) +
  guides(color = guide_legend(title = "Disease")) +
  xlab("Index")
```



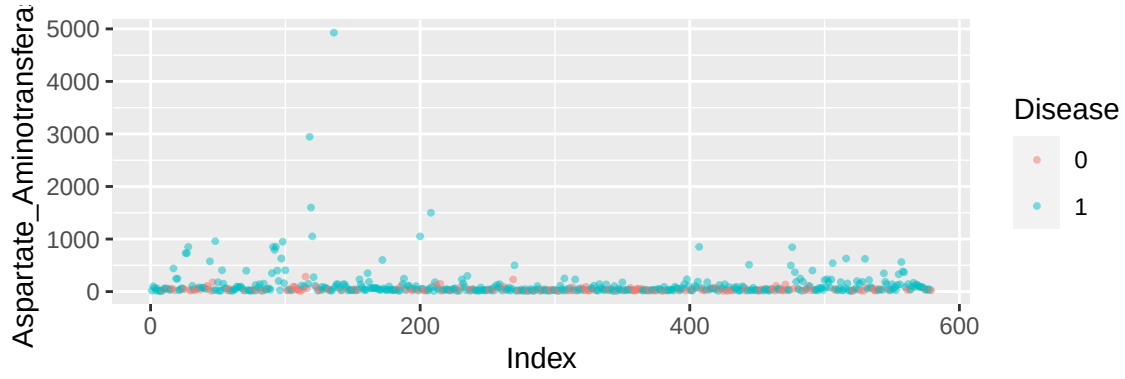
```
dat %>% ggplot(aes(x = seq(1,nrow(dat),1), y = Alkaline_Phosphotase)) +
  geom_point(aes(col=as.factor(Disease)), alpha = 0.5, size = 0.7) +
  guides(color = guide_legend(title = "Disease")) +
  xlab("Index")
```



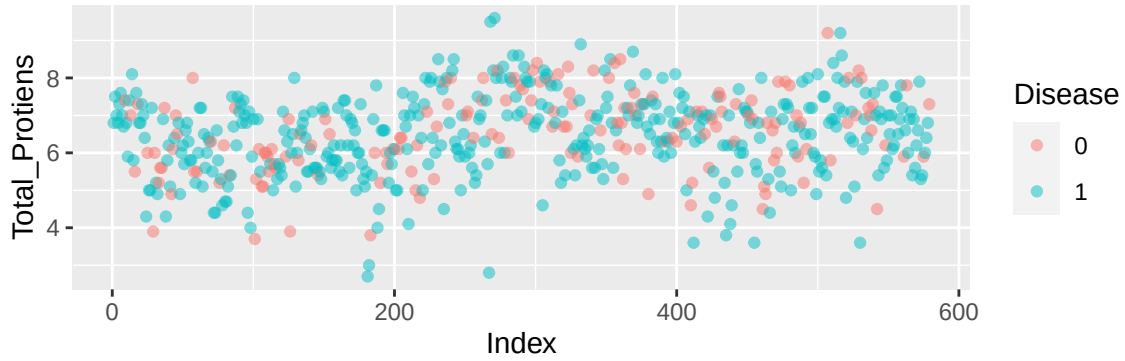

```
dat %>% ggplot(aes(x = seq(1,nrow(dat),1), y = Alamine_Aminotransferase)) +
  geom_point(aes(col=as.factor(Disease)), alpha = 0.5, size = 0.7) +
  guides(color = guide_legend(title = "Disease")) +
  xlab("Index")
```



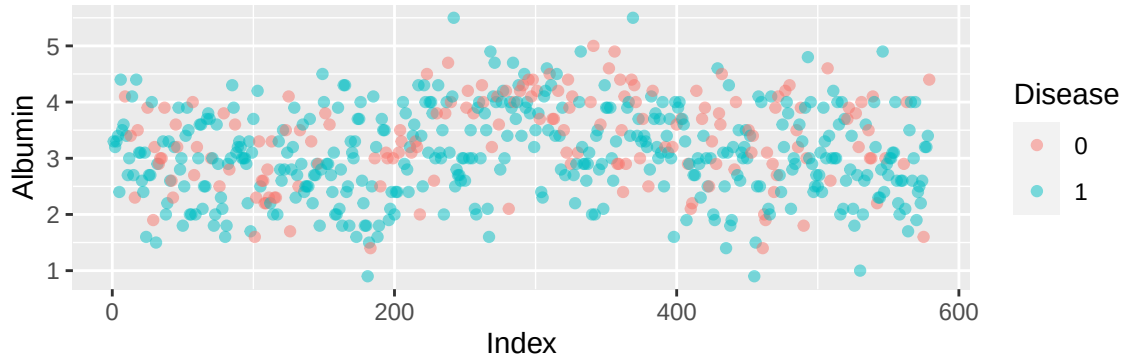

```
dat %>% ggplot(aes(x = seq(1,nrow(dat),1), y = Aspartate_Aminotransferase)) +
  geom_point(aes(col=as.factor(Disease)), alpha = 0.5, size = 0.7) +
  guides(color = guide_legend(title = "Disease")) +
  xlab("Index")
```



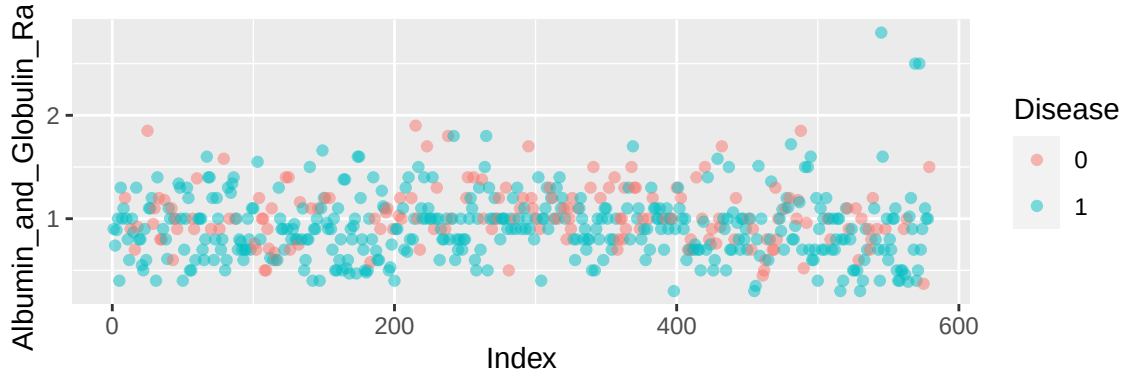
```
dat %>% ggplot(aes(x = seq(1,nrow(dat),1), y = Total_Protiens)) +
  geom_point(aes(col=as.factor(Disease)), alpha = 0.5) +
  guides(color = guide_legend(title = "Disease")) +
  xlab("Index")
```




```
dat %>% ggplot(aes(x = seq(1,nrow(dat),1), y = Albumin)) +
  geom_point(aes(col=as.factor(Disease)), alpha = 0.5) +
  guides(color = guide_legend(title = "Disease")) +
  xlab("Index")
```




```
dat %>% ggplot(aes(x = seq(1,nrow(dat),1), y = Albumin_and_Globulin_Ratio)) +
  geom_point(aes(col=as.factor(Disease)), alpha = 0.5) +
  guides(color = guide_legend(title = "Disease")) +
  xlab("Index")
```



Individual plots do not give away clues of visible patterns of individual variables. However, from the above plots it can be observed that following variables have large range, but have more points close to zero than

away from zero.

```
Range_Total_Bilirubin <- range(dat$Total_Bilirubin)
Range_Total_Bilirubin
```

```
## [1]  0.4 75.0
```

```
Range_Direct_Bilirubin <- range(dat$Direct_Bilirubin)
Range_Direct_Bilirubin
```

```
## [1]  0.1 19.7
```

```
Range_Alkaline_Phosphotase <- range(dat$Alkaline_Phosphotase)
Range_Alkaline_Phosphotase
```

```
## [1]   63 2110
```

```
Range_Alamine_Aminotransferase <- range(dat$Alamine_Aminotransferase)
Range_Alamine_Aminotransferase
```

```
## [1]   10 2000
```

```
Range_Aspartate_Aminotransferase <- range(dat$Aspartate_Aminotransferase)
Range_Aspartate_Aminotransferase
```

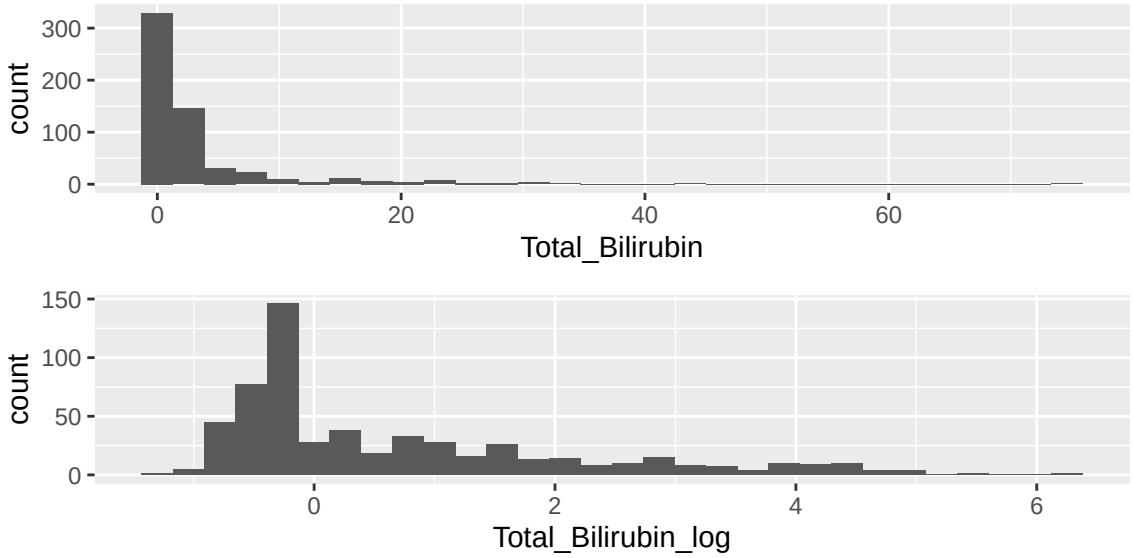
```
## [1]   10 4929
```

This suggests that log transformations applied to each of the following variables may improve the performance of a prediction algorithm. Log transformations are applied using the following code. The new Dataframe with log transformations is named data_log.

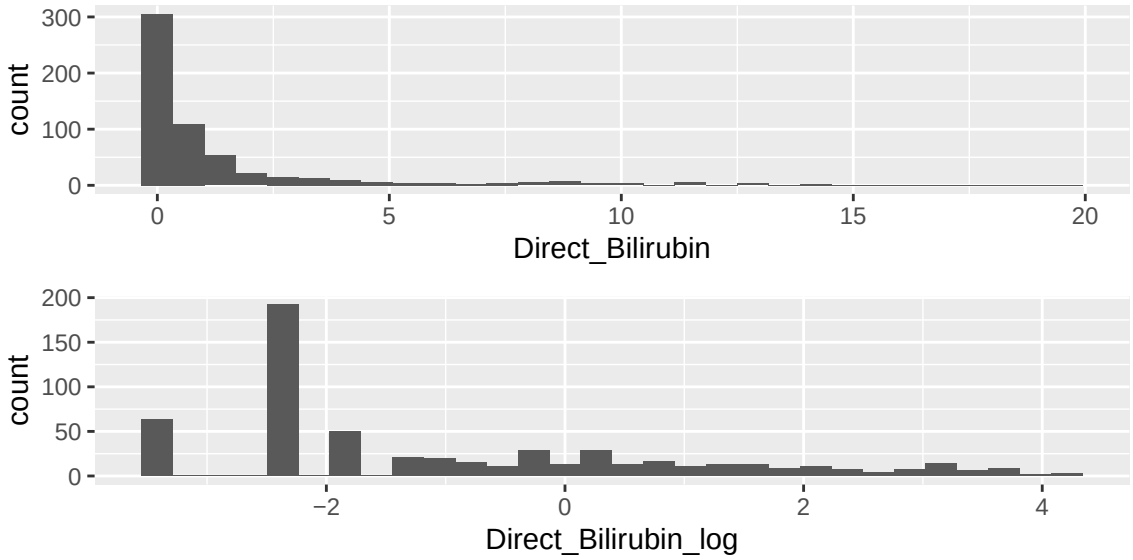
```
dat_log <- dat %>%
  mutate(Total_Bilirubin_log = log2(Total_Bilirubin),
         Direct_Bilirubin_log = log2(Direct_Bilirubin),
         Alkaline_Phosphotase_log = log10(Alkaline_Phosphotase),
         Alamine_Aminotransferase_log = log10(Alamine_Aminotransferase),
         Aspartate_Aminotransferase_log = log10(Aspartate_Aminotransferase)) %>%
  select(-Total_Bilirubin, -Direct_Bilirubin, -Alkaline_Phosphotase,
         -Alamine_Aminotransferase, -Aspartate_Aminotransferase)
```

The following histograms show a comparison of histograms for each high range variable before applying log transformation and after, respectively.

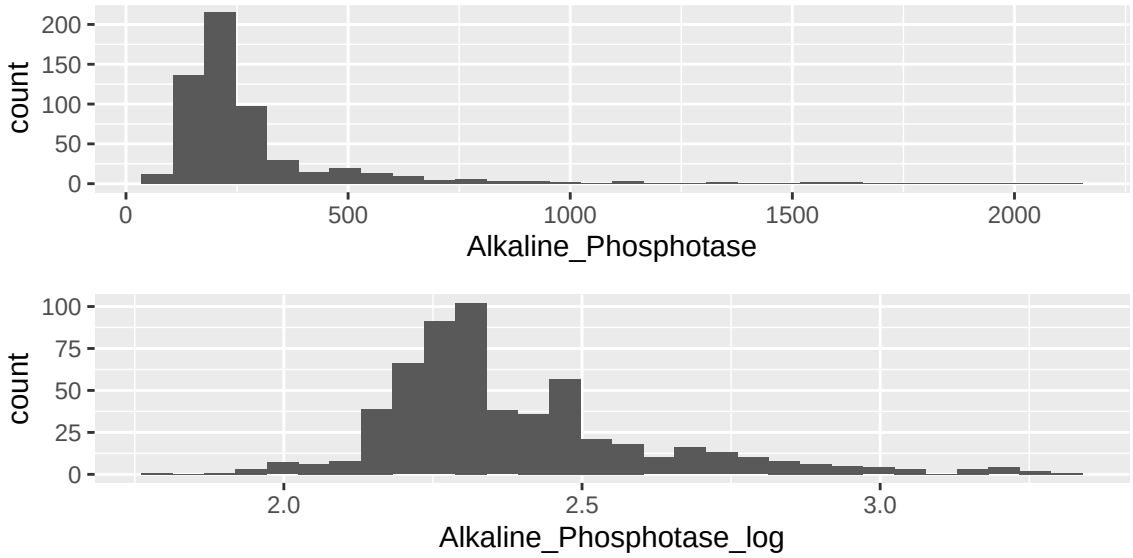
Total Bilirubin:



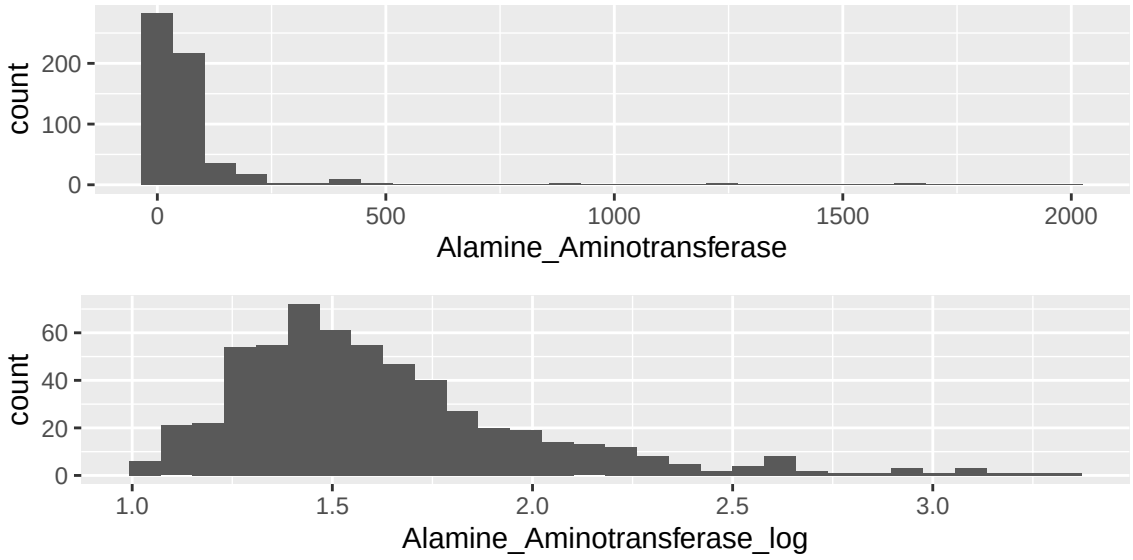
Direct Bilirubin:



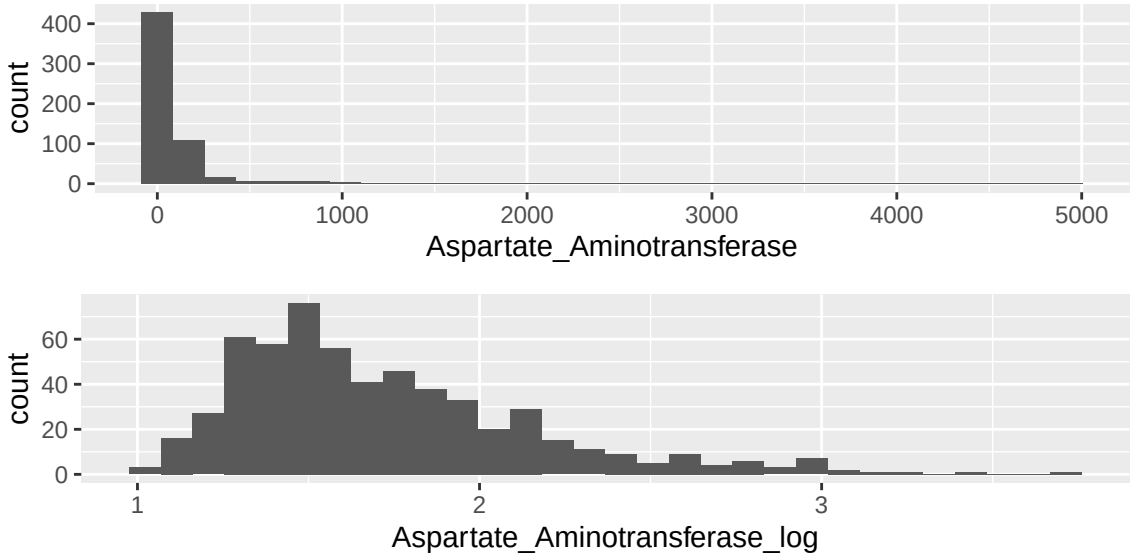
Alkaline Phosphotase:



Alamine Aminotransferase:



Aspartate Aminotransferase:



4.2 Correlation Between Numerical Variables

The correlation matrix for the numerical variables can be obtained using cor() function. However, the variable names are too long an effective presentation. Therefore, a temporary Dataframe called dat_for_corr is created with name abbreviations.

```
dat_log_for_corr <- dat_log %>% select(-Age,-Gender,-Disease)
colnames(dat_log_for_corr) <-
  c("Tot Pr","Alb", "Alb Glb", "Tot Bil", "Dir Bil", "ALP", "ALA", "ASA")
cr <- round(cor(dat_log_for_corr),4)
cr
```

```
##          Tot Pr     Alb Alb Glb Tot Bil Dir Bil     ALP     ALA     ASA
## Tot Pr   1.0000  0.7831  0.2349 -0.0514 -0.0354  0.0021 -0.0133 -0.0638
## Alb      0.7831  1.0000  0.6896 -0.2901 -0.2703 -0.1679 -0.0415 -0.1780
## Alb Glb  0.2349  0.6896  1.0000 -0.2706 -0.2631 -0.2822 -0.0609 -0.1448
## Tot Bil -0.0514 -0.2901 -0.2706  1.0000  0.9657  0.3585  0.4424  0.5406
## Dir Bil -0.0354 -0.2703 -0.2631  0.9657  1.0000  0.3568  0.4297  0.5300
## ALP      0.0021 -0.1679 -0.2822  0.3585  0.3568  1.0000  0.3396  0.3195
## ALA     -0.0133 -0.0415 -0.0609  0.4424  0.4297  0.3396  1.0000  0.8416
## ASA     -0.0638 -0.1780 -0.1448  0.5406  0.5300  0.3195  0.8416  1.0000
```

It can be observed from the correlation matrix that several variables may have linear relationship between them. For example following groups of variables have correlations larger than 0.5 between each other.

1. Total Protiens - Albumin - Albumin_and_Globulin_Ratio
2. Alamine Aminotransferase - Aspartate Aminotransferase - Total Bilirubin - Direct Bilirubin

These relationships may be exploited using Principal Component Analysis (PCA) while applying machine learning methods.

4.3 Applying Machine Learning

To apply machine learning we first partition the Dataset in to training-set 60% and test-set 40% using the following code.

```
set.seed(1)
test_index <- createDataPartition(dat_log$Disease, times=1, p=0.4, list=FALSE)
test_set <- dat_log[test_index,]
train_set <- dat_log[-test_index,]
```

4.3.1 Preprocessing and Variable Transformation We first preprocess the training set and test using a PCA trained on training set. The number of PCA components are selected to be 8 since there are 9 numerical variables in total. This values will later be fine tuned once a suitable machine learning method is developed. Preprocessing based on PCA can be achieved using the following code.

```
preProc <- preProcess(train_set, method = c("pca"), pcaComp = 8)
train_set_transformed <- predict(preProc, train_set)
test_set_transformed <- predict(preProc, test_set)
```

The above code omits the non-numerical columns in the training and test sets.

4.3.2 Model Selection We intends to apply kNN, Decision-Tree, Random-Forest and SVM with Radial Kernal and fine tune their parameters using cross-validation. We will compare the validation accuracy of each fine-tuned model and select the best one for the prediction of liver disease.

1. K-Nearest-Neighbor

The following code is used for cross-validation of kNN for k.

```
set.seed(2)
k <- seq(2,100,2)
fit_knn <- train(Disease ~ .,
                 data = train_set_transformed,
                 method = "knn",
                 tuneGrid = data.frame(k))
knn_validation_acc <- max(fit_knn$results$Accuracy) #Holds kNN Validation Accuracy
```

2. Decision-Tree

The following code is used for cross-validation of Decision-Tree for its cp parameter.

```
set.seed(3)
cp <- seq(0, 0.2, 0.004)
fit_rpart <- train(Disease ~ .,
                   data=train_set_transformed,
                   method = "rpart",
                   tuneGrid = data.frame(cp = cp))
rpart_validation_acc <- max(fit_rpart$results$Accuracy) #Holds Decision-Tree Validation Accuracy
```

3. Random-Forest

Caret package cross-validation for Random-Forest does not automatically include ntree and nodesize. Therefore, ntree and nodesize parameters are fine-tuned manually using the following code.

```
set.seed(4)
ntree <- seq(100, 130, 5) #Hold vector for ntree
nodesize <- c(1,2,3) #Hold Vector for nodesize
exgrd <- expand.grid(ntree,nodesize) #Vector with all combinations of ntree and nodesize
vec <- seq(1,nrow(exgrd)) #Index vector to sweep through exgrd vector using sapply
```

```
acc <- sapply(vec, function(n){
  train(Disease ~.,
        data = train_set_transformed,
        method = "rf",
        tuneGrid = data.frame(mtry = 1),
        ntree = exgrd$Var1[n],
        nodesize = exgrd$Var2[n])$results$Accuracy
})
opt_mtry <- 1
opt_ntree <- exgrd$Var1[which.max(acc)] #Holds optimal ntree value
opt_nodesize <- exgrd$Var2[which.max(acc)] #Holds optimal nodesize value

#The following code build a Random-Forest predictor with optimal ntree and nodesize
set.seed(4)
fit_rf <- train(Disease ~.,
                data = train_set_transformed,
                method = "rf",
                tuneGrid = data.frame(mtry = opt_mtry),
                ntree = opt_ntree,
                nodesize = opt_nodesize)
rf_validation_acc <- max(fit_rf$results$Accuracy) #Holds Random-Forest Validation Accuracy
```

4. SVM with Radial Kernal

The following code is used for cross-validation of SVM with Radial Kernal for its sigma and C parameters.

```
set.seed(5)
C <- seq(0.1, 2, 20)
sigma <- seq(0.1, 2, 20)
fit_svmRadial <- train(Disease ~.,
                  data = train_set_transformed,
                  method = "svmRadial",
                  tuneGrid = data.frame(C = C, sigma = sigma))
svmRadial_validation_acc <- max(fit_svmRadial$results$Accuracy) #Holds Svm-Radial Validation Accuracy
```

The following code prints the obtained validation accuracies for each fine-tuned model.

```
knn_validation_acc
```

```
## [1] 0.7030122
```

```
rpart_validation_acc
```

```
## [1] 0.7139274
```

```
rf_validation_acc
```

```
## [1] 0.7383134
```

```
svmRadial_validation_acc
```

```
## [1] 0.7108151
```

5. Results

It can be observed from the above validation accuracies that Random-Forest with following parameters gives the maximum validation accuracy.

```
opt_mtry
```

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

```
opt_ntree
```

```
## [1] 115
```

```
opt_nodesize
```

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

We can now change the number of PCA components used during preprocessing and check whether there is a significant change in the validation accuracy. It was observed that pcaComp=7 gives a lower accuracy than pcaComp=8 and pcaComp=9 does not make a significant change. Therefore, pcaComp=8 is kept unchanged.

We use the Random-Forest model built together with PCA based preprocessing with 8 components for testing and final results.

The following code can be used to obtain and print accuracy, sensitivity, specificity, F1-score and other related metrices.

```
Disease_hat_rf_test <- predict(fit_rf, test_set_transformed)
cm <- confusionMatrix(Disease_hat_rf_test, test_set_transformed$Disease, positive = "1")
cm[["overall"]][["Accuracy"]]
```

```
## [1] 0.737069
```

```
cm[["byClass"]][["Sensitivity"]]
```

```
## [1] 0.9698795
```

```
cm[["byClass"]][["Specificity"]]
```

```
## [1] 0.1515152
```

```
cm[["byClass"]][["F1"]]
```

```
## [1] 0.8407311
```

```
cm[["byClass"]][["Pos Pred Value"]]
```

```
## [1] 0.7419355
```

```
cm[["byClass"]][["Neg Pred Value"]]
```

```
## [1] 0.6666667
```

It can be observed that the prediction results in a high sensitivity but with a low specificity. This may be due to the Dataset imbalance. The Dataset consists of significantly more number of cases with liver disease than cases without liver disease. Therefore, the prediction algorithm is inherently biased towards positive cases, which results in higher sensitivity and a lower specificity.

The values “Pos Pred Value” and “Neg Pred Value” gives the probability of liver desease when the prediction is positive and the probability of no liver disease when the prediction is negative, respectively. Out of these probabilities “Neg Pred Value” is more important for this case due to medical reasons. It can be observed from the above values that “Neg Pred Value” (Probability of no liver disease when prediction is negative) is larger than 0.5, which suggests that the prediction algorithm works better than simple guess work. Since, the prediction of liver disease should be supportive for medical decisions,

6. Conclusion

It was observed that a Random-Forest Predictor provides the best accuracy. However, the significant Dataset imbalance limits the algorithms performance. A Dataset, with more observations for the cases of no liver disease may increase the performance of prediction algorithm. However, the prediction algorithm developed using the current Dataset provides higher accuracy than guess work.

7. References

The Dataset was obtained from Kaggle.com which can be downloaded from "https://www.kaggle.com/datasets/uciml/indian-liver-patient-records" (accessed on 08/04/2013).