Liver Disease Project

Viktoriia Pylypets_Romaniuk

2023-03-12

Liver disease classification project

1. Introduction

Quick and accurate medical diagnoses are crucial for the successful treatment of diseases. The main goal of liver disease classification project is to train a machine learning algorithm that will predict if the patient has a liver disease or not with the highest possible accuracy, sensitivity and specificity, based on blood test result, age, and gender information. We put a lot of attention to specificity and sensitivity, because it is very important in medicine do not get false positive and false negative results. In the first case the patient will get unnecessary treatment, in the second case the patient will not get a treatment but needs it. For this project we got data set from Kaggle.com. This is the link: https://www.kaggle.com/datasets/abhi8923shriv/liver-disease-patient-dataset. This data set does not have a lot of projects on Kaggle.com site. In the Discussion section on the Kaggle.com we can find the description of the data set, given by the creator Abhishek Shrivastava. "This data set contains 10 variables, that are age, gender, total Bilirubin, direct Bilirubin, total proteins, albumin, A/G ratio, SGPT, SGOT, and Alkphos". Column #11 is "selector field used to split the data into two sets(assume, categories) (labeled by the experts) 1 Liver Patient, 2 Non Liver Patient"(1). For learning purposes we assume that 1 means patient who has the liver disease, and 2 means the patient, who does not. And columns 3-10 are the blood test results. The data set consists of two tables, for our project we used the first one. The dataset, that we used, can be find on GitHub, using this link(2) https://github.com/VictoriaRomano/LiverDisease/blob/main/liver_patient.xls.

2. Data exploratory and cleaning

We can see that our data set is a table of 30691 rows and 11 columns. Also columns have long names that is not convenient to work with.

```
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: tidyverse
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.3.6      v purrr   0.3.4
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.4.1
## v readr   2.1.3      v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: caret
## Loading required package: lattice
##
## Attaching package: 'caret'
```

```
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```
# import needed libraries
library(tidyverse)
library(dplyr)
library(tidyr)
library(caret)
library(readxl)
library(ggplot2)

#Import file
liver_patient <- read_excel("liver_patient.xls")
#Check the structure of the file
head(liver_patient)
```

```
## # A tibble: 6 x 11
##   Age of the p~1 Gende~2 Total~3 Direc~4  Alkp~5  Sgpt~6 Sgot ~7 Total~8  ALB ~9
##            <dbl> <chr>     <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
## 1             65 Female      0.7     0.1     187      16      18     6.8     3.3
## 2             62 Male       10.9     5.5     699      64     100     7.5     3.2
## 3             62 Male        7.3     4.1     490      60      68     7       3.3
## 4             58 Male        1       0.4     182      14      20     6.8     3.4
## 5             72 Male        3.9     2       195      27      59     7.3     2.4
## 6             46 Male        1.8     0.7     208      19      14     7.6     4.4
## # ... with 2 more variables: `A/G Ratio Albumin and Globulin Ratio` <dbl>,
## #   Result <dbl>, and abbreviated variable names 1: `Age of the patient`,
## #   2: `Gender of the patient`, 3: `Total Bilirubin`, 4: `Direct Bilirubin`,
## #   5: ` Alkphos Alkaline Phosphotase`, 6: ` Sgpt Alamine Aminotransferase`,
## #   7: `Sgot Aspartate Aminotransferase`, 8: `Total Protiens`,
## #   9: ` ALB Albumin`
```

```
str(liver_patient)
```

```
## tibble [30,691 x 11] (S3: tbl_df/tbl/data.frame)
##  $ Age of the patient                  : num [1:30691] 65 62 62 58 72 46 26 29 17 55 ...
##  $ Gender of the patient               : chr [1:30691] "Female" "Male" "Male" "Male" ...
##  $ Total Bilirubin                     : num [1:30691] 0.7 10.9 7.3 1 3.9 1.8 0.9 0.9 0.9 0.7 ...
##  $ Direct Bilirubin                    : num [1:30691] 0.1 5.5 4.1 0.4 2 0.7 0.2 0.3 0.3 0.2 ...
##  $  Alkphos Alkaline Phosphotase       : num [1:30691] 187 699 490 182 195 208 154 202 202 290 ...
##  $  Sgpt Alamine Aminotransferase      : num [1:30691] 16 64 60 14 27 19 NA 14 22 53 ...
##  $ Sgot Aspartate Aminotransferase     : num [1:30691] 18 100 68 20 59 14 12 11 19 58 ...
##  $ Total Protiens                      : num [1:30691] 6.8 7.5 7 6.8 7.3 7.6 7 6.7 7.4 6.8 ...
##  $  ALB Albumin                        : num [1:30691] 3.3 3.2 3.3 3.4 2.4 4.4 3.5 3.6 4.1 3.4 ...
##  $ A/G Ratio Albumin and Globulin Ratio: num [1:30691] 0.9 0.74 0.89 1 0.4 1.3 1 1.1 1.2 1 ...
##  $ Result                              : num [1:30691] 1 1 1 1 1 1 1 1 2 1 ...
```

That's why we change names of columns for shorter ones.

```
#Change names of columns for shorter ones and more usable
names(liver_patient) = c("age", "gender", "total_bil","direct_bil","aap","salam","sasam","total_prot","a
head(liver_patient)
```

```
## # A tibble: 6 x 11
##     age gender total_bil direct~1   aap salam sasam total~2   alb ag_ra~3 result
```

```
##   <dbl> <chr>      <dbl>    <dbl> <dbl> <dbl> <dbl>   <dbl> <dbl>   <dbl>  <dbl>
## 1    65 Female       0.7      0.1   187    16    18     6.8   3.3    0.9       1
## 2    62 Male        10.9      5.5   699    64   100     7.5   3.2    0.74      1
## 3    62 Male         7.3      4.1   490    60    68     7     3.3    0.89      1
## 4    58 Male         1        0.4   182    14    20     6.8   3.4    1         1
## 5    72 Male         3.9      2     195    27    59     7.3   2.4    0.4       1
## 6    46 Male         1.8      0.7   208    19    14     7.6   4.4    1.3       1
## # ... with abbreviated variable names 1: direct_bil, 2: total_prot, 3: ag_ratio
```

Next we check if we have some NA in the data set. We can see that we have 5425 cells with NA value.

```
#Check if there some NA
sum(is.na(liver_patient))
```

```
## [1] 5425
```

We(assumed) have blood test results and it is not safe to use some average values instead of NA for distinguishing if there is a disease. In this project we drop all rows with NA values.

```
# Drop rows with NA values
liver_data <- drop_na(liver_patient)
# Check if in the data no NA values left
sum(is.na(liver_data))
```

```
## [1] 0
```

After dropping all rows with NA values, we check the structure of our data set. Now it has 27 158 rows. It means we lost 3533 rows.It is not terrible for learning purposes and we continue to work with cleaned table.

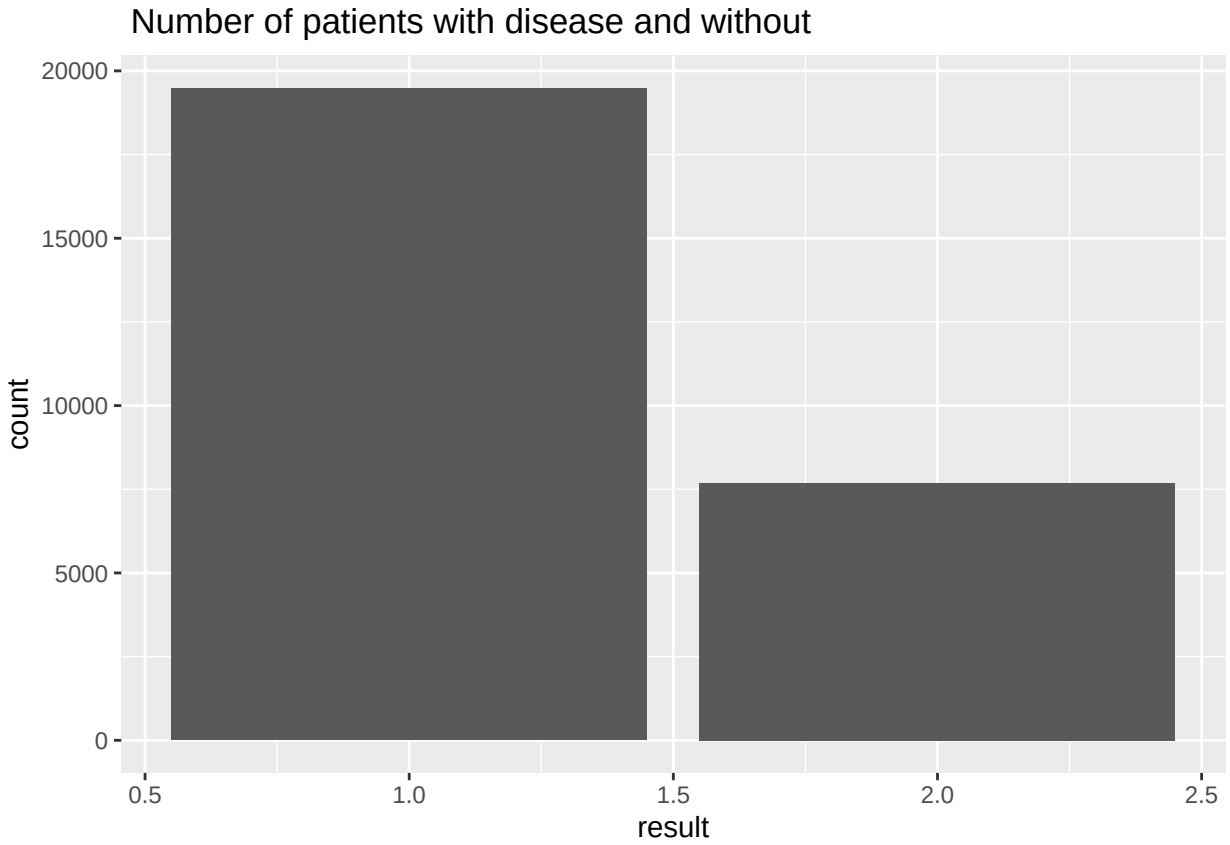
```
#Check the structure of the data without NA values
str(liver_data)
```

```
## tibble [27,158 x 11] (S3: tbl_df/tbl/data.frame)
##  $ age       : num [1:27158] 65 62 62 58 72 46 29 17 55 57 ...
##  $ gender    : chr [1:27158] "Female" "Male" "Male" "Male" ...
##  $ total_bil : num [1:27158] 0.7 10.9 7.3 1 3.9 1.8 0.9 0.9 0.7 0.6 ...
##  $ direct_bil: num [1:27158] 0.1 5.5 4.1 0.4 2 0.7 0.3 0.3 0.2 0.1 ...
##  $ aap       : num [1:27158] 187 699 490 182 195 208 202 202 290 210 ...
##  $ salam     : num [1:27158] 16 64 60 14 27 19 14 22 53 51 ...
##  $ sasam     : num [1:27158] 18 100 68 20 59 14 11 19 58 59 ...
##  $ total_prot: num [1:27158] 6.8 7.5 7 6.8 7.3 7.6 6.7 7.4 6.8 5.9 ...
##  $ alb       : num [1:27158] 3.3 3.2 3.3 3.4 2.4 4.4 3.6 4.1 3.4 2.7 ...
##  $ ag_ratio  : num [1:27158] 0.9 0.74 0.89 1 0.4 1.3 1.1 1.2 1 0.8 ...
##  $ result    : num [1:27158] 1 1 1 1 1 1 1 2 1 1 ...
```

3. Data analysis

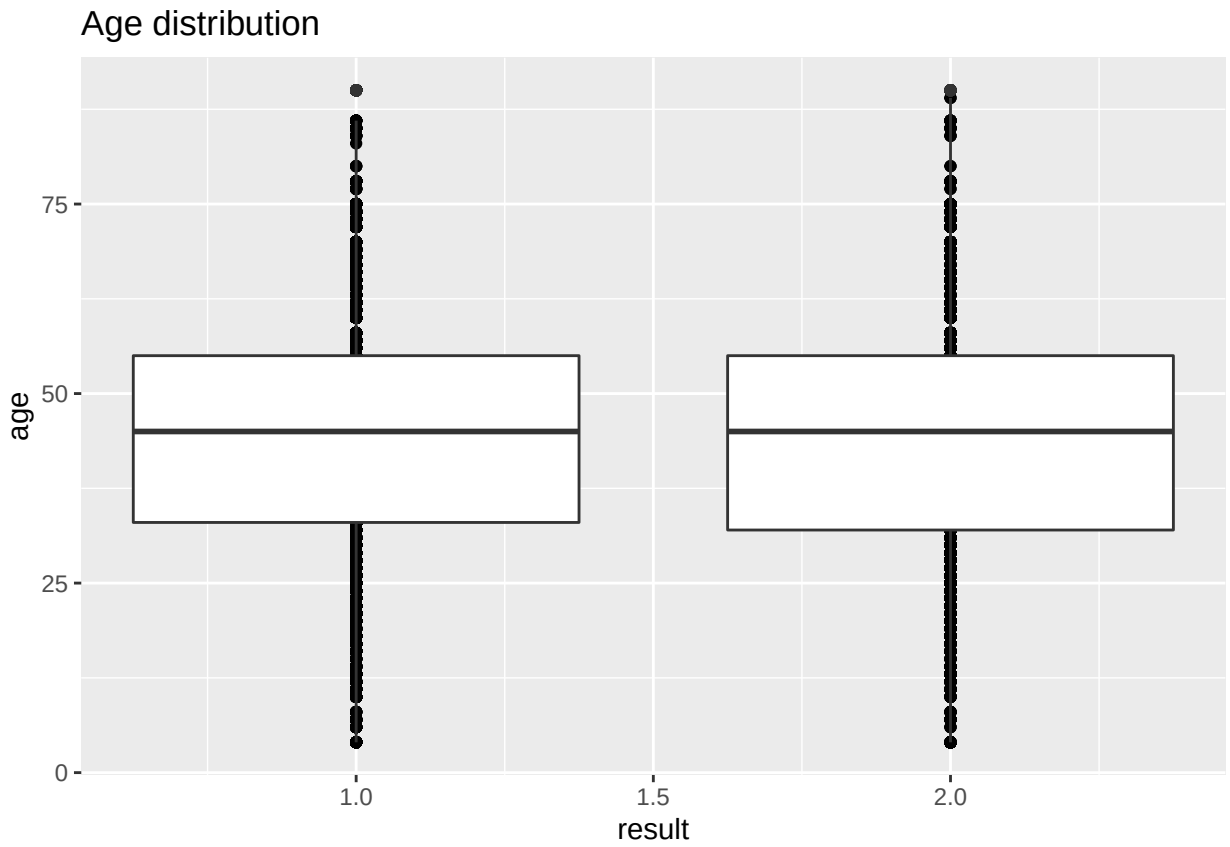
First of all we check if we have balanced data set. We plot numbers of patients with disease and without. From this figure we can see we have much bigger number of patients with disease than without. We could assume, that patients from the data set were sent for the blood test to diagnose the certain condition, not for general health assessment. That's why we have an unbalanced data set. In this situation it is very important to assess not just accuracy of our algorithm, also specificity and sensitivity.

```
#Plot numbers of patients with disease and without
liver_data %>% ggplot(aes(result)) + geom_bar() + ggtitle(" Number of patients with disease and without
```



Next we try to find if there are some relationships between age of patients and their age. From the plot we can see patients in the 1 and 2 result groups have almost the same age distribution.

```
# Plot age distribution in both groups
liver_data  %>% ggplot(aes(result, age, group = result)) + geom_point()+
  geom_boxplot() + ggtitle("Age distribution")
```



Then we check if there are some relationships between gender and results.From the table and plot we can not see that some gender are more tending to have a disease.

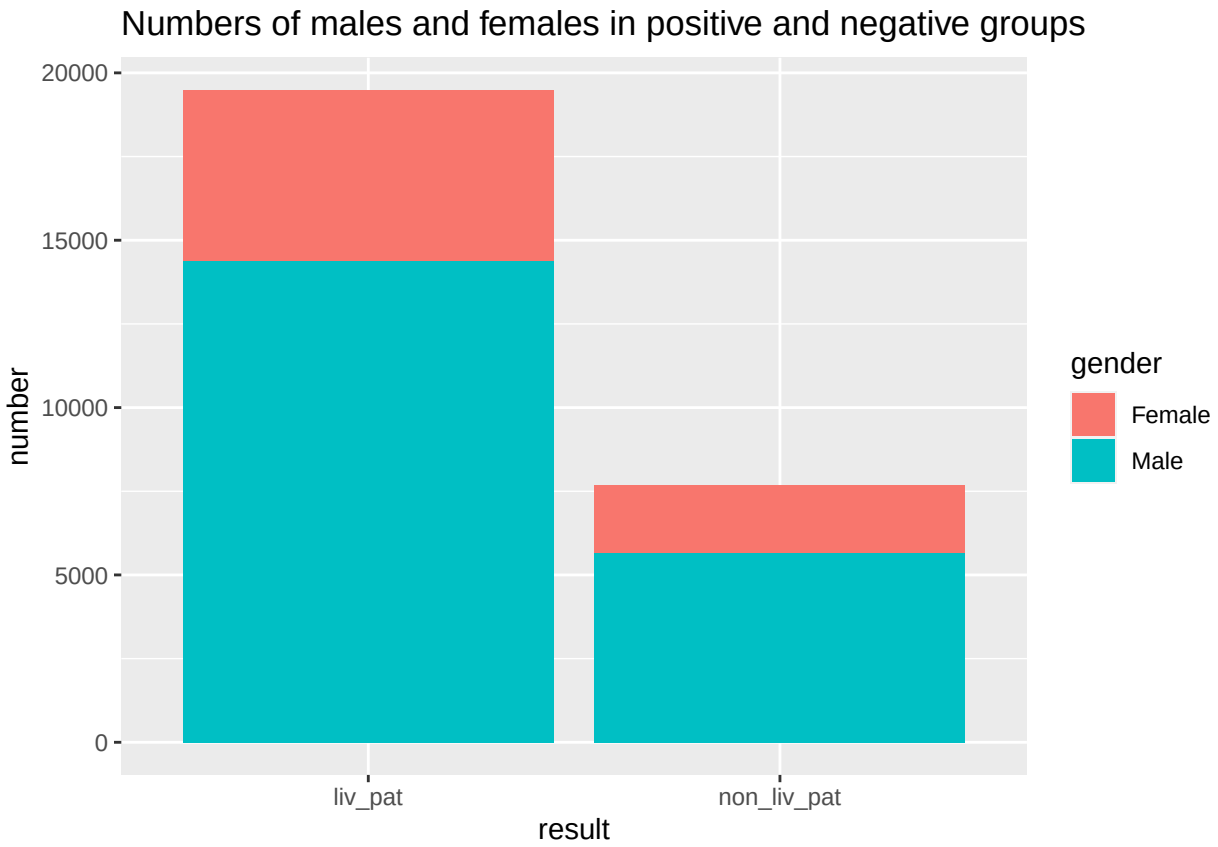
```
# Calculate numbers of males and females in both groups (with disease and without)
gender_data <- liver_data  %>% group_by(gender) %>% summarise(liv_pat = sum(result =="1"), non_liv_pat =
gender_data
```

```
## # A tibble: 2 x 3
##   gender liv_pat non_liv_pat
##   <chr>    <int>       <int>
## 1 Female    5104        2020
## 2 Male     14374        5660
```

```
# Make data tidy for visualization
plot_gender <- gender_data %>% gather(key = result,value = number, 2:3)
plot_gender
```

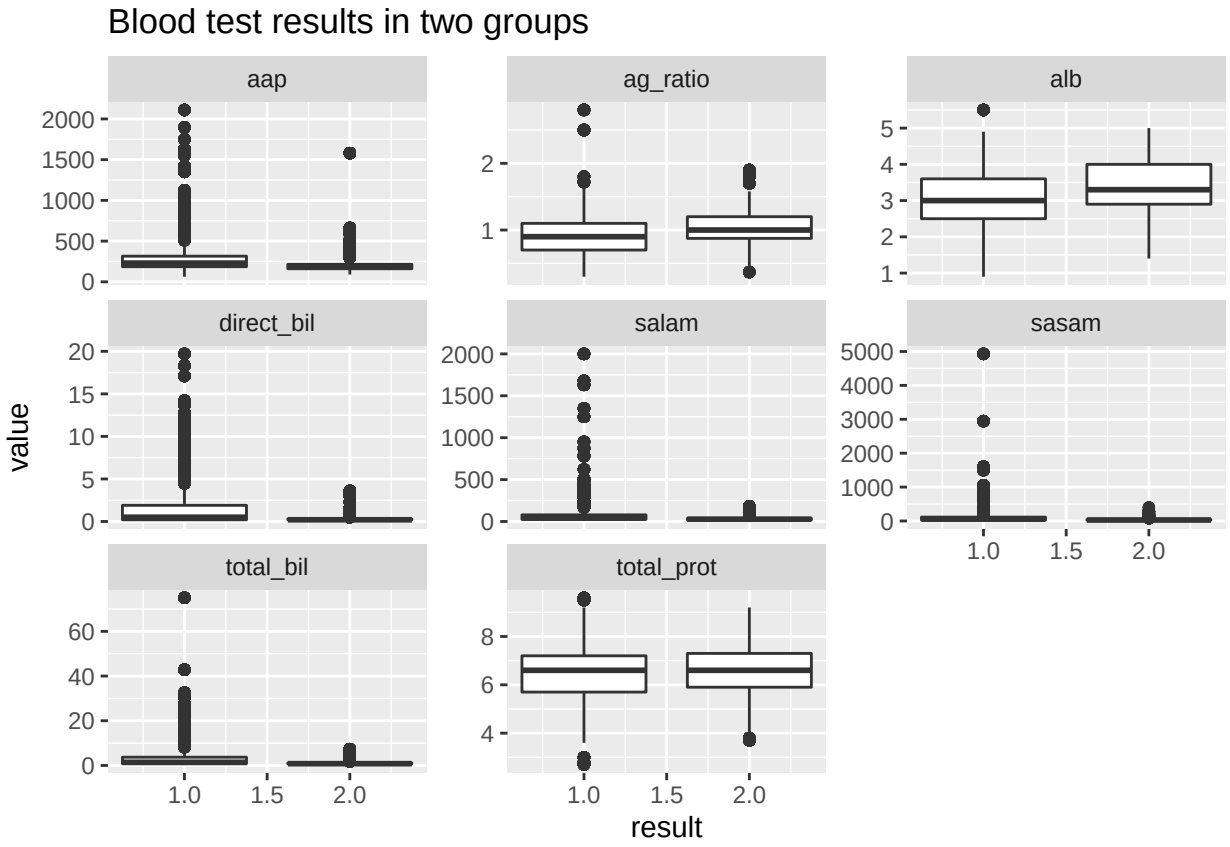
```
## # A tibble: 4 x 3
##   gender result      number
##   <chr>  <chr>        <int>
## 1 Female liv_pat       5104
## 2 Male   liv_pat      14374
## 3 Female non_liv_pat   2020
## 4 Male   non_liv_pat   5660
```

```
# Plot numbers of males and females in both groups
plot_gender %>% ggplot(aes(x = result, y = number)) + geom_col(aes(fill = gender))+
  ggtitle("Numbers of males and females in positive and negative groups")
```



After we plot all blood test results to see if we have some parameter which allows us to distinguish patients without machine learning algorithms. Also we can check if we have outliners or unusual results in different tests. Generally data looks good and results are in the possible range. But we can not say that any blood test has a strong correlation with the diagnosis.

```
# Plot 8 combined plots(boxplots) to visualize blood results
facet_liver <- gather(liver_data, key = "measure", value = "value",
                      c("total_bil", "direct_bil", "aap", "salam", "sasam",
                        "total_prot", "alb", "ag_ratio"))
ggplot(facet_liver, aes( x = result, y = value, group = result)) + geom_boxplot()+
  facet_wrap(~ measure, scales = "free_y") + ggtitle("Blood test results in two groups")
```



4. Data preparation

For data training and testing the final model we split our data into two sets - practice and final test set. Next the practice test we split for train and test set to try out different algorithms. Also we do not scale or normalize data, because using the original values as parameter values is more comprehensible(3) when you work with blood test results.

```
# Splitting data for final test set for testing the final model(10% of liver_data set)
# and practice set for training different algorithms
set.seed(4, sample.kind="Rounding") # if using R 3.6 or later
```

```
## Warning in set.seed(4, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
test_index <- createDataPartition(y = liver_data$result, times = 1, p = 0.1, list = FALSE)
practice_set <- liver_data[-test_index,]
final_test <- liver_data[test_index,]

# Create train and test set from practice set
set.seed(6, sample.kind="Rounding") # if using R 3.6 or later
```

```
## Warning in set.seed(6, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

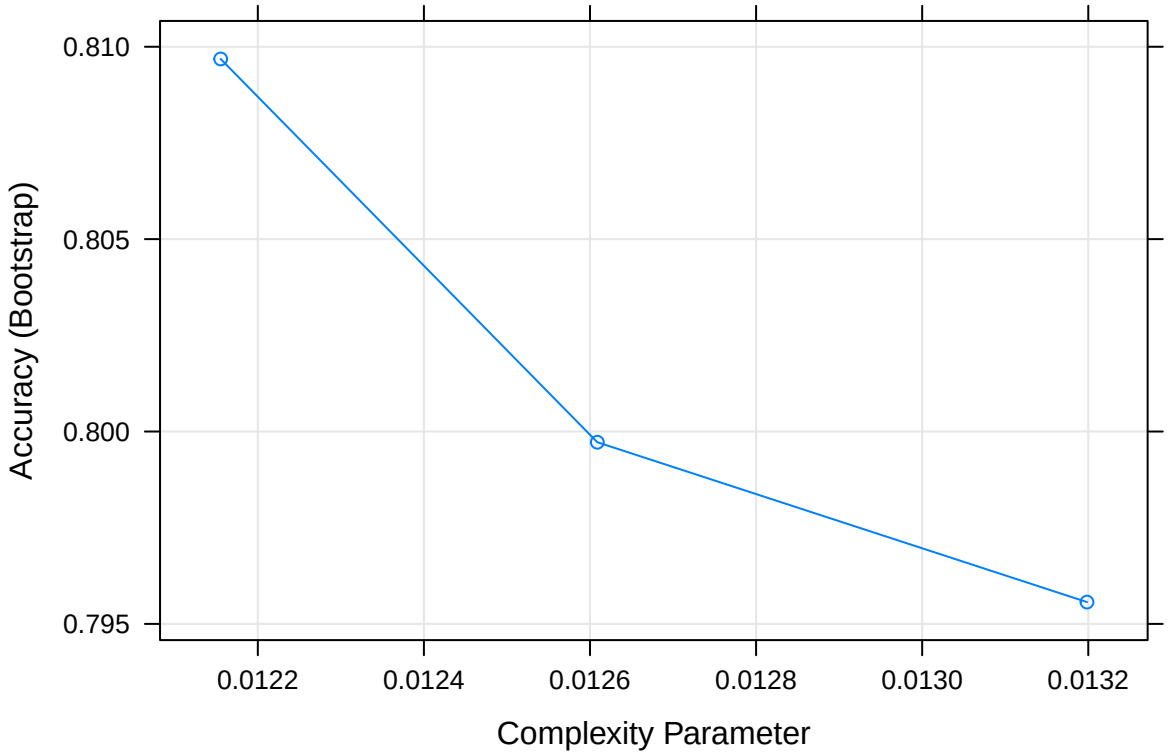
```
test_index <- createDataPartition(y = practice_set$result, times = 1,
                                  p = 0.2, list = FALSE)
train_set <- practice_set[-test_index,]
test_set <- practice_set[test_index,]
```

5.Methods and analysis.

5.1. Classification tree model

At first, we try classification tree model. From the table we can see that this model has low accuracy (0.835) and very low specificity(0.559)

```
# Classification tree model
train_rpart <- train(as.factor(result)~.,method = "rpart", data = train_set)
plot(train_rpart)
```



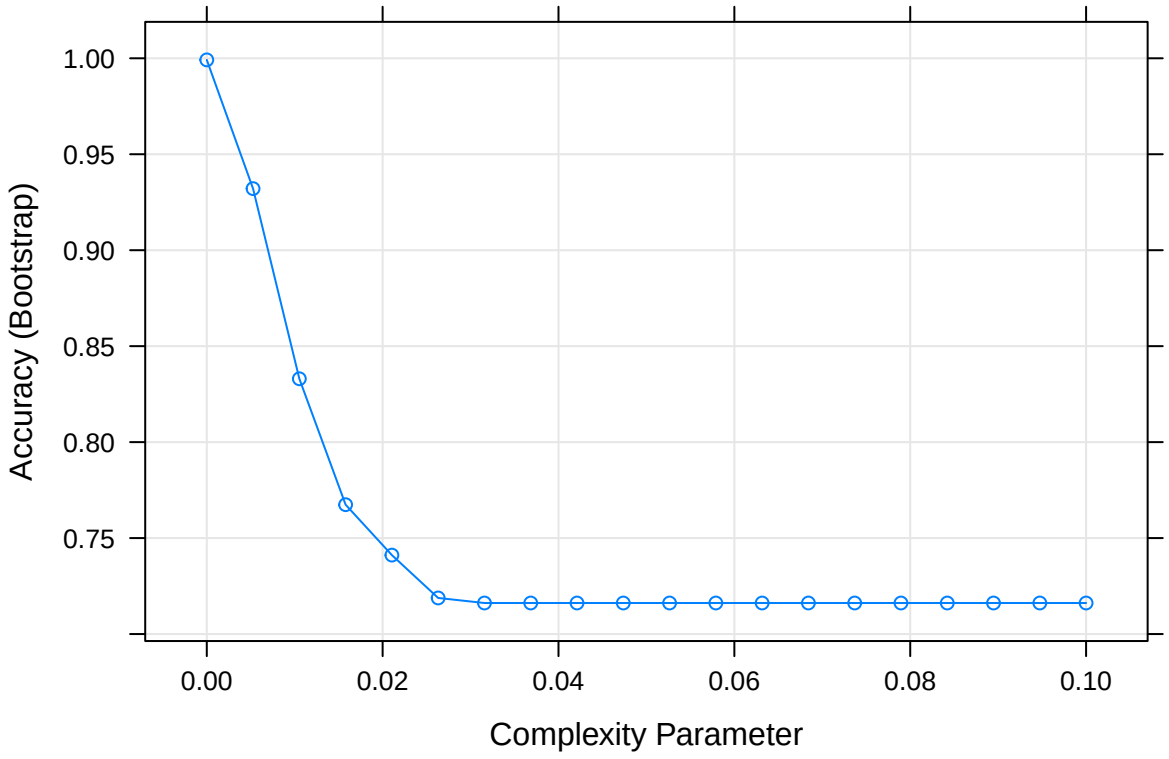

```
rpart_model <- confusionMatrix(predict(train_rpart, test_set),
                               as.factor(test_set$result))

#Make tibble with model results
model_result <- tibble(Model = "Classification tree",
                       Accuracy = rpart_model$overall["Accuracy"],
                       Sensitivity = rpart_model$byClass["Sensitivity"],
                       Specificity = rpart_model$byClass["Specificity"])
model_result
```

```
## # A tibble: 1 x 4
##   Model               Accuracy Sensitivity Specificity
##   <chr>                  <dbl>       <dbl>       <dbl>
## 1 Classification tree    0.835       0.946       0.559
```

Next we try to tune classification tree model. Now we get much better results. Our accuracy, sensitivity and specificity are close to 1. From the plot we can see that this result we get when complexity parameter is 0, which can lead us to over fitting.

```
# Classification tree model with tuning
train_rpart_t <- train(as.factor(result)~.,method = "rpart",
                       tuneGrid = data.frame(cp = seq(0.0, 0.1, len = 20)),
                       data = train_set)
plot(train_rpart_t)
```




```
tuned_rpart_model <- confusionMatrix(predict(train_rpart_t, test_set), as.factor(test_set$result))

#Adding to tibble tuned classification tree model results

model_result <- bind_rows(model_result,tibble(Model = "Classification tree with tuning",
                       Accuracy = tuned_rpart_model$overall["Accuracy"],
                       Sensitivity = tuned_rpart_model$byClass["Sensitivity"],
                       Specificity = tuned_rpart_model$byClass["Specificity"]))
model_result
```

```
## # A tibble: 2 x 4
##   Model                          Accuracy Sensitivity Specificity
##   <chr>                             <dbl>       <dbl>       <dbl>
## 1 Classification tree               0.835       0.946       0.559
## 2 Classification tree with tuning   0.999       0.999       1
```

5.2. Generalized linear model

Our next attempt is to try generalized linear model. It does not give a good result. This model gives accuracy = 0.725

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## # A tibble: 3 x 4
##   Model                           Accuracy Sensitivity Specificity
```

```
##   <chr>                              <dbl>       <dbl>       <dbl>
## 1 Classification tree                0.835       0.946       0.559
## 2 Classification tree with tuning    0.999       0.999       1
## 3 Generalized linear model           0.725       0.938       0.192
```

5.3. K-nearest neighbors model

After we try k-nearest neighbors model with default parameters and with trainControl. Both model gives us almost the same results: accuracy = 0.96, sensitivity = 0.97 and specificity = 0.94. These results are better than glm model and classification tree model with default parameters.

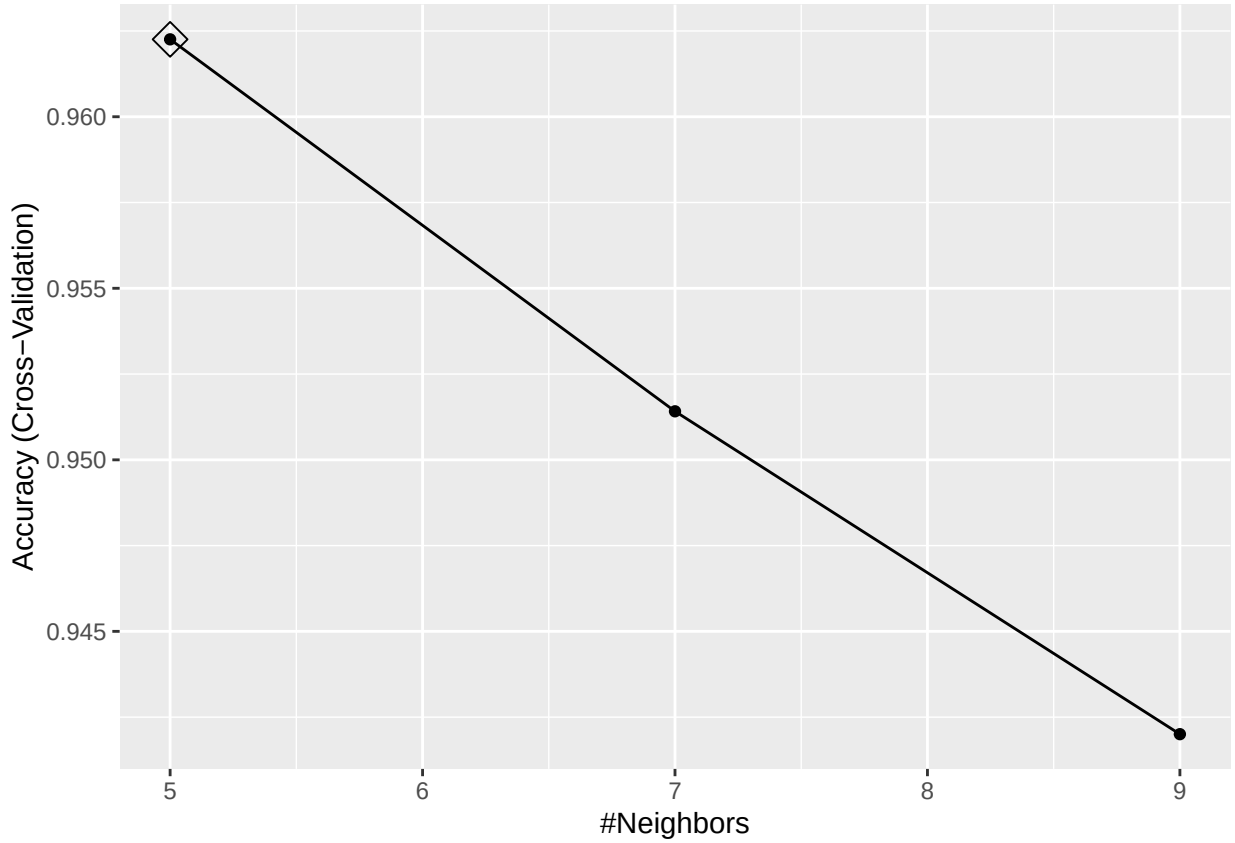
```
#k-nearest neighbors model
train_knn <- train(as.factor(result)~.,method = "knn", data = train_set)
knn_model <- confusionMatrix(predict(train_knn, test_set), as.factor(test_set$result))

#Adding to tibble k-nearest neighbors model results

model_result <- bind_rows(model_result,tibble(Model = "k-nearest neighbors model",
                                              Accuracy = knn_model$overall["Accuracy"],
                                              Sensitivity = knn_model$byClass["Sensitivity"],
                                              Specificity = knn_model$byClass["Specificity"]))
model_result
```

```
## # A tibble: 4 x 4
##   Model                            Accuracy Sensitivity Specificity
##   <chr>                               <dbl>       <dbl>       <dbl>
## 1 Classification tree                 0.835       0.946       0.559
## 2 Classification tree with tuning     0.999       0.999       1
## 3 Generalized linear model            0.725       0.938       0.192
## 4 k-nearest neighbors model           0.964       0.971       0.947
```

```
#k-nearest neighbors model with trainControl
control <- trainControl(method = "cv", number = 10, p = 0.9)
train_knn_cv <- train(as.factor(result)~., method = "knn", data = train_set,
                      trControl = control)
ggplot(train_knn_cv, highlight = TRUE)
```




```
knn_cv_model <- confusionMatrix(predict(train_knn_cv, test_set), as.factor(test_set$result))

#Adding to tibble k-nearest neighbors model with trainControl results

model_result <- bind_rows(model_result,tibble(Model = "k-nearest neighbors model with trainControl",
                                              Accuracy = knn_cv_model$overall["Accuracy"],
                                              Sensitivity = knn_cv_model$byClass["Sensitivity"],
                                              Specificity = knn_cv_model$byClass["Specificity"]))
model_result
```

```
## # A tibble: 5 x 4
##   Model                                     Accuracy Sensitivity Specificity
##   <chr>                                        <dbl>       <dbl>       <dbl>
## 1 Classification tree                          0.835       0.946       0.559
## 2 Classification tree with tuning              0.999       0.999       1
## 3 Generalized linear model                     0.725       0.938       0.192
## 4 k-nearest neighbors model                    0.964       0.971       0.947
## 5 k-nearest neighbors model with trainControl  0.965       0.972       0.947
```

5.4. Random forest model

As we can see from the table the random forest model gives us the best results. We get Accuracy, Sensitivity and Specificity = 1.

```
#Random forest model
set.seed(6, sample.kind = "Rounding")
```

```
## Warning in set.seed(6, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
```

```
## used
```

```
train_rf <- train(as.factor(result)~., method = "rf", data = train_set)
rf_model <- confusionMatrix(predict(train_rf, test_set), as.factor(test_set$result))

#Adding to tibble random forest model results

model_result <- bind_rows(model_result,tibble(Model = "Random forest",
                                              Accuracy = rf_model$overall["Accuracy"],
                                              Sensitivity = rf_model$byClass["Sensitivity"],
                                              Specificity = rf_model$byClass["Specificity"]))
model_result
```

```
## # A tibble: 6 x 4
##   Model                                     Accuracy Sensitivity Specificity
##   <chr>                                        <dbl>       <dbl>       <dbl>
## 1 Classification tree                          0.835       0.946       0.559
## 2 Classification tree with tuning              0.999       0.999       1
## 3 Generalized linear model                     0.725       0.938       0.192
## 4 k-nearest neighbors model                    0.964       0.971       0.947
## 5 k-nearest neighbors model with trainControl  0.965       0.972       0.947
## 6 Random forest                                1           1           1
```

6.Results. Random forest model on the final test.

On the final test random forest model gives accuracy, sensitivity, and specificity almost = 1, which is a very good performance.

```
#Trying random forest model on the final test set
final_results <- confusionMatrix(predict(train_rf, final_test), as.factor(final_test$result))

#Adding to tibble final test results

model_result <- bind_rows(model_result,tibble(Model = "Final test",
                                              Accuracy = final_results$overall["Accuracy"],
                                              Sensitivity = final_results$byClass["Sensitivity"],
                                              Specificity = final_results$byClass["Specificity"]))
model_result
```

```
## # A tibble: 7 x 4
##   Model                                     Accuracy Sensitivity Specificity
##   <chr>                                        <dbl>       <dbl>       <dbl>
## 1 Classification tree                          0.835       0.946       0.559
## 2 Classification tree with tuning              0.999       0.999       1
## 3 Generalized linear model                     0.725       0.938       0.192
## 4 k-nearest neighbors model                    0.964       0.971       0.947
## 5 k-nearest neighbors model with trainControl  0.965       0.972       0.947
## 6 Random forest                                1           1           1
## 7 Final test                                   1.00        1           0.999
```

7. Conclusion

Random forest model predicts if the patient has the liver disease with almost 100% accuracy, specificity and sensitivity. All other tried models(classification tree, generalized linear model, k- nearest neighbors) did not give us the same good result. Classification tree with complexity parameter = 0 gives us almost the same results as random forest model on the training test set, but using this value of complexity parameter could

cause overfitting and model could not work good on the new data.
At this project we trained our algorithm on the data set with no missing values, but in the real life it is not the case. The future work it is to create an model, that will be able to work with missing values. Also we have to think with what value we can replace NA(get a consultation with doctors?). Random forest model could perform good in the data set with missing values.

References

1.https://www.kaggle.com/datasets/abhi8923shriv/liver-disease-patient-dataset/discussion/201685

2.https://github.com/VictoriaRomano/LiverDisease/blob/main/liver_patient.xls

3. Gregor Guncar, Matjaz kukar,Matejia Notar, Miran Bvar, Peter Cernelc, Manca Notar & Marco Notar.An application of machine learning to haematological diagnosis, Scientific Reports 8, Article number:411(2018)