Liver Disease Machine Learning Capstone Project

Alexandra Cirstoc

1/7/2021

Contents

Introduction

Patients with Liver disease have been continuously increasing because of excessive consumption of alcohol, inhale of harmful gases, intake of contaminated food, pickles and drugs.

The data set ILPD (Indian Liver Patient Dataset) was collected from north east of Andhra Pradesh, India.

This report aims at using various classification machine learning techniques to predict if a patient has a liver disease or not.

Data exploration

Before doing any data exploration and pre-processing, the original liver dataset will be split into a **validation** set and a **liver subset**, in order to mimic real-life machine learning techniques, where the dataset to be predicted is generally unknown. Therefor the "validation" dataset will be used as the final test set, on which the best performing algorithm will be used.

The "liver subset" will then be further split into train and test sets, on which several machine learning methods will be applied.

In both steps the datasets will be split into 20% for testing and 80% for training. The choice for using 20% and not less is due to the relatively small size of the overall dataset, which enables the test dataset to be large enough to perform validation of the different models.

```
set.seed(1, sample.kind="Rounding")

test_index   <- createDataPartition(y = liver$Disease, times = 1, p = 0.2, list = FALSE)
liver_subset <- liver[-test_index,]
validation   <- liver[test_index,]

rm(test_index)
```

Check if the validation and liver_subset datasets have similar proportions of liver disease:

```
## [1] 0.7139785
```

```
## [1] 0.7094017
```

it looks like the 2 datasets are balanced.

From now on all data exploration and pre-processing will be done on liver_subset.

Check if there are any null values in the dataset and if yes, drop those rows.

	Age	Gender	totalBilirubin	directBilirubin	totalProteins	Albumin	AGratio	SGPT	SGOT	Alkphos	Disease
209	45	Female	0.9	0.3	189	23	33	6.6	3.9	NA	yes
253	35	Female	0.6	0.2	180	12	15	5.2	2.7	NA	no
312	27	Male	1.3	0.6	106	25	54	8.5	4.8	NA	no

Check if there are duplicate rows in the dataset:

	Age	Gender	totalBilirubin	directBilirubin	totalProteins	Albumin	AGratio	SGPT	SGOT	Alkphos	Disease
19	40	Female	0.9	0.3	293	232	245	6.8	3.1	0.8	yes
26	34	Male	4.1	2.0	289	875	731	5.0	2.7	1.1	yes
62	58	Male	1.0	0.5	158	37	43	7.2	3.6	1.0	yes
108	36	Male	0.8	0.2	158	29	39	6.0	2.2	0.5	no
143	30	Male	1.6	0.4	332	84	139	5.6	2.7	0.9	yes
201	49	Male	0.6	0.1	218	50	53	5.0	2.4	0.9	yes

There seem to be 6 duplicate rows in the dataset, which might cause noise later on in the analysis. The duplicates will be removed from the dataset.

A total of 9 rows have been removed from the training dataset **liver_subset**.

Dataset information

The overview of the dataset:

```
## 'data.frame':    456 obs. of  11 variables:
##  $ Age            : int  62 62 58 72 26 29 17 55 57 74 ...
##  $ Gender         : chr  "Male" "Male" "Male" "Male" ...
##  $ totalBilirubin : num  10.9 7.3 1 3.9 0.9 0.9 0.9 0.7 0.6 1.1 ...
##  $ directBilirubin: num  5.5 4.1 0.4 2 0.2 0.3 0.3 0.2 0.1 0.4 ...
##  $ totalProteins  : int  699 490 182 195 154 202 202 290 210 214 ...
##  $ Albumin        : int  64 60 14 27 16 14 22 53 51 22 ...
##  $ AGratio        : int  100 68 20 59 12 11 19 58 59 30 ...
##  $ SGPT           : num  7.5 7 6.8 7.3 7 6.7 7.4 6.8 5.9 8.1 ...
##  $ SGOT           : num  3.2 3.3 3.4 2.4 3.5 3.6 4.1 3.4 2.7 4.1 ...
##  $ Alkphos        : num  0.74 0.89 1 0.4 1 1.1 1.2 1 0.8 1 ...
##  $ Disease        : chr  "yes" "yes" "yes" "yes" ...
##  - attr(*, "na.action")= 'omit' Named int [1:3] 173 204 252
##   ..- attr(*, "names")= chr [1:3] "209" "253" "312"
```

The dataset has 456 observations and one class **Disease**, used for dividing the data into 2 groups (patients with a liver disease or not).

The dataset also contains 10 features (or predictor variables): age, gender, total Bilirubin, direct Bilirubin, total proteins, albumin, A/G ratio, SGPT, SGOT and Alkphos. The predictor variables will be used to determine if a patient has a liver disease or not.

Any patient whose age exceeded 89 is listed as being of age "90" (as per the creators of the dataset).

Attribute Information:

1. Age: Age of the patient
2. Gender: Gender of the patient
3. totalBilirubin: Total Bilirubin
4. directBilirubin: Direct Bilirubin
5. Alkphos: Alkaline Phosphotase
6. Sgpt: Alamine Aminotransferase
7. Sgot: Aspartate Aminotransferase
8. totalProteins: Total Proteins
9. Albumin: Albumin
10. AGratio: Albumin and Globulin Ratio
11. Disease: Selector field used to split the data into two sets (labeled by the experts)

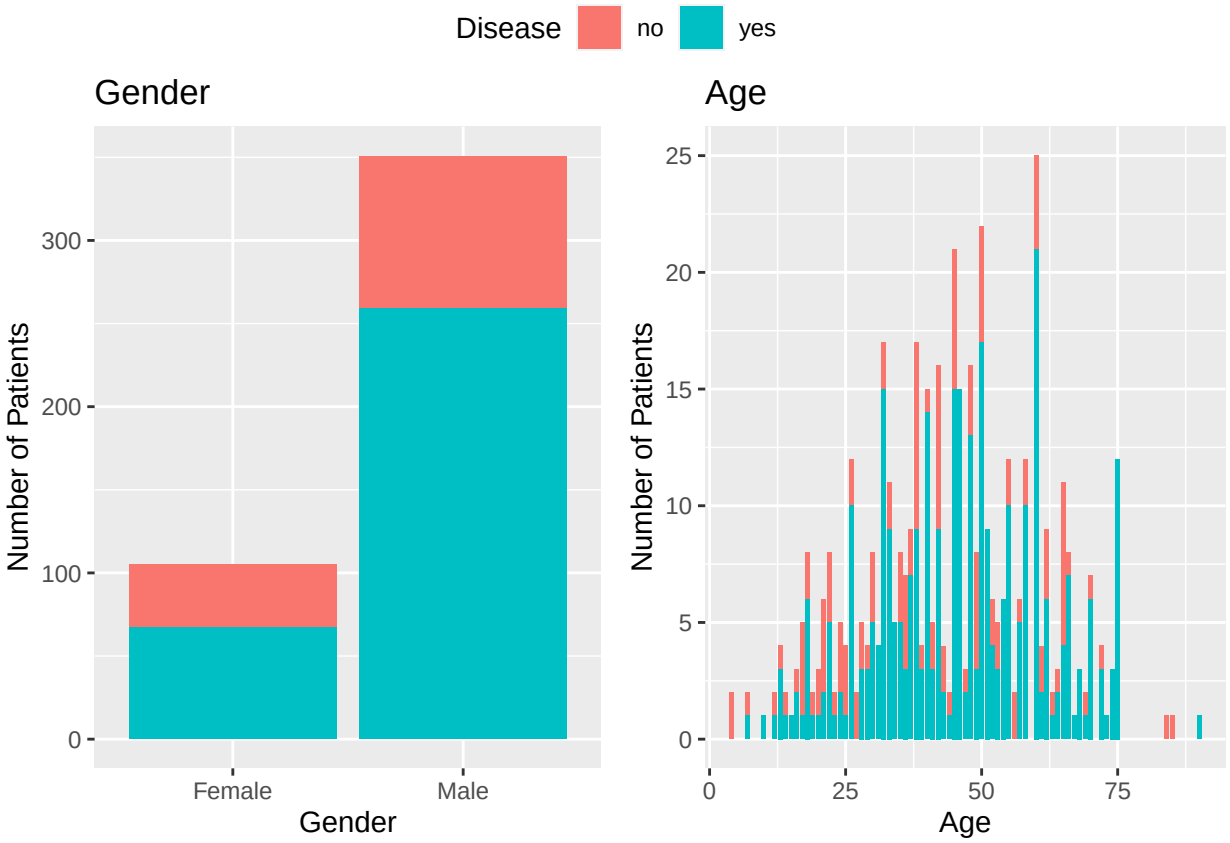
I have found an article which mentions: "Nonalcoholic fatty liver disease (NAFLD) is emerging as an important cause of liver disease in India. Epidemiological studies suggest prevalence of NAFLD in around 9% to 32% of general population in India with higher prevalence in those with overweight or obesity and those with diabetes or prediabetes."

This dataset contains (71%) of patients with a liver disease, which implies there is a class imbalance in the dataset.

Machine learning models on a dataset with high prevalence of disease might over-predict disease on the general population. This will be discussed in more detail in the Data preprocessing and Modeling sections.

Analyses of the predictor variables

First let's have a look into the gender and age variables:



The gender graph reveals that the dataset contains many more males than females but having a liver disease seems to be prevalent for both genders. However males seem a bit more prevalent to have a liver disease than females (73.8% vs. 63.8%), but the difference is not significant.

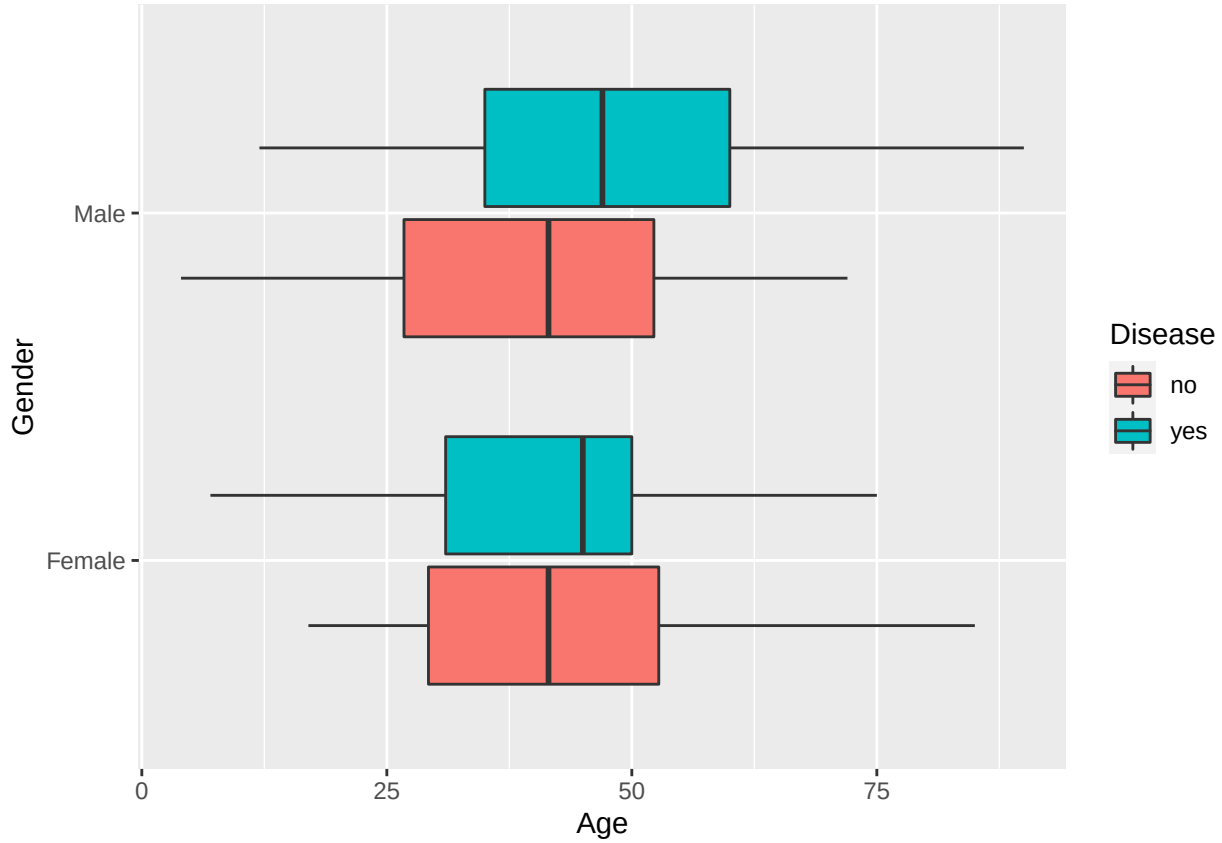
Gender	Disease	count	share_per_gender
Female	no	38	36.19048
Female	yes	67	63.80952
Male	no	92	26.21083
Male	yes	259	73.78917

Looking into age a bit closer reveals that having a liver disease happens at an average age of around 46, but the full range of liver patients goes from age 7 to age 90 in this dataset.

Disease	avg	min	max	sd
no	40.86923	4	85	17.02182
yes	45.88344	7	90	15.55136

The distribution of liver disease over age:

Also viewed as a boxplot:

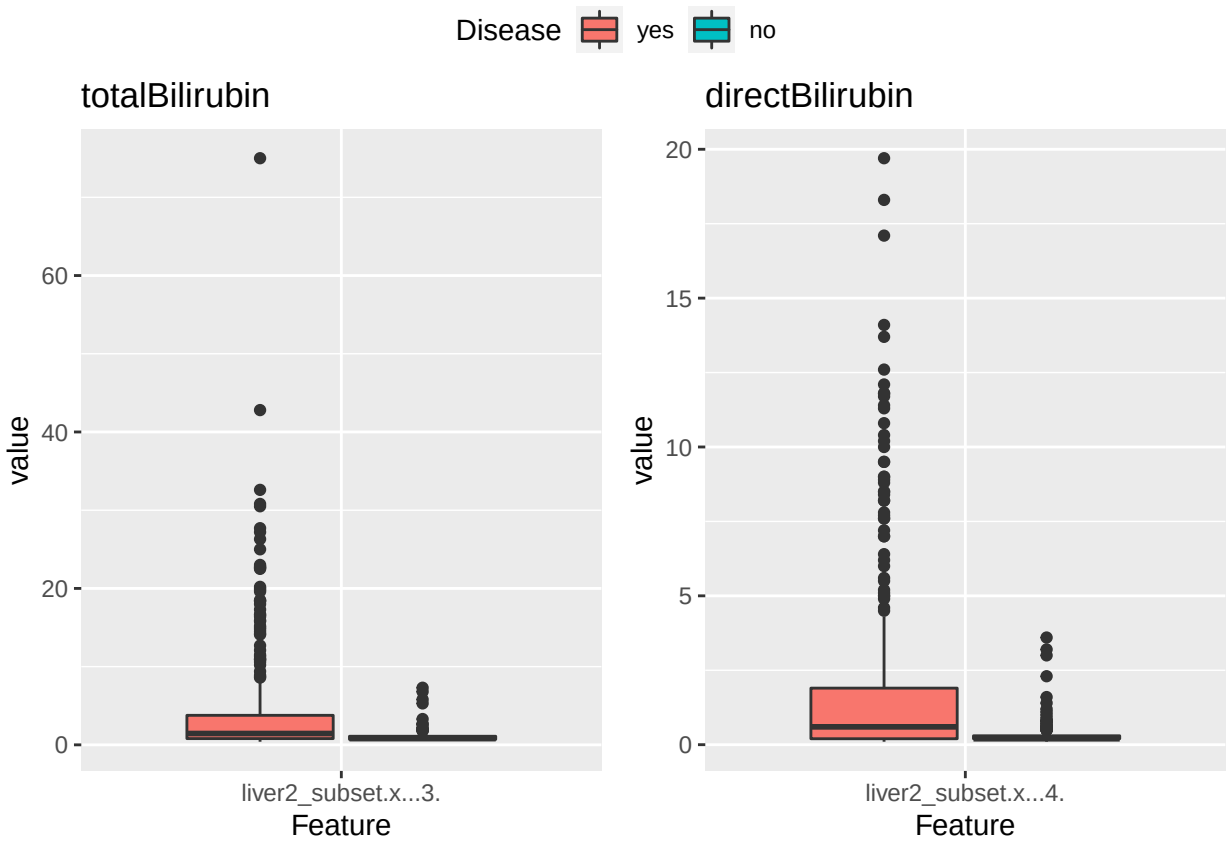


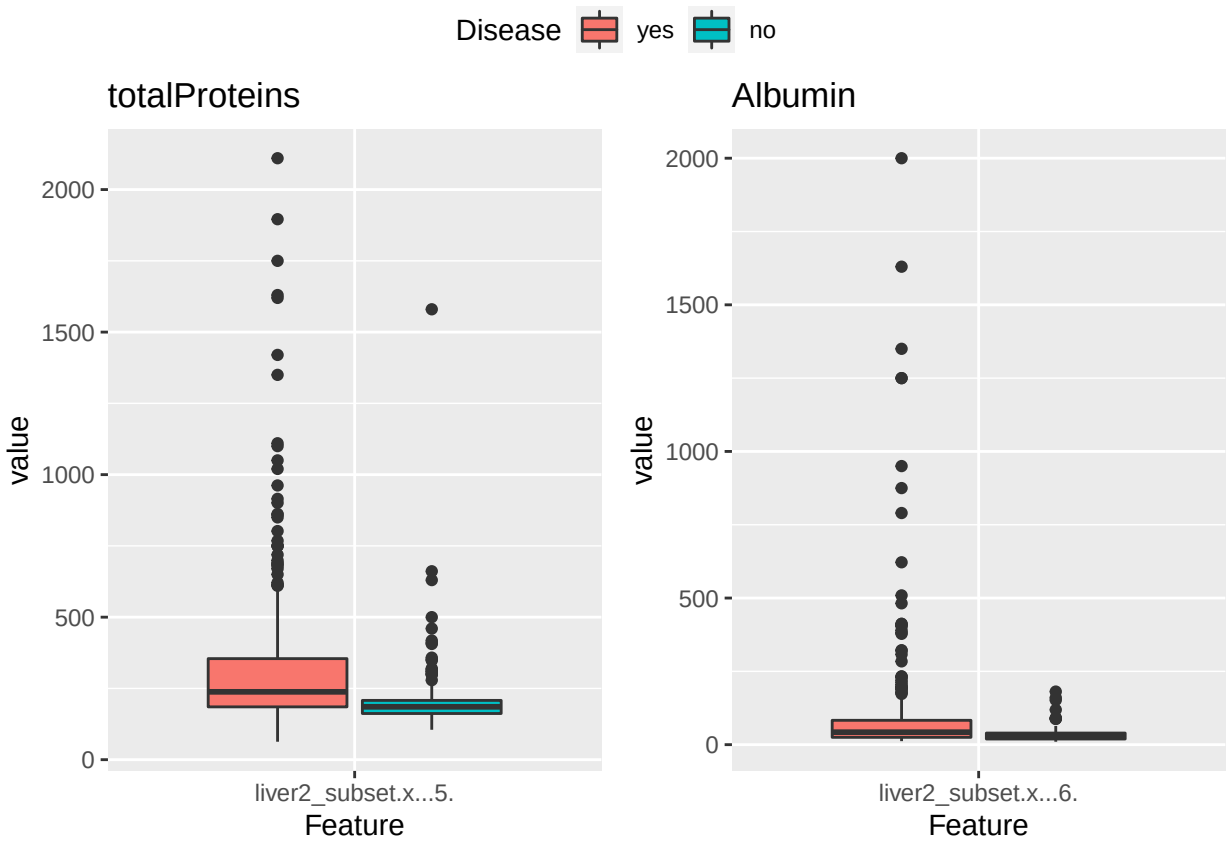
Based on the boxplot the age distribution of male diseased patients seems to be skewed more to the right than females (i.e. males that are a bit older than females seem to be getting a liver disease). The age plots also show us that relatively young people get a liver disease (somewhere between age 35 and 60), which could imply that having a liver disease is not an "old age" problem (at least in this small data sample).

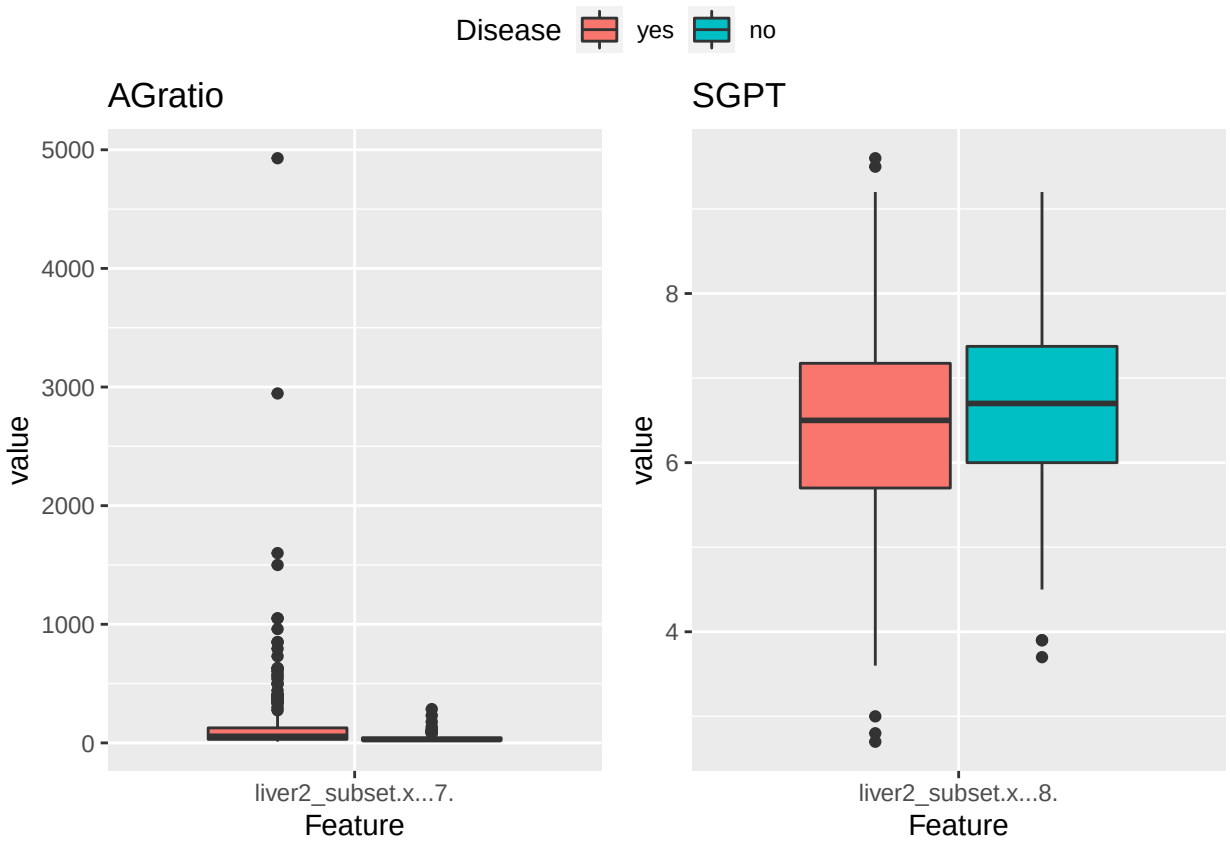
Now let's have an overall look at the predictor variables.

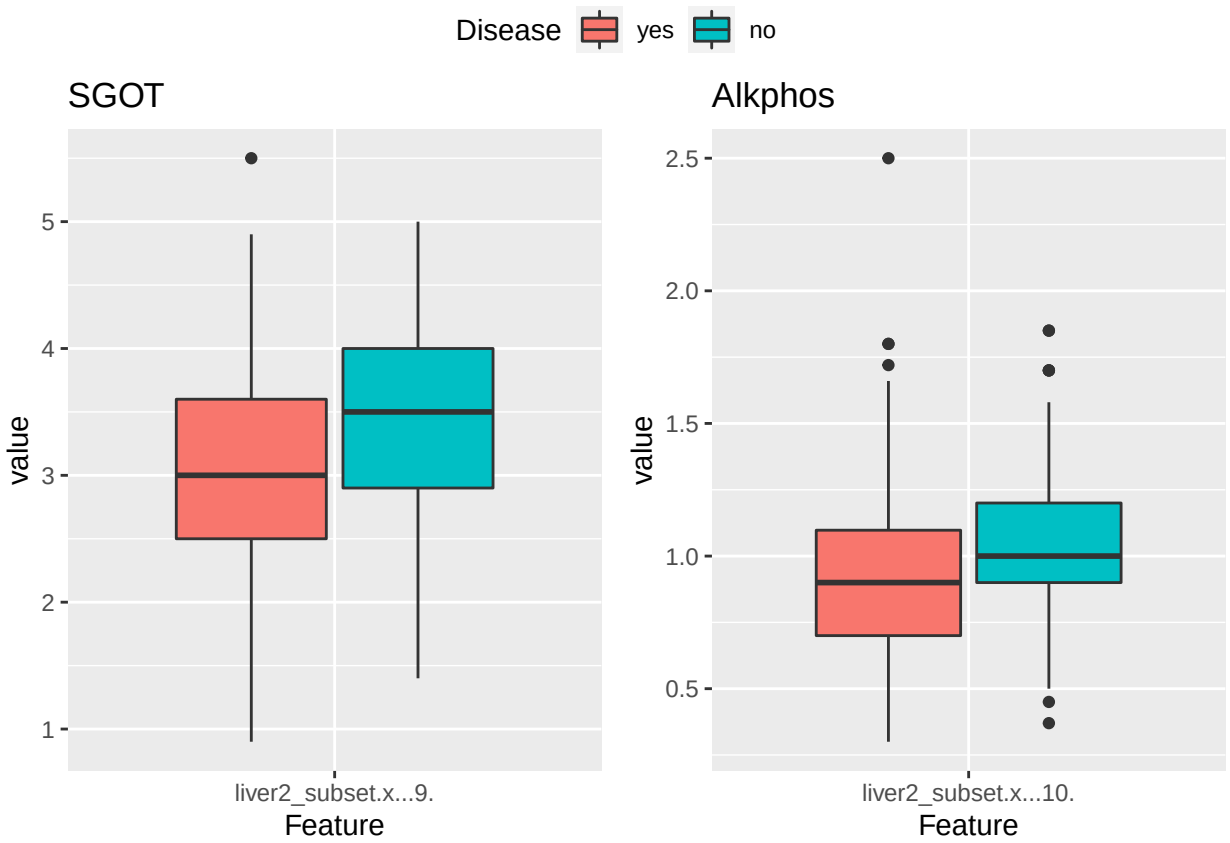
Computing box plots for continuous variables and jitter plots for categorical values:


Disease
yes
no
Age
75
50
25
0
value
liver2_subset.x...1.
Feature
Gender
yes
no
Disease
Female
Male
Gender


Disease
yes
no
totalBilirubin
directBilirubin
value
60
40
20
0
20
15
10
5
0
liver2_subset.x...3.
liver2_subset.x...4.
Feature
Feature


Disease
yes
no
totalProteins
Albumin
value
2000
1500
1000
500
0
liver2_subset.x...5.
liver2_subset.x...6.
Feature
Feature


Disease
yes
no
AGratio
SGPT
value
5000
4000
3000
2000
1000
0
8
6
4
liver2_subset.x...7.
liver2_subset.x...8.
Feature
Feature



Some initial conclusions can be drawn from these plots: Age, Gender, SGPT, SGOT and Alkphos seem to be relatively weak predictors, because the box plots/jitter plot do not seem to indicate a clear presence/no presence of a liver disease.

I will next perform several data pre-processing steps, which hopefully will shed more light into each predictor.

Data pre-processing

In machine learning, we often transform predictors before running machine learning algorithms. We can also remove predictors if they are clearly not useful. Examples of pre-processing include standardizing the predictors, taking the log transform of some predictors, removing predictors that are highly correlated with others, and removing predictors with very few non-unique values or close to zero variation.

Dataset resampling

Due to the imbalanced nature of this dataset, I will attempt to oversample my overall training dataset. Inspiration for performing this step is taken from an article about imbalanced datasets, which I found quite useful and interesting.

Oversampling is generally done on small datasets and is the process of adding instances from the under-represented class to the training dataset.

```
set.seed(1, sample.kind="Rounding")

tmp <- liver_subset %>%
```

```
  mutate(Disease = as.numeric(Disease == "yes")) %>%
  mutate(Gender = as.numeric(Gender == "Male"))

liver_subset_sampled <- oversample(
  tmp,
  ratio = 0.7,
  method = "SMOTE",
  classAttr = "Disease"
) %>%
  mutate(Gender = round(Gender))
```

The new dataset now contains 555 rows with 59% of patients with a liver disease, which does result in a more balanced dataset. This is what will be used from now on in the analysis.

Data normalization

As a second step in data processing, I will normalize the data. The motivation for doing this step is that the features of the dataset have quite varying ranges (as we can see in the dataset overview). By normalizing the variables, we can make sure that we have comparable values in this analysis.

This step is also important for producing more meaningful correlation matrix, principal component analysis and calculation of distances.

In several iterations of standardization steps, I have applied the R "scale" function (z score normalization) and the Min-Max normalization.

I have decided in the end to use the scale function, because this one produced the best predictions (even if only marginally so).

```
# Normalizing data with the scale function

liver2_subset_scale <- liver2_subset
liver2_subset_scale$x <- scale(liver2_subset$x)


paste0("Mean: ",mean(liver2_subset_scale$x))
```

```
## [1] "Mean: -4.62775788367884e-17"
```

```
paste0("Standard deviation: ",sd(liver2_subset_scale$x))
```

```
## [1] "Standard deviation: 0.999188713978319"
```

```
head(liver2_subset_scale$x)
```

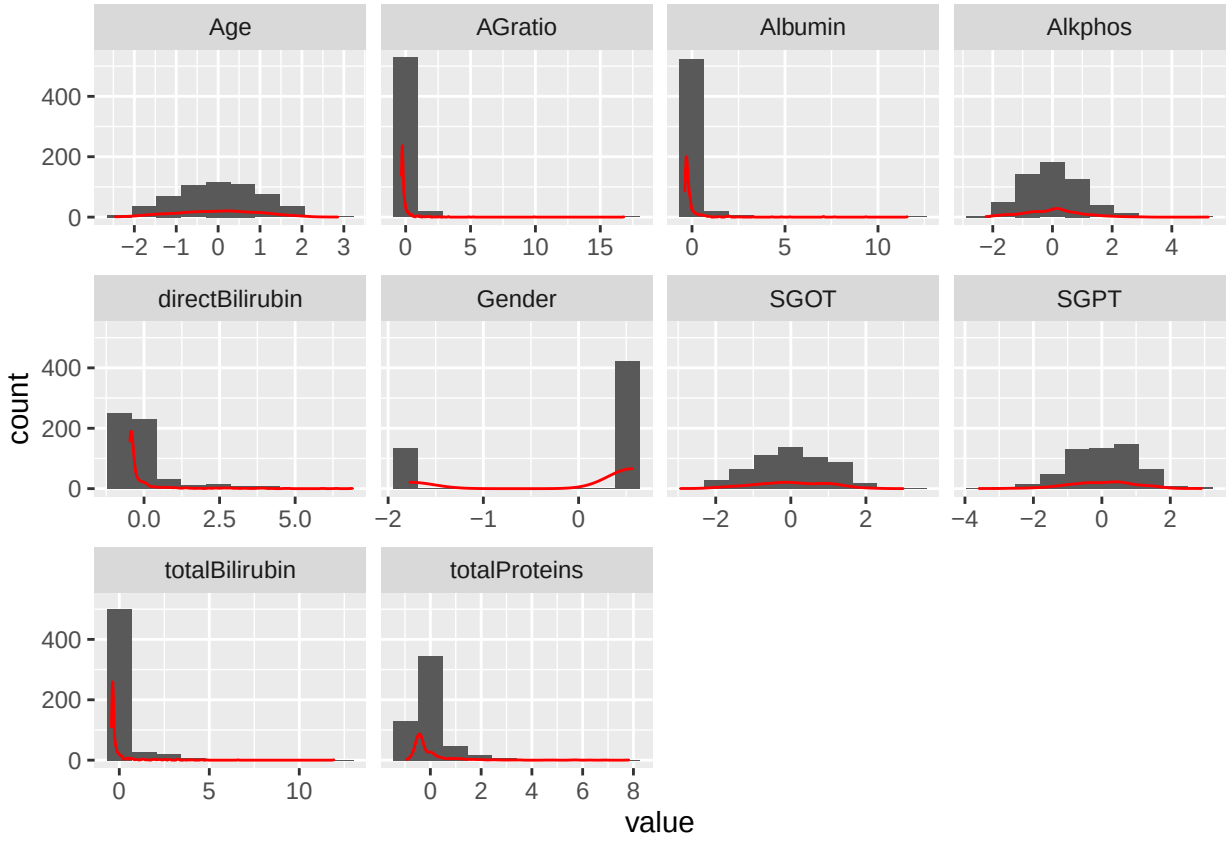
```
##             Age     Gender totalBilirubin directBilirubin totalProteins
## [1,]  1.1287771  0.5636633      1.3016054       1.5603297     1.7784033
## [2,]  1.1287771  0.5636633      0.7054634       1.0349032     0.8841675
## [3,]  0.8819648  0.5636633     -0.3377850      -0.3537239    -0.4336537
## [4,]  1.7458078  0.5636633      0.1424404       0.2467635    -0.3780314
## [5,] -1.0925335 -1.7709122     -0.3543445      -0.4287848    -0.5534556
## [6,] -0.9074243 -1.7709122     -0.3543445      -0.3912544    -0.3480809
```

```
##          Albumin      AGratio      SGPT        SGOT    Alkphos
## [1,] -0.05642494 -0.009558839 0.9328116  0.02653578 -0.7393987
## [2,] -0.08045398 -0.121026320 0.4621016  0.15524901 -0.2314651
## [3,] -0.35678791 -0.288227543 0.2738176  0.28396225  0.1410196
## [4,] -0.27869354 -0.152376550 0.7445276 -1.00317009 -1.8907149
## [5,] -0.34477339 -0.316094413 0.4621016  0.41267548  0.1410196
## [6,] -0.35678791 -0.319577772 0.1796756  0.54138872  0.4796420
```

Checking for skewness in data

Motivation: Skewed data is data that has a distribution that is pushed to one side or the other (larger or smaller values) rather than being normally distributed. Some machine learning methods assume normally distributed data and can perform better if the skew is removed. I have used this article as an inspiration for this subsection.

An overview of the distributions of all predictors:



And their skewness coefficients:

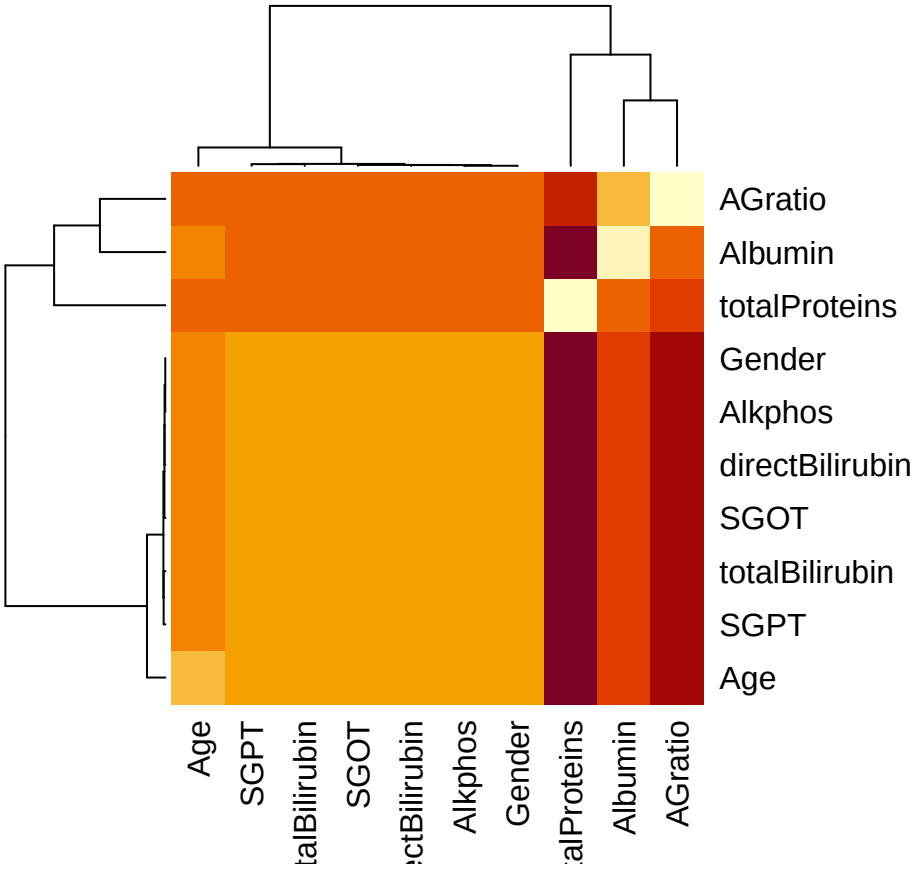
```
##            Age         Gender  totalBilirubin directBilirubin   totalProteins
##    -0.03070886    -1.20833805     5.36828485     3.50202426     3.93386310
##        Albumin        AGratio           SGPT           SGOT         Alkphos
##     7.21779848    11.17887273    -0.30586442    -0.15137443     0.42911421
```

If skewness value lies above +1 or below -1, data is highly skewed. If it lies between +0.5 to -0.5, it is moderately skewed. If the value is 0, then the data is symmetric (normal distributed). Although some of

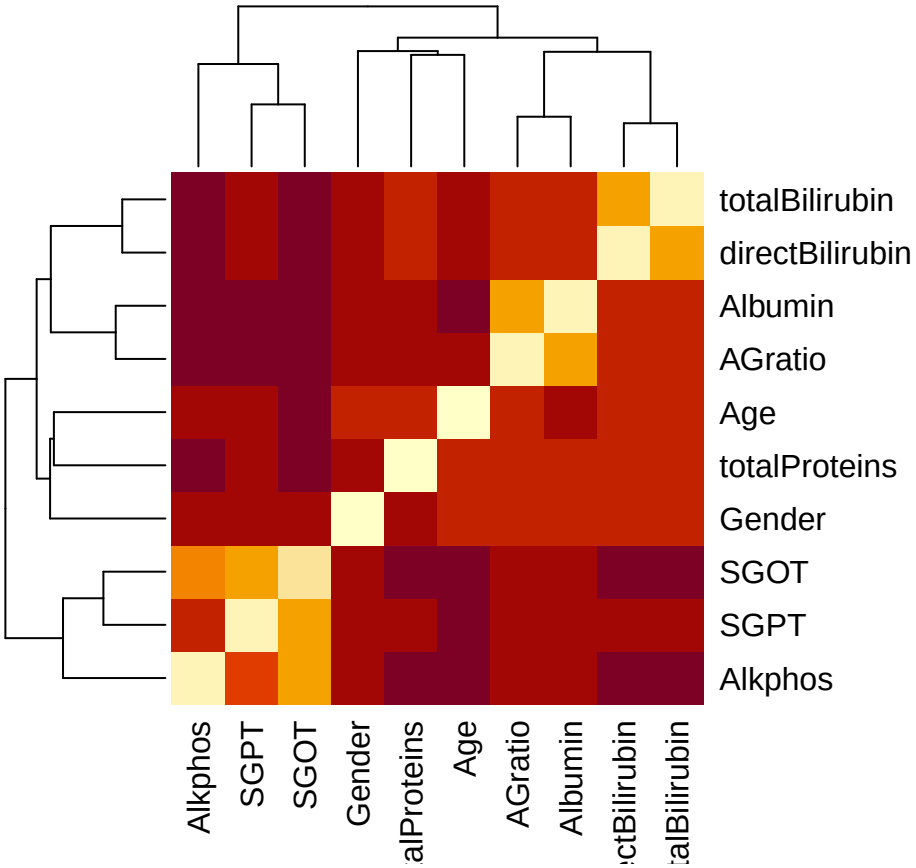
the predictors are indeed skewed, further data transformations (log, sqrt etc.) for removing skewness have proved to not improve the prediction power overall, and therefor is now left out of this report.

Heatmaps

Before data processing:



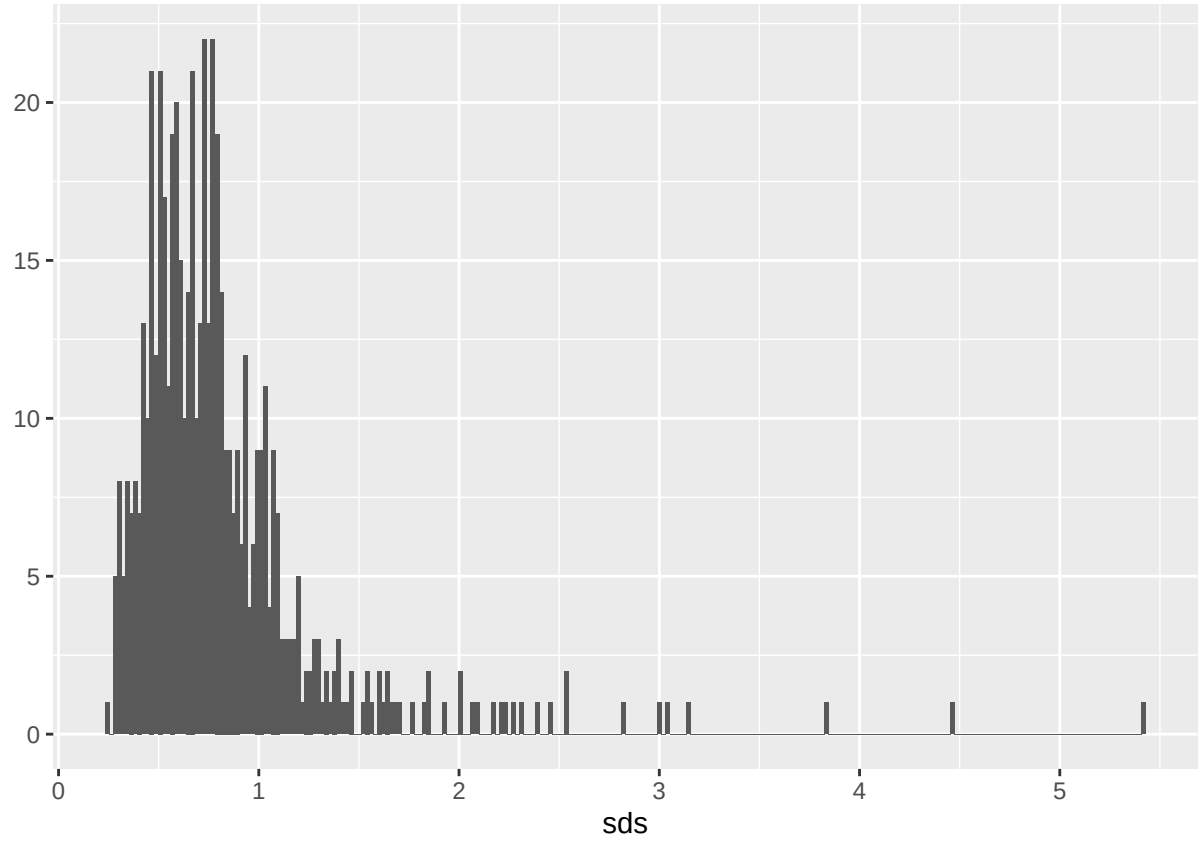
and after data processing:



We can clearly see that data processing has improved the overall "interpretability" of the data, since we can already see some correlations between predictors.

I will also run the nearZero function from the caret package to see if features do not vary much from observation to observation, as well check if there are any features with no variability:

```
sds <- rowSds(liver2_subset_scale$x)
qplot(sds, bins = 256) # there are no features with no variability
```

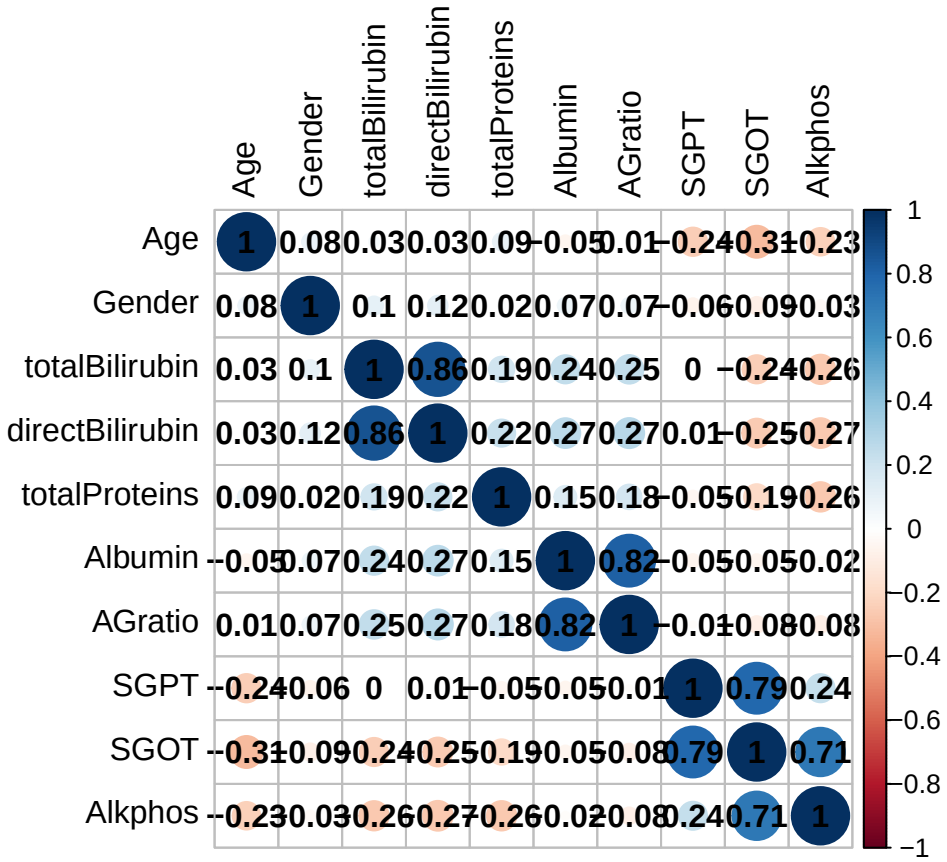



```
nearZeroVar(liver2_subset_scale$x)
```

```
## integer(0)
```

The near zero variance function does not recommend the removal of any feature, as well there seem to be no features with no variability.

Correlation Matrix



From the correlation matrix, it looks like the following pairs of features are strongly correlated: - total Bilirubin and direct Bilirubin - AGratio (Albumin and Globulin ratio) and Albumin - SGPT and SGOT - Alkphos and SGOT

Due to high correlation, some of these columns could be dropped. Motivation: some algorithms degrade in importance with the existence of highly correlated attributes.

Before dropping any columns I will also have a look into the PCA analysis.

Principal Component Analysis

Principal Component Analysis, or PCA, is a dimensionality-reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information from the larger set.

In this analysis I don't have many dimensions, but I will use PCA to check if I can learn more about the training dataset.

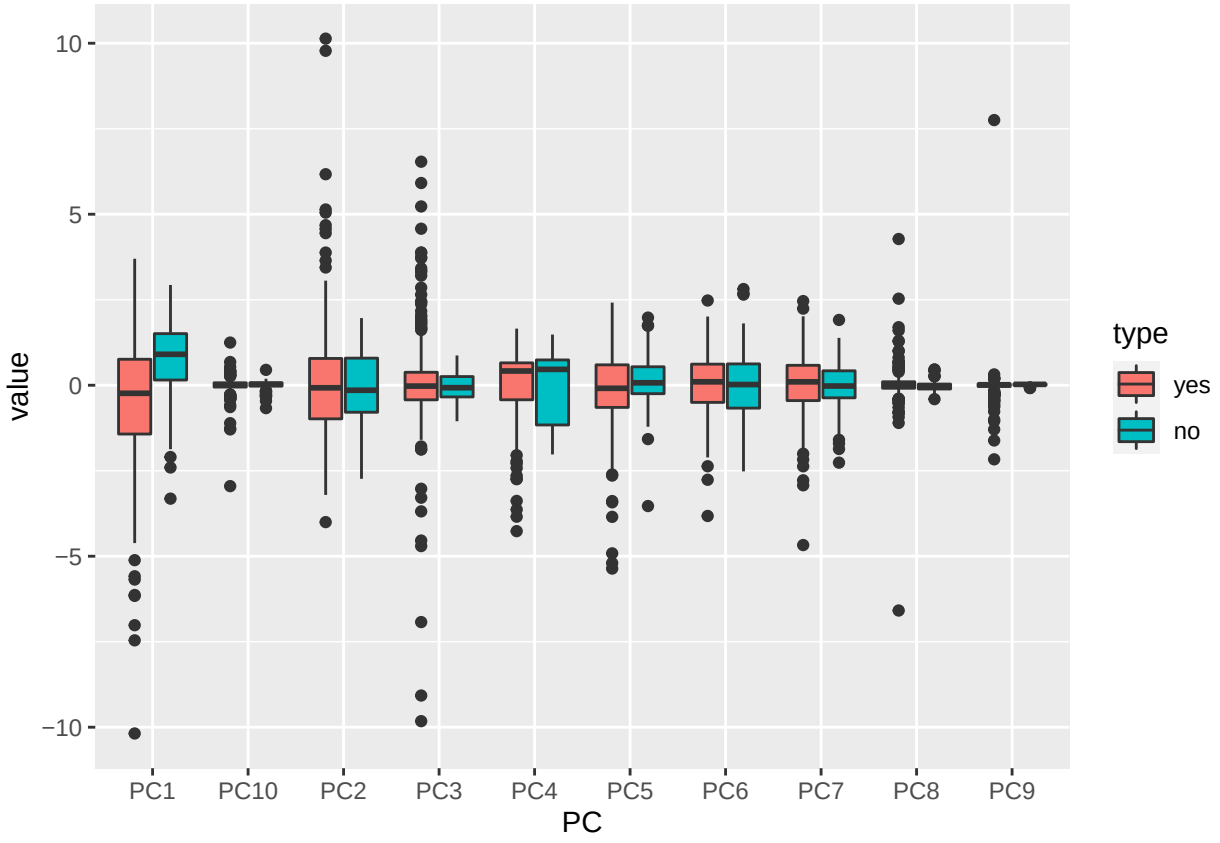
```
pca <- prcomp(liver2_subset_scale$x)
summary(pca)
```

```
## Importance of components:
##                           PC1    PC2    PC3    PC4     PC5     PC6     PC7
## Standard deviation     1.7066 1.4225 1.1562 1.0080 0.95702 0.89614 0.79551
```

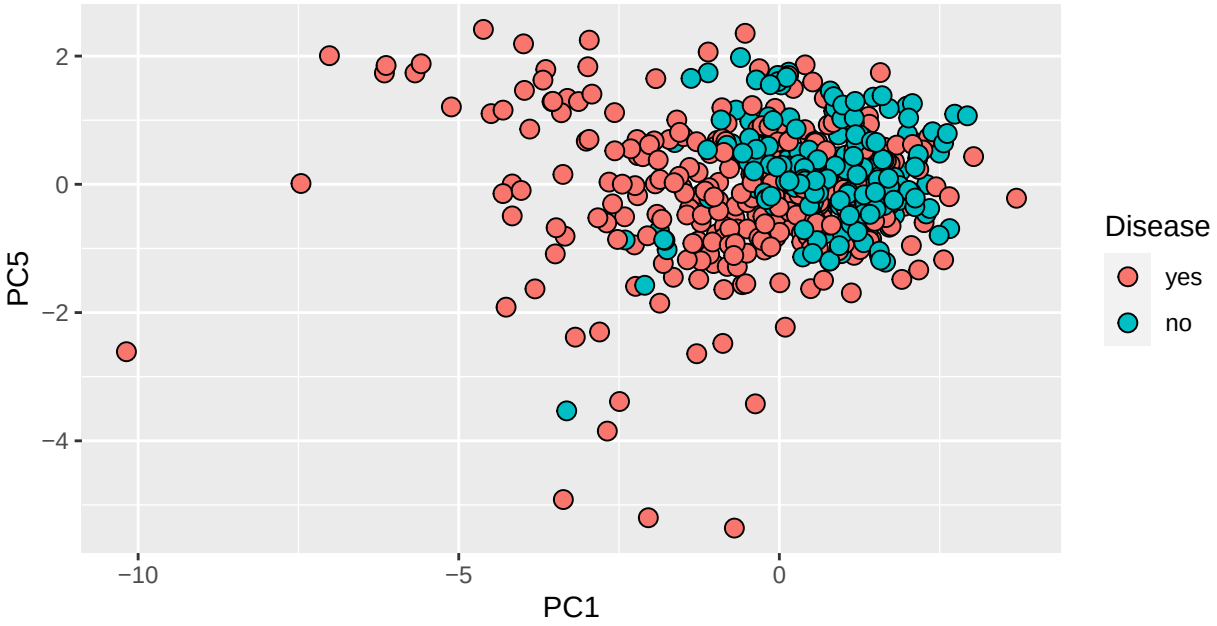
```
## Proportion of Variance 0.2913 0.2024 0.1337 0.1016 0.09159 0.08031 0.06328
## Cumulative Proportion  0.2913 0.4936 0.6273 0.7289 0.82048 0.90079 0.96407
##                            PC8     PC9    PC10
## Standard deviation     0.42060 0.37215 0.20947
## Proportion of Variance 0.01769 0.01385 0.00439
## Cumulative Proportion  0.98176 0.99561 1.00000
```

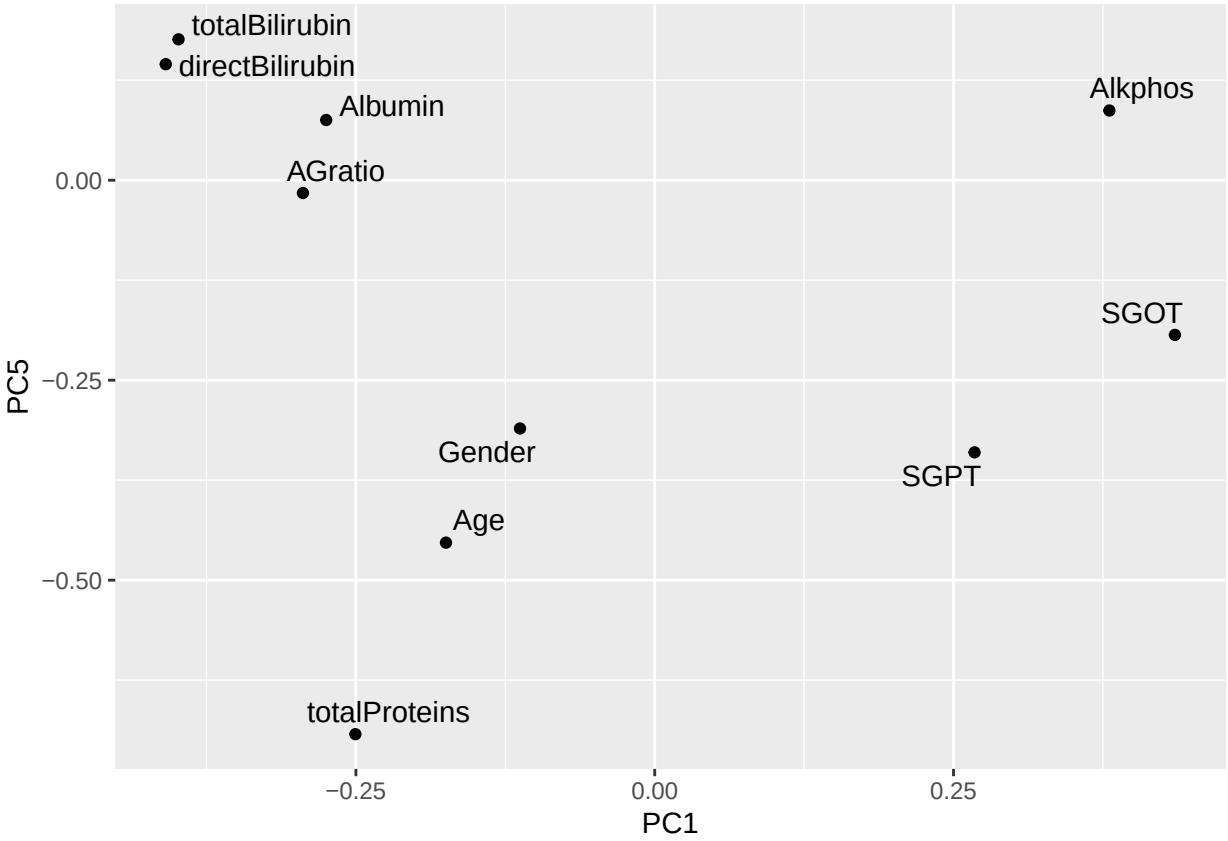
Findings: around 80% of the variance is explained by the first 5 PCs.

When looking at the boxplot, we can see that the IQRs overlap for all PCs, but PC1 has the least overlap. Generally, good predictive features should show distinctions in the boxplots (no or very little overlap).



Studying various interpretations of the first 5 PCs, such as the examples below, suggests further that some of the highly correlated variables can be dropped. Based on this interpretation, as well as running multiple machine learning models with and without dropping variables, I have reached the conclusion that removing totalBilirubin, AGratio and SGPT can actually improve the predictive power of the final model.


PC5
2
0
−2
−4
−10
−5
0
PC1
Disease
yes
no



Classification Predictive Modeling

A bit of introduction to the type of modeling I will apply in this report:

Classification predictive modeling is the task of approximating a mapping function (f) from input variables (X) to discrete output variables (y). The output variables are often called labels or categories. The mapping function predicts the class or category for a given observation. For example, an email of text can be classified as belonging to one of two classes: “spam“ and “not spam“.

My dataset presents a *two-class (binary) classification problem.*

There are many ways to estimate the skill of a classification predictive model, but perhaps the most common is to calculate the classification **accuracy**. The classification accuracy is the percentage of correctly classified examples out of all predictions made.

However on a disease diagnosis dataset such as this one, we need to look into other measures as well. The other 2 measures would be **sensitivity** (the proportion of liver patients who are correctly diagnosed) and **specificity** (the proportion of non-liver patients that are correctly diagnosed).

Another way of quantifying specificity is by the proportion of liver-diagnoses that are actually liver-patients, also called **precision**. Precision in this analysis is also very important, since having high sensitivity in the model will not necessarily mean it is a good prediction, without also looking at precision.

Another measure I will look into is the **F1 Score** (the harmonic average between sensitivity and precision). It is also called the F Score or the F Measure. Put another way, the F1 score conveys the balance between the precision and the recall.

Furthermore, in the medical world, I would assume that it is most important to correctly diagnose diseased patients (therefor having a high sensitivity), than correctly identifying the non-liver patients (therefor having a lower specificity should be fine). In other words, having a higher **False Positive** rate (healthy patients incorrectly diagnosed as diseased) is fine as long as the **False Negatives** is kept at its lowest (diseased patients incorrectly diagnosed as being healthy). My reasoning behind this is that it is far more dangerous to send people with a liver disease home without treatment than incorrectly identifying a liver disease in a healthy person, which can be eliminated later on by doing more lab tests etc.

Split dataset into training and test sets

The dataset will be split into 20% for testing and 80% for training (reasoning is explained in the data exploration section).

Before splitting the dataset, 3 columns will be dropped (the highly correlated predictors): totalBilirubin, AGratio and SGPT.

```
liver_final <- liver2_subset_scale

liver_final$x <- subset(liver_final$x, select = -c(totalBilirubin, AGratio, SGPT))

set.seed(1, sample.kind="Rounding")

test_index <- createDataPartition(liver_final$y, times = 1, p = 0.2, list = FALSE)
test_x <- liver_final$x[test_index,]
test_y <- liver_final$y[test_index]
train_x <- liver_final$x[-test_index,]
train_y <- liver_final$y[-test_index]

rm(test_index)

test <- data.frame(test_x, test_y) %>%
  rename(Disease = test_y)
train <- data.frame(train_x, train_y) %>%
  rename(Disease = train_y)
```

Check if the training and test sets have similar proportions of liver disease:

```
## [1] 0.5869074
```

```
## [1] 0.5892857
```

It looks like the training and test sets are balanced.

In the next subsections I will use various machine learning algorithms, mostly the ones presented in class. One by one, I will also store several measures from each models **confusion matrix** in a dataframe, to be able to compare each model.

Logistic regression model

```r
set.seed(1, sample.kind="Rounding")

train_glm <- train(train_x, train_y, method = "glm")
pred_glm <- predict(train_glm, test_x, type = "raw")
cm <- confusionMatrix(pred_glm, test_y)
```

method	Accuracy	Sensitivity	Specificity	Prevalence	F1score	TruePositives	FalsePositives	FalseNegatives	TrueNegatives
Logistic regression model (glm)	0.6875	0.6515152	0.7391304	0.5892857	0.6188399	43	12	23	34

Variable importance:

```
## glm variable importance
##
##                 Overall
## Albumin          100.00
## directBilirubin   76.50
## Age               56.81
## totalProteins     31.73
## Alkphos           22.36
## SGOT              10.02
## Gender             0.00
```

Linear discriminant analysis

```r
set.seed(1, sample.kind="Rounding")

train_lda <- train(train_x, train_y, method = "lda")
pred_lda <- predict(train_lda, test_x, type = "raw")
cm3 <- confusionMatrix(pred_lda, test_y)
```

method	Accuracy	Sensitivity	Specificity	Prevalence	F1score	TruePositives	FalsePositives	FalseNegatives	TrueNegatives
Linear discriminant analysis (LDA)	0.6785714	0.7878788	0.5217391	0.5892857	0.6742633	52	22	14	24

Variable importance:

```
## ROC curve variable importance
##
##                 Importance
## Albumin              100.00
## totalProteins         81.29
## directBilirubin       81.12
## Alkphos               57.56
## SGOT                  47.41
## Age                   24.95
## Gender                 0.00
```

Quadratic discriminant analysis

```r
set.seed(1, sample.kind="Rounding")

train_qda <- train(train_x, train_y, method = "qda")
pred_qda <- predict(train_qda, test_x, type = "raw")
cm4 <- confusionMatrix(pred_qda, test_y)
```

method	Accuracy	Sensitivity	Specificity	Prevalence	F1score	TruePositives	FalsePositives	FalseNegatives	TrueNegatives
Quadratic discriminant analysis (QDA)	0.625	0.4242424	0.9130435	0.5892857	0.4933262	28	4	38	42

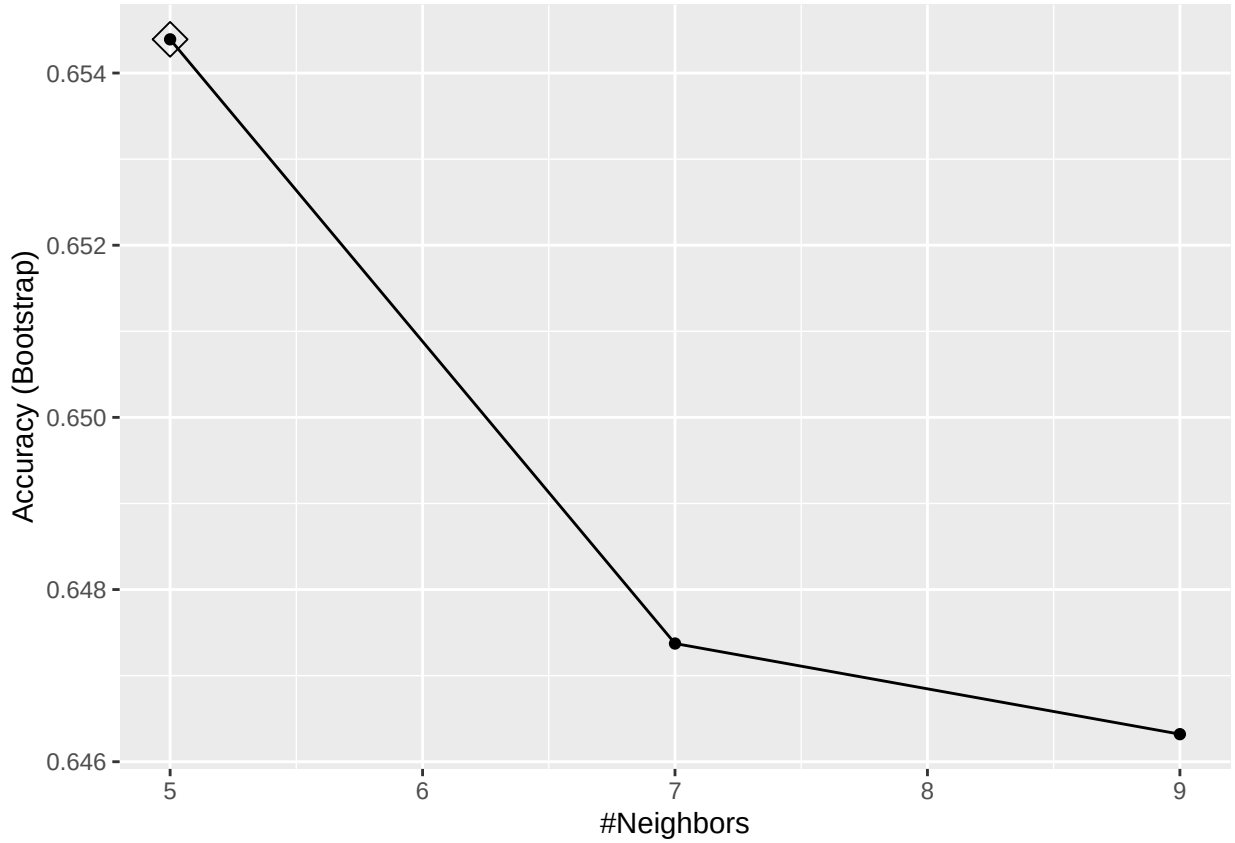
Variable importance:

```
## ROC curve variable importance
##
##                 Importance
## Albumin             100.00
## totalProteins        81.29
## directBilirubin      81.12
## Alkphos              57.56
## SGOT                 47.41
## Age                  24.95
## Gender                0.00
```

k-nearest neighbors

First run the model with no tuning:

```r
set.seed(1, sample.kind="Rounding")

train_knn <- train(Disease ~ ., method = "knn", data = train)  # running the model without any tuning p

ggplot(train_knn, highlight = TRUE)
```




```
pred_knn <- predict(train_knn, test_x, type = "raw")
cm5 <- confusionMatrix(pred_knn, test_y)
```

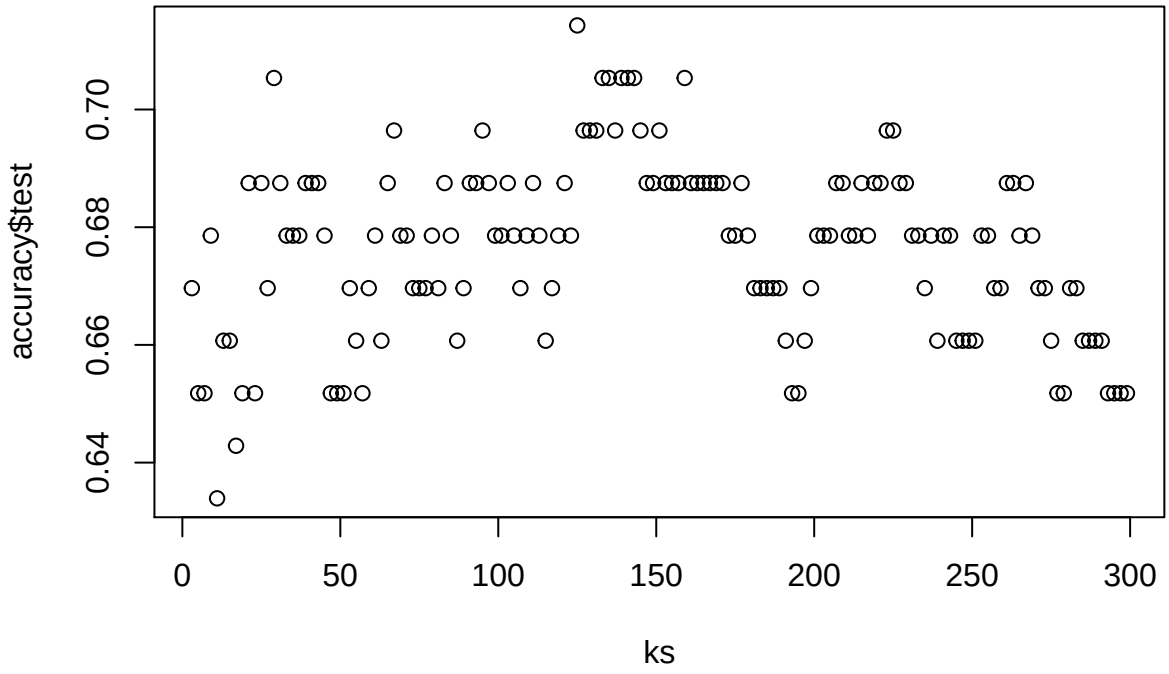
method	Accuracy	Sensitivity	Specificity	Prevalence	F1score	TruePositives	FalsePositives	FalseNegatives	TrueNegatives
k-nearest neighbors (knn) - no tuning	0.6517857	0.6818182	0.6086957	0.5892857	0.6321839	45	18	21	28

Now I will use various methods for model optimization.

Method 1. Max accuracy

```
set.seed(1, sample.kind="Rounding")

ks <- seq(3, 300, 2)

accuracy <- map_df(ks, function(k){
  fit <- knn3(Disease ~ ., data = train, k = k)
  y_hat <- predict(fit, train, type = "class")
  cm_train <- confusionMatrix(data = y_hat, reference = train$Disease)
  train_error <- cm_train$overall["Accuracy"]
  y_hat <- predict(fit, test, type = "class")
  cm_test <- confusionMatrix(data = y_hat, reference = test$Disease)
  test_error <- cm_test$overall["Accuracy"]

  tibble(train = train_error, test = test_error)
})
```

```r
#pick the k that maximizes accuracy
plot(ks,accuracy$test)
```




```r
k1 <- ks[which.max(accuracy$test)]
k1 %>%
  kable() %>%
  kable_styling(full_width = FALSE)
```

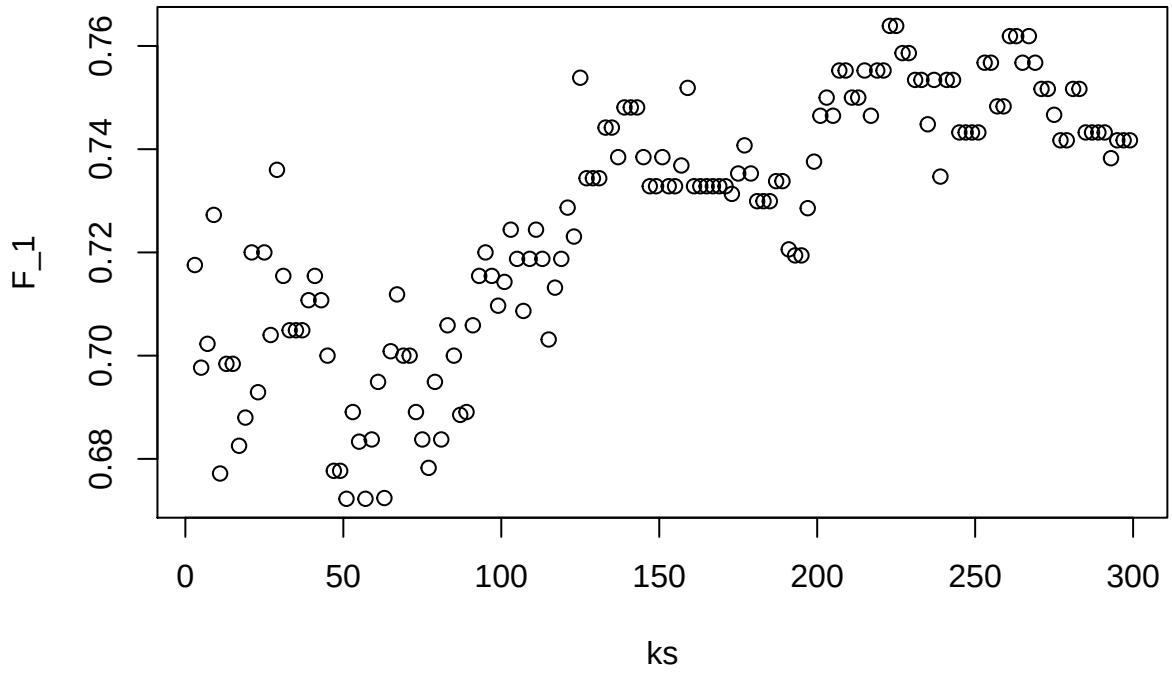
x
125

```r
train_knn_maxacc <- knn3(Disease ~ ., data = train, k = k1)
pred_knn_maxacc <- predict(train_knn_maxacc, test, type = "class")

cm6 <- confusionMatrix(pred_knn_maxacc, test_y)
```

method	Accuracy	Sensitivity	Specificity	Prevalence	F1score	TruePositives	FalsePositives	FalseNegatives	TrueNegatives
k-nearest neighbors (knn) - max. acc.	0.7142857	0.7424242	0.673913	0.5892857	0.65705	49	15	17	31

Method 2. F1 score (balanced accuracy)

```r
set.seed(1, sample.kind="Rounding")

F_1 <- sapply(ks, function(k){
  fit <- knn3(Disease ~ ., data = train, k = k)
  y_hat <- predict(fit, test, type = "class")

  F_meas(data = y_hat, reference = test$Disease)
})

plot(ks, F_1)
```




```r
k2 <- ks[which.max(F_1)]
k2 %>%
  kable() %>%
  kable_styling(full_width = FALSE)
```

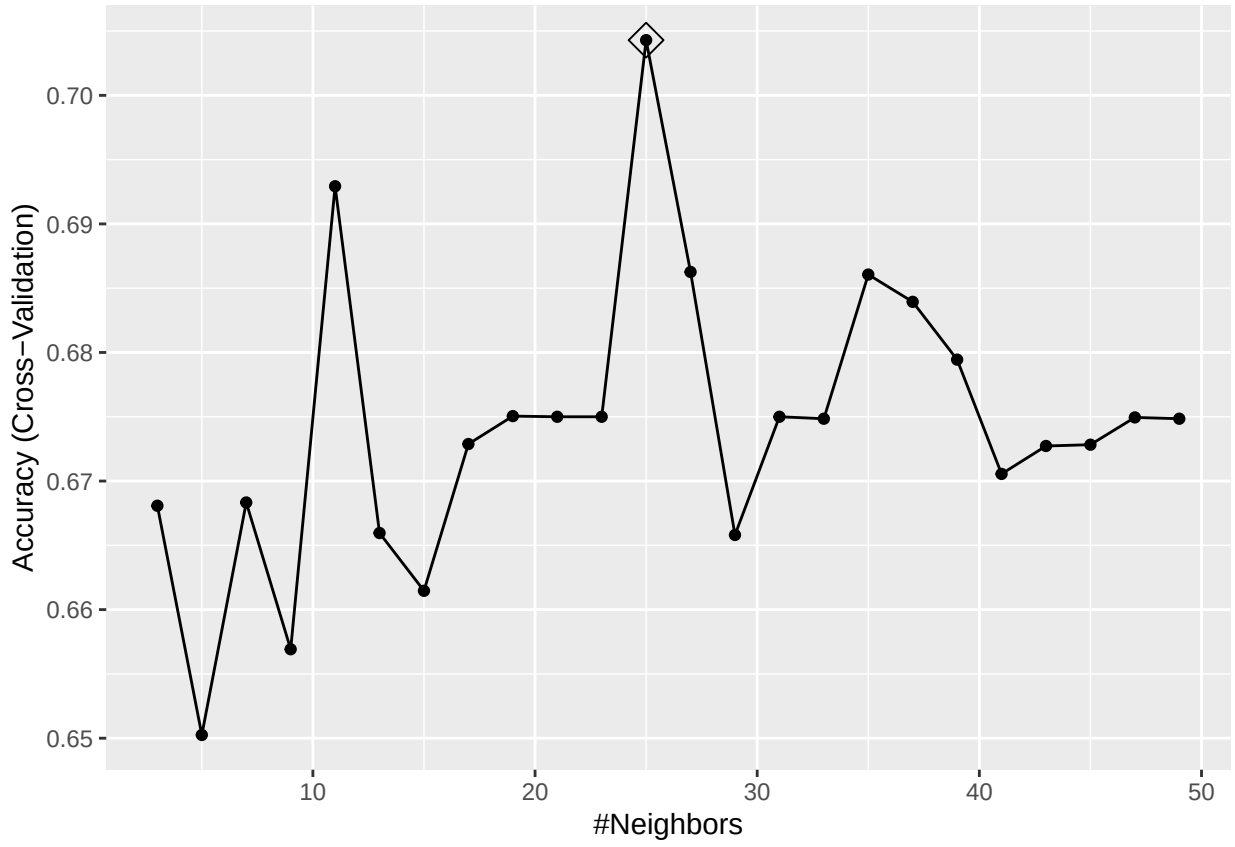
x
223

```r
train_knn_F1 <- knn3(Disease ~ ., data = train, k = k2)
pred_knn_F1 <- predict(train_knn_F1, test, type = "class")

cm7 <- confusionMatrix(pred_knn_F1, test_y)
```

method	Accuracy	Sensitivity	Specificity	Prevalence	F1score	TruePositives	FalsePositives	FalseNegatives	TrueNegatives
k-nearest neighbors (knn) - F1 score	0.6964286	0.8333333	0.5	0.5892857	0.6903766	55	23	11	23

Method 3. Cross validation

```
set.seed(1, sample.kind="Rounding")

control <- trainControl(method = "cv", number = 10, p = .9)
train_knn_cv <- train(Disease ~ ., method = "knn", data = train,
                      tuneGrid = data.frame(k = seq(3, 50, 2)),
                      trControl = control)
ggplot(train_knn_cv, highlight = TRUE)
```




```
train_knn_cv$bestTune %>%
  kable() %>%
  kable_styling(full_width = FALSE)
```

	k
12	25

```
pred_knn_cv <- predict(train_knn_cv, test, type = "raw")

cm8 <- confusionMatrix(pred_knn_cv, test_y)
```

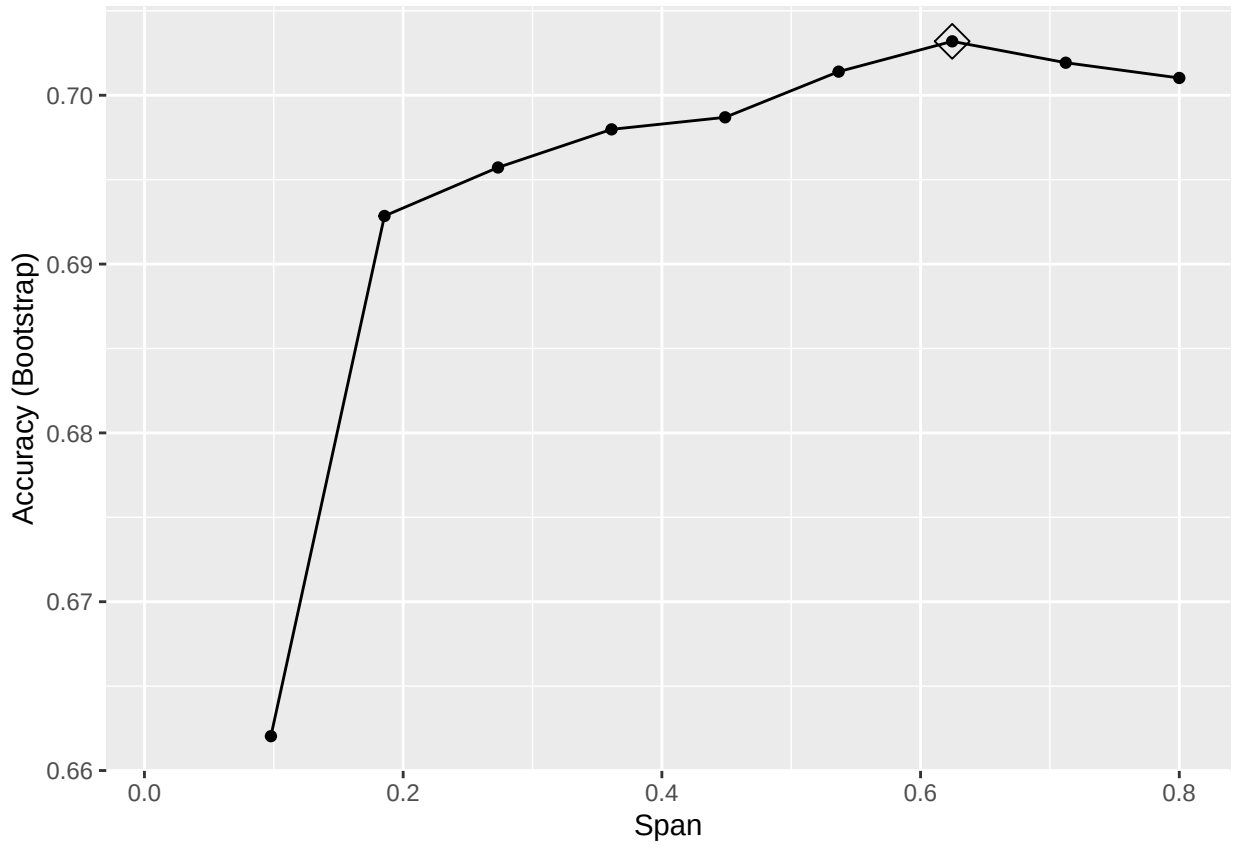
method	Accuracy	Sensitivity	Specificity	Prevalence	F1score	TruePositives	FalsePositives	FalseNegatives	TrueNegatives
k-nearest neighbors (knn) - cross valid.	0.6785714	0.6666667	0.6956522	0.5892857	0.6255924	44	14	22	32

Variable importance:

```
## ROC curve variable importance
##
##                  Importance
## Albumin              100.00
## totalProteins         81.29
## directBilirubin       81.12
## Alkphos               57.56
## SGOT                  47.41
## Age                   24.95
## Gender                 0.00
```

Loess model

```
set.seed(1, sample.kind="Rounding")

grid <- expand.grid(span = seq(0.01, 0.8, len = 10), degree = 1)

train_loess <- train(Disease ~ ., method = "gamLoess",
                     tuneGrid=grid,
                     data = train)

ggplot(train_loess, highlight = TRUE)
```




```
pred_loess <- predict(train_loess, test, type = "raw")

cm9 <- confusionMatrix(pred_loess, test_y)
```

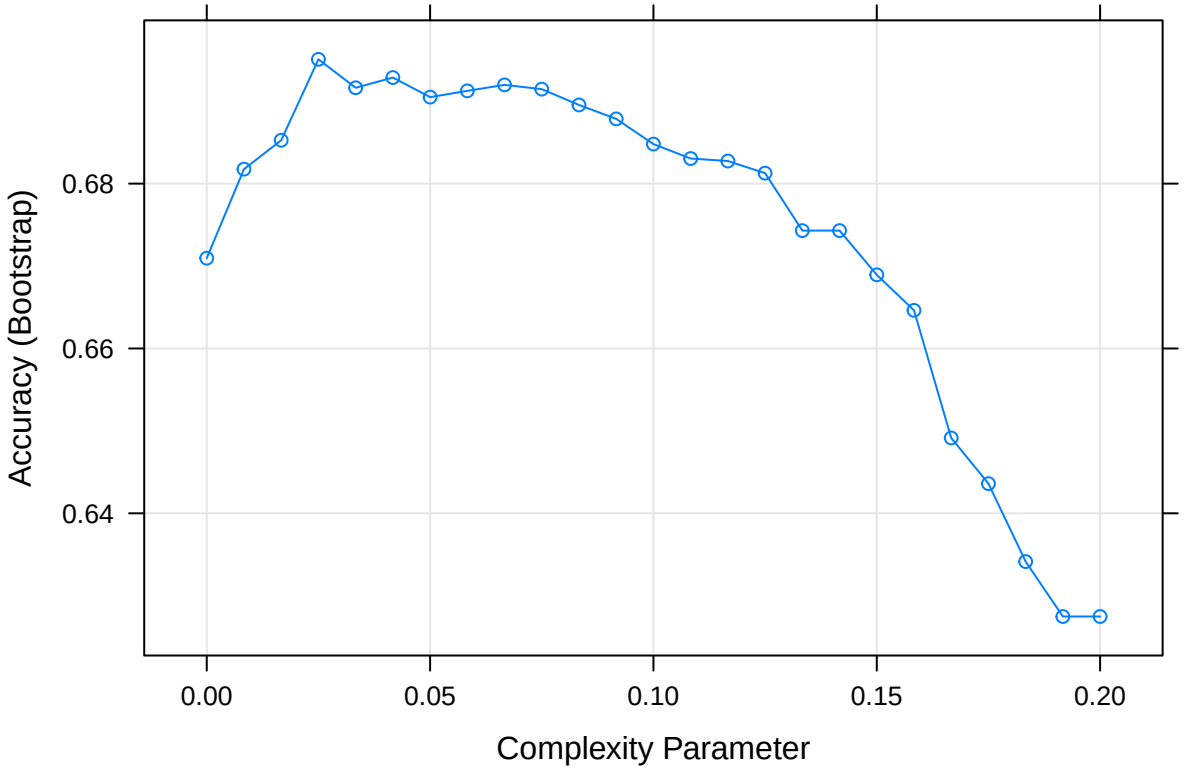
method	Accuracy	Sensitivity	Specificity	Prevalence	F1score	TruePositives	FalsePositives	FalseNegatives	TrueNegatives
Loess model	0.7053571	0.7272727	0.673913	0.5892857	0.6510481	48	15	18	31

Variable importance:

```
## gamLoess variable importance
##
##                  Overall
## totalProteins   100.0000
## Alkphos          75.7941
## directBilirubin  49.9335
## Albumin          20.2672
## Gender            3.1763
## Age               0.7521
## SGOT              0.0000
```

Classification trees

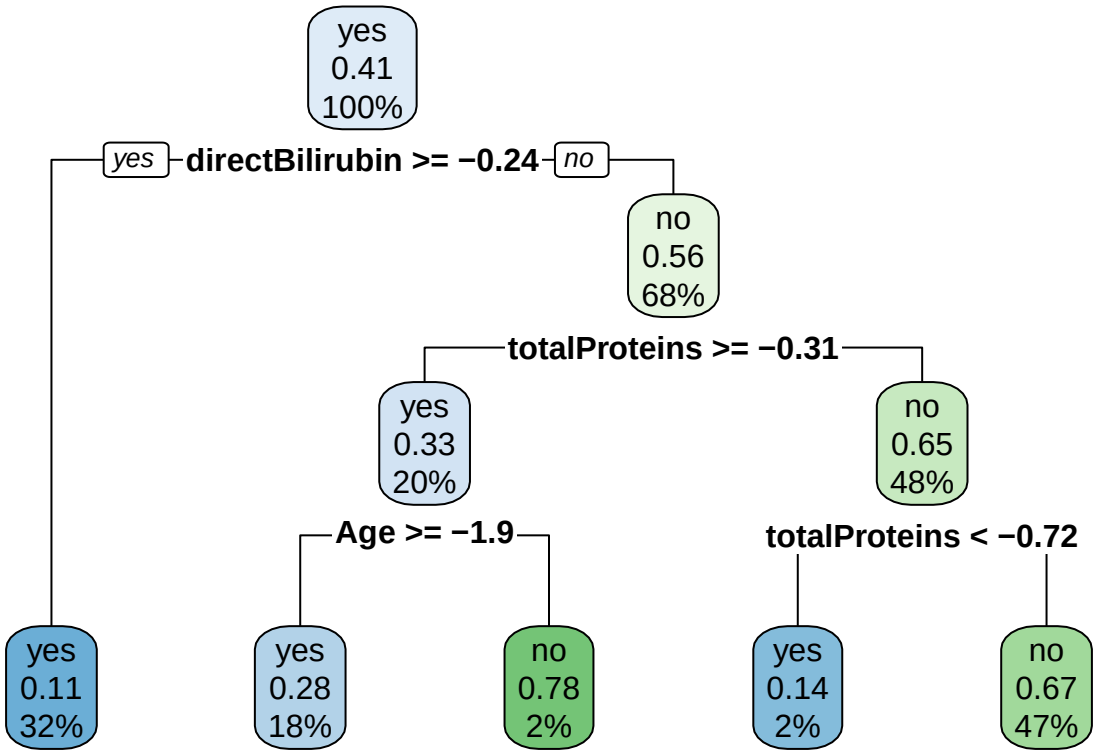
```r
set.seed(1, sample.kind="Rounding")

train_rpart <- train(Disease ~ ., method = "rpart",
                     tuneGrid = data.frame(cp = seq(0, 0.2, len = 25)),
                     data = train)

plot(train_rpart)
```




```r
pred_rpart <- predict(train_rpart, test, type = "raw")

cm10 <- confusionMatrix(pred_rpart, test_y)
```

method	Accuracy	Sensitivity	Specificity	Prevalence	F1score	TruePositives	FalsePositives	FalseNegatives	TrueNegatives
Classification trees	0.7053571	0.7575758	0.6304348	0.5892857	0.6629168	50	17	16	29

The tree:

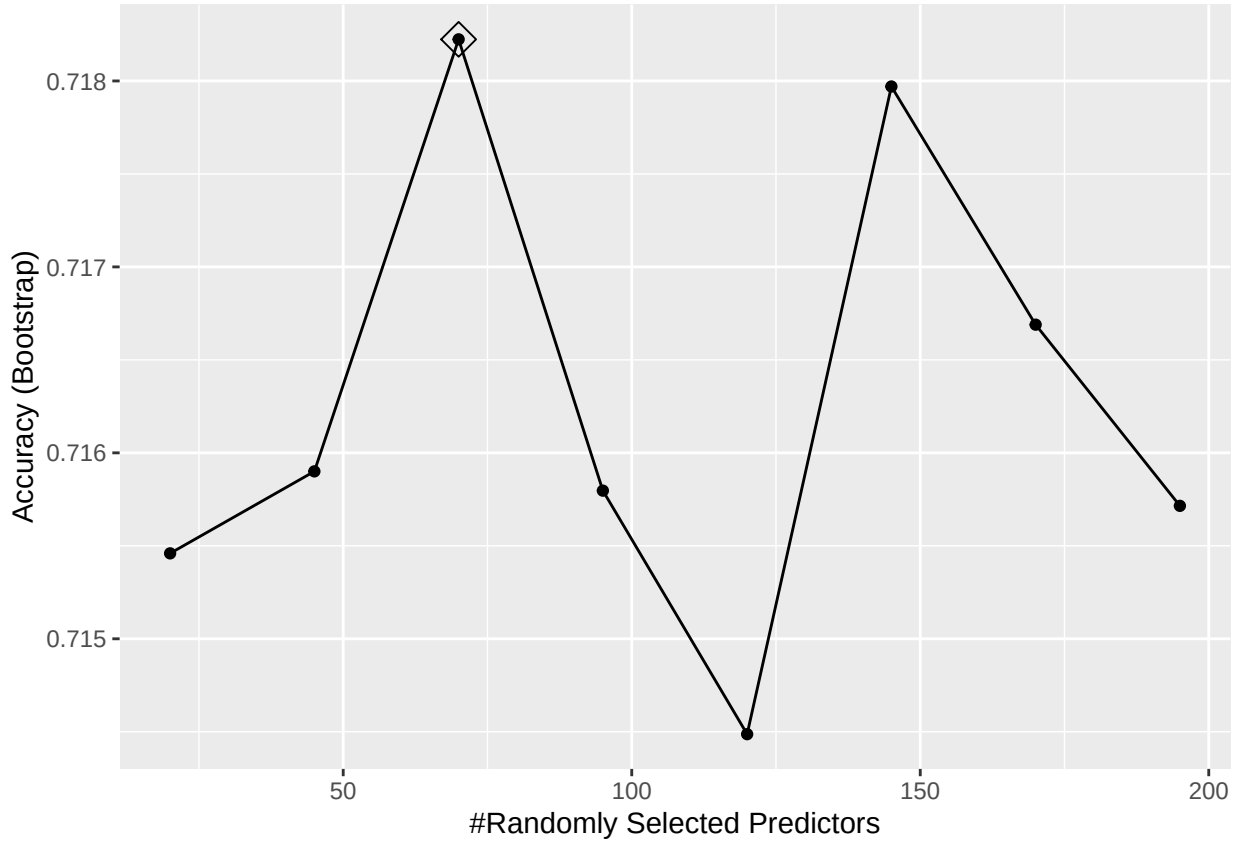


Variable importance:

term	importance	rank
totalProteins	100.00000	1
directBilirubin	81.36848	3
Age	21.80386	6

Random forests

```
set.seed(1, sample.kind="Rounding")

train_rf <- train(train_x, train_y, method = "rf",
                  tuneGrid = data.frame(mtry = seq(20, 200, 25)), importance = TRUE,
                  nodesize = 1)

ggplot(train_rf, highlight = TRUE)
```




```
train_rf$bestTune %>%
  kable() %>%
  kable_styling(full_width = FALSE)
```

	mtry
3	70

```
pred_rf <- predict(train_rf, test_x, type = "raw")

cm11 <- confusionMatrix(pred_rf, test_y)
```

method	Accuracy	Sensitivity	Specificity	Prevalence	F1score	TruePositives	FalsePositives	FalseNegatives	TrueNegatives
Random forests	0.75	0.8484848	0.6086957	0.5892857	0.6955213	56	18	10	28

Variable importance:

```
## rf variable importance
##
##                 Importance
## Albumin             100.00
## directBilirubin      60.92
## totalProteins        48.71
```

```
## Age                  42.38
## Alkphos              32.83
## SGOT                 22.26
## Gender                0.00
```

Ensemble

The motivation behind creating an ensemble is that the combined power of several machine learning models is generally higher than the individual models.

I have chosen the top 2 models with the highest F1 scores, because my areas of concern have been to keep False Negatives as low as possible while having high sensitivity.

It makes sense that the top 2 most performing models according to my metrics are Knn max F1 and Random forests, since these are known to be powerful models for classification datasets.

method	Accuracy	Sensitivity	Specificity	Prevalence	F1score	TruePositives	FalsePositives	FalseNegatives	TrueNegatives
Random forests	0.7500000	0.8484848	0.6086957	0.5892857	0.6955213	56	18	10	28
k-nearest neighbors (knn) - F1 score	0.6964286	0.8333333	0.5000000	0.5892857	0.6903766	55	23	11	23

An overview of all the models ordered by F1 scores:

method	Accuracy	Sensitivity	Specificity	Prevalence	F1score	TruePositives	FalsePositives	FalseNegatives	TrueNegatives
Ensemble (knn F1 + rf)	0.8035714	0.7878788	0.8260870	0.8666667	0.8253968	NA	NA	NA	NA
Random forests	0.7500000	0.8484848	0.6086957	0.5892857	0.6955213	56	18	10	28
k-nearest neighbors (knn) - F1 score	0.6964286	0.8333333	0.5000000	0.5892857	0.6903766	55	23	11	23
Linear discriminant analysis (LDA)	0.6785714	0.7878788	0.5217391	0.5892857	0.6742633	52	22	14	24
Classification trees	0.7053571	0.7575758	0.6304348	0.5892857	0.6629168	50	17	16	29
k-nearest neighbors (knn) - max. acc.	0.7142857	0.7424242	0.6739130	0.5892857	0.6570500	49	15	17	31
Loess model	0.7053571	0.7272727	0.6739130	0.5892857	0.6510481	48	15	18	31
k-nearest neighbors (knn) - no tuning	0.6517857	0.6818182	0.6086957	0.5892857	0.6321839	45	18	21	28
k-nearest neighbors (knn) - cross valid.	0.6785714	0.6666667	0.6956522	0.5892857	0.6255924	44	14	22	32
Logistic regression model (glm)	0.6875000	0.6515152	0.7391304	0.5892857	0.6188399	43	12	23	34
Quadratic discriminant analysis (QDA)	0.6250000	0.4242424	0.9130435	0.5892857	0.4933262	28	4	38	42

The ensemble seems to outperform the individual models, because its F1 score is the highest.

Final model

The final step of the report is to apply the final model - the ensemble between KNN - F1 score, Random forest and LDA, to the validation dataset. As a training set I am using the original training subset, after NAs and duplicates have been removed. My reason for not taking the training dataset with all the data processing is because those steps were performed only for finding the best models.

Applying the ensemble model

method	Accuracy	Sensitivity	Specificity	Prevalence	F1score	TruePositives	FalsePositives	FalseNegatives	TrueNegatives
KNN - F1	0.7068966	1	0	0.7068966	0.8282828	82	34	0	0

method	Accuracy	Sensitivity	Specificity	Prevalence	F1score	TruePositives	FalsePositives	FalseNegatives	TrueNegatives
Random forests	0.6810345	0.8536585	0.2647059	0.7068966	0.7733764	70	25	12	9

method	Accuracy	Sensitivity	Specificity	Prevalence	F1score	TruePositives	FalsePositives	FalseNegatives	TrueNegatives
KNN - F1	0.7068966	1.0000000	0.0000000	0.7068966	0.8282828	82	34	0	0
Random forests	0.6810345	0.8536585	0.2647059	0.7068966	0.7733764	70	25	12	9
Ensemble	0.6810345	0.8536585	0.2647059	0.7368421	0.7909605	NA	NA	NA	NA

Results

Applying the final model (the ensemble) to the validation dataset (the final hold-out test set), results in a good F1 score, however given the high prevalence of the data, I would not be comfortable enough to say that this is an optimal model, since it may out-predict true positives (sensitivity is quite close to 1), as well as predict too many false positives (and run the risk of sending people home that actually have a liver disease):

method	Accuracy	Sensitivity	Specificity	Prevalence	F1score
Ensemble	0.6810345	0.8536585	0.2647059	0.7368421	0.7909605

All in all, it seems that Albumin and total Bilirubin predictors seem to be the most powerful, and Gender the least - we could also see these findings in the data exploration and processing steps of the analysis.

Conclusions

Overall, this analysis has offered me many new insights into machine learning, and in particular into classification predictions. My knowledge as of now is still too limited and narrow to fully grasp all the possibilities and paths I could have taken in the analysis.

For example, the entire report could have taken a different approach by using k-fold cross-validation on the entire dataset, instead of splitting it into train, test and validation sets. Second, I learned that there are many different ways to measure the predictive powers of any given model, and perhaps a good approach in general would be to know from the beginning what exactly you are trying to achieve. Another task is to pre-process data, which varies widely depending on the dataset. I have used a few methods, but there are many more out there which might have performed better. And last but not least are the machine learning algorithms themselves and tuning parameters, which I need to learn more about, as well look into algorithms that I have not used but could have potentially performed better. The measures to look at and optimize depend a lot on what the purpose of the analysis is.

A big win in such an analysis would be, I suspect, to get insights from stakeholders on what they consider to be the most important parameters to optimize. If a panel of doctors would request such an analysis in real life, would they be most interested in keeping the False positives as low as possible, or would it be a combination of low False positives but also low False negatives, if let's say the hospital cannot afford misdiagnosis of healthy people (not enough availability of tests, no extra beds in hospitals etc.)

In conclusion I would like to say a big thank you to professor Irizarry as well as the supporting teachers, for opening up my eyes to this vast expanse of new and exciting territory. My next step is to continue learning, and one day return to this report and try to make it better.