Project 1

Introduction and Overview

Hepatitis C is a disease of the liver caused by a hepatitis C virus (HCV) infection. The virulence of HCV varies, and the virus can cause both acute and chronic illness. Hepatitis C can be a life-threatening condition. There is currently no effective vaccine for the HCV virus; however, antiviral medications can be used to treat the disease. These medications can cure greater than 95% of those infected with HCV, but widespread access to this treatment is low.

Crucially, early detection, and subsequent fast treatment of the disease, can prevent the development of serious liver damage (such as fibrosis, cirrhosis, and hepatocellular carcinoma – cancer of the liver cells) – this can improve a patient's health in the long-term. According to the World Health Organisations (WHO), it is estimated that 58 million people globally are infected with HCV. It is also estimated that 290,000 deaths occurred in 2019 from related complications of HCV infection; most of these deaths were from cirrhosis, and hepatocellular carcinoma. Medical classification with machine learning is a fast-burgeoning field in data science. Its application in the detection of liver disease can provide enormous clinical and diagnostic benefits, such as faster disease detection and improved patient prognoses. We shall attempt, in this project, to create such a classifier.

That data used in this project consisted of 615 observations, with 12 features, and a target category. Two classifiers are compared in this project: random forest (RF), and support vector machine (SVM). The data are split into a test and train set; the training set, with 12 features, is used by the classifiers in training. The metrics of accuracy, sensitivity, and specificity were used to compare the performance of the two models. Over-sampling, using the method SMOTE-NC (synthetic minority over-sampling technique – nominal continuous), was employed to balance the minority data in a way that tries to improve the model, without introducing too much bias. PCA was used as a method of data analysis. The best performing model, for this instance of binary classification, was the hyper-parameter tuned RF model with SMOTE-NC, with the results (97.8%, 0.863, 0.994). We discuss whether such a model may be an option for the prediction of the presence of hepatitis C.

The Data

The dataset used in this project is the freely-available, and publicly-accessible, Hepatitis C dataset. The dataset was accessed via the MCI Machine Learning Repository [1]. The dataset is comprised of 615 instances (or observations). There are 12 features and a target attribute called 'Category.' The target attribute has values of 'blood donors' (and 'suspected blood donors'), as well as 'hepatitis C,' 'fibrosis,' and 'cirrhosis.'

For the purposes of binary classification, we shall group these data into two categories: 'Donor' (comprised of 'blood donors' and 'suspected blood donors'), and 'Liver Disease' (comprised of instances with 'hepatitis C,' 'fibrosis,' and 'cirrhosis').

This list of features are as follows: age (in years), sex (m,f), and the blood markers Albumin (ALB), Alkaline phosphatase (ALP), Alanine amino-transferase (ALT), Aspartate amino-transferase (AST), Bilirubin (BIL), Choline esterase (CHE), Cholesterol (CHOL), Creatinine (CREA), Gamma-glutamyl transferase (GGT), Protein (PROT).

Exploratory Data Analysis (EDA)

```
library(readr)
library(archive)

# read data from UCI Machine Learning Repository
url_ <- "https://archive.ics.uci.edu/static/public/571/hcv+data.zip"
data <- read_csv(archive_read(url_, 1), show_col_types = FALSE)
```

```
## New names:
## * `` -> `...1`
```

```
data <- subset(data, select = -...1)

# Let's look at the data, and do some EDA

# Data dimensions
dim(data)
```

```
## [1] 615  13
```

```
# Check for missing data (where there is an 'NA')
colSums(is.na(data))
```

```
## Category      Age      Sex      ALB      ALP      ALT      AST      BIL
##        0        0        0        1       18        1        0        0
##      CHE     CHOL     CREA      GGT     PROT
##        0       10        0        0        1
```

```
# Check for missing data
no.na <- sum(colSums(is.na(data)))

# There are 615 rows, and 13 columns, so the total number of data points is:
no.data.points <- (ncol(data)) * nrow(data)

# Proportion of total data is missing
(no.na / no.data.points) * 100
```

```
## [1] 0.3877423
```

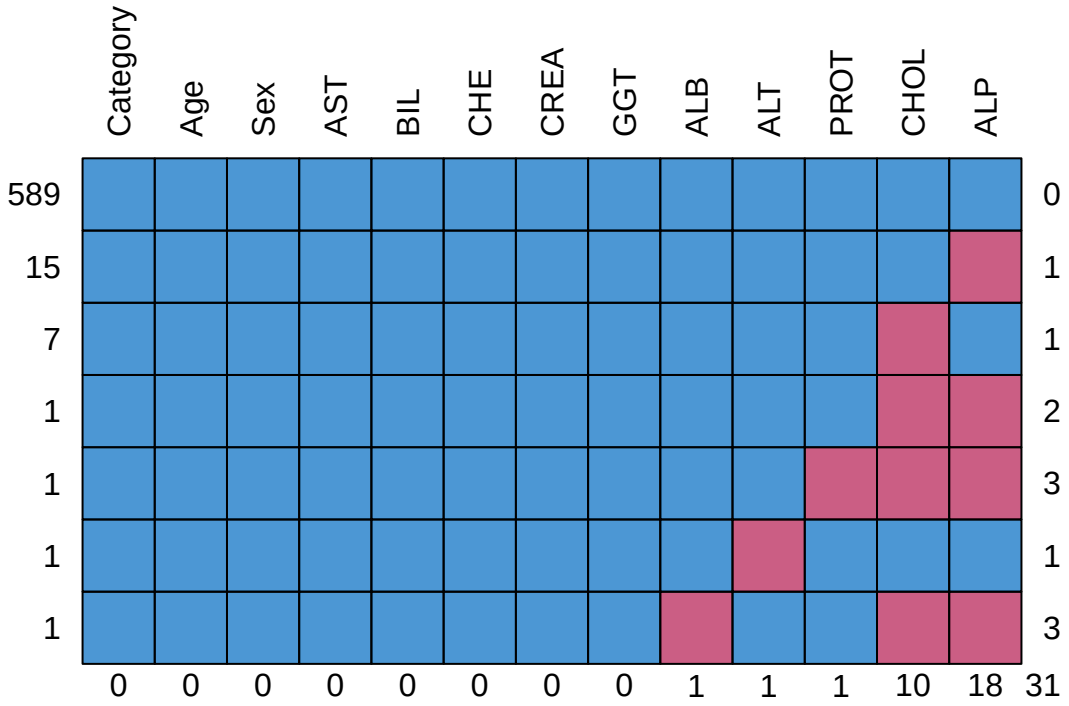
```
# Inspect the data
str(data)
```

```
## tibble [615 x 13] (S3: tbl_df/tbl/data.frame)
##  $ Category: chr [1:615] "0=Blood Donor" "0=Blood Donor" "0=Blood Donor" "0=Blood Donor" ...
##  $ Age     : num [1:615] 32 32 32 32 32 32 32 32 32 32 ...
##  $ Sex     : chr [1:615] "m" "m" "m" "m" ...
##  $ ALB     : num [1:615] 38.5 38.5 46.9 43.2 39.2 41.6 46.3 42.2 50.9 42.4 ...
##  $ ALP     : num [1:615] 52.5 70.3 74.7 52 74.1 43.3 41.3 41.9 65.5 86.3 ...
##  $ ALT     : num [1:615] 7.7 18 36.2 30.6 32.6 18.5 17.5 35.8 23.2 20.3 ...
##  $ AST     : num [1:615] 22.1 24.7 52.6 22.6 24.8 19.7 17.8 31.1 21.2 20 ...
##  $ BIL     : num [1:615] 7.5 3.9 6.1 18.9 9.6 12.3 8.5 16.1 6.9 35.2 ...
##  $ CHE     : num [1:615] 6.93 11.17 8.84 7.33 9.15 ...
##  $ CHOL    : num [1:615] 3.23 4.8 5.2 4.74 4.32 6.05 4.79 4.6 4.1 4.45 ...
##  $ CREA    : num [1:615] 106 74 86 80 76 111 70 109 83 81 ...
##  $ GGT     : num [1:615] 12.1 15.6 33.2 33.8 29.9 91 16.9 21.5 13.7 15.9 ...
##  $ PROT    : num [1:615] 69 76.5 79.3 75.7 68.7 74 74.5 67.1 71.3 69.9 ...
```

We can see that there are five features (ALB, ALP, ALT, CHOL, PROT) in whose columns are entries labelled 'NA.' Less than 1% of the data is missing or has the value 'NA.' This is quite a small value, and therefore our subsequent data imputation is likely to be successful.

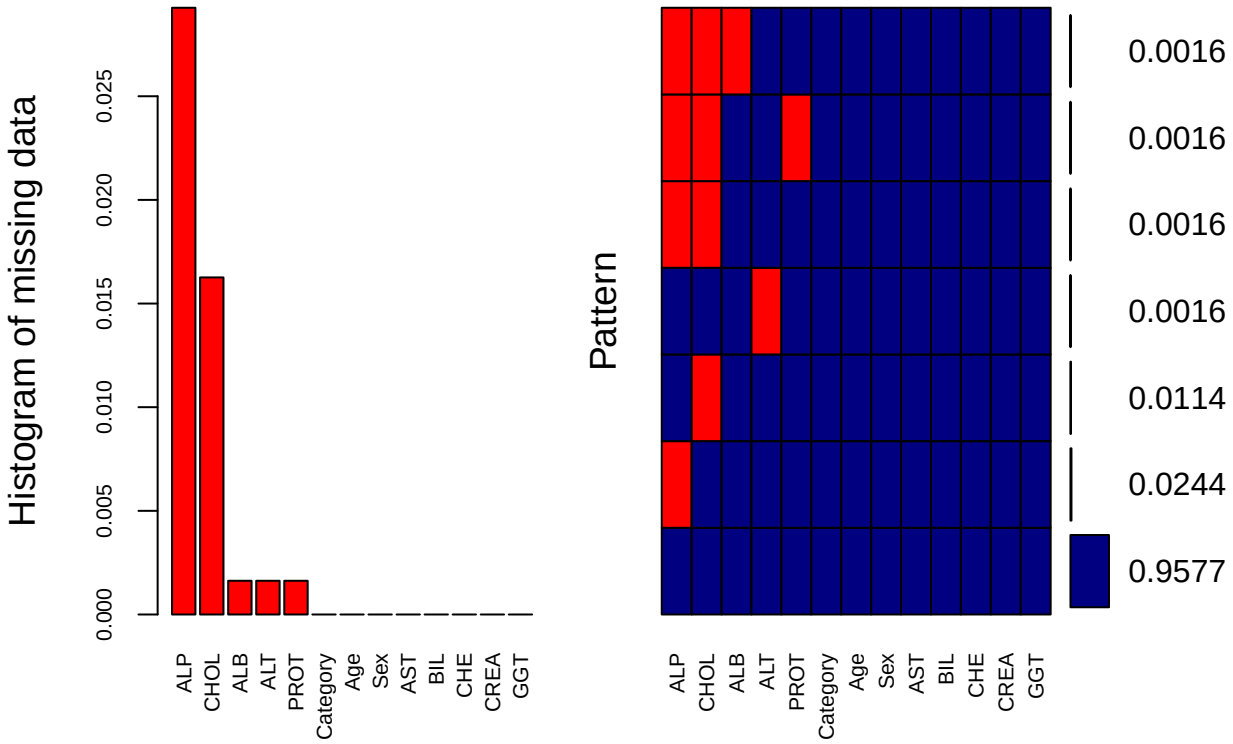
Let us create plots to visualise the missing data

```
library(mice)
md.pattern(data, rotate.names=TRUE)
```




```
##     Category Age Sex AST BIL CHE CREA GGT ALB ALT PROT CHOL ALP
## 589        1   1   1   1   1   1    1   1   1   1    1    1   1   0
## 15         1   1   1   1   1   1    1   1   1   1    1    1   0   1
## 7          1   1   1   1   1   1    1   1   1   1    1    0   1   1
## 1          1   1   1   1   1   1    1   1   1   1    1    0   0   2
## 1          1   1   1   1   1   1    1   1   1   1    0    0   0   3
## 1          1   1   1   1   1   1    1   1   1   0    1    1   1   1
## 1          1   1   1   1   1   1    1   1   0   1    1    0   0   3
##            0   0   0   0   0   0    0   0   1   1    1   10  18  31
```

```
# The following visual plot takes inspiration the website linked below
# https://datascienceplus.com/imputing-missing-data-with-r-mice-package/

# Create plots to visualise the nature of the missing data
library(VIM)
aggr_plot <- aggr(data, col=c('navyblue','red'),
                  numbers=TRUE,
                  sortVars=TRUE,
                  labels=names(data),
                  cex.axis=.7,
                  gap=3,
                  ylab=c("Histogram of missing data","Pattern"))
```




```
##
##  Variables sorted by number of missings:
##  Variable       Count
##       ALP 0.029268293
##      CHOL 0.016260163
##       ALB 0.001626016
##       ALT 0.001626016
##      PROT 0.001626016
##  Category 0.000000000
##       Age 0.000000000
##       Sex 0.000000000
##       AST 0.000000000
##       BIL 0.000000000
##       CHE 0.000000000
##      CREA 0.000000000
##       GGT 0.000000000
```

Dealing With Missing Values – Imputation

Often, in the realm of data science and data analytics, one comes across many datasets within which the data are sparse. That is, there are multiple observations with data-values that are empty, 'NULL,' or simply coded as 'NA.'

There are a few issues presented by the presence of missing data. Firstly, it can create bias. Next, it can make more difficult – and sometimes intractable – the utilisation, and the analysis, of data. Furthermore, the efficiency of a model can be greatly reduced by its presence. [2]

So, how can we attempt to fix this issue? Importantly, one should not rush to simply erase the data with missing values (with the ostensible purpose of 'improving' or 'cleaning' the data) from the dataset entirely – this presents a number of problems. This method can lead to poor, or completely false, results. Additionally, removing data could mask (or sabotage) the true nature, pattern, and values within the overall data.

A process called imputation allows use to replace these absent data with (sometimes cleverly chosen) substitutes. A naive approach could be to impute zeros into all the missing values. Alternatively, one may use the mean of the column, or the median. These methods are often bad, since they do not take into consideration the relationships among the many other features. Also, it is limited to certain categories of data: namely numerical data, but not categorical, or strings.

Multiple imputation by chained equations (MICE) was used, and it has the following steps [3]:

1. Impute values into the missing positions are 'placeholders.' E.g., the mean can be imputed.
2. The the imputed 'placeholders' for only one variable (or feature) – call this 'V' – are put back to missing, or 'NA.'
3. Observed values from V are regressed onto all of the other features. Put differently, V is used as the dependent variable of a regression model. The independent variables are all the other variables.
4. The missing values of the original feature V (i.e., the dependent feature) are subsequently replaced by values that are predicted by the regression model – these are the imputations. Note: as you impute other features, you use the changed observations of the prior features as independent variables in the following imputation.
5. For each feature that contains missing values, steps 2-4 are repeated until all the features have been imputed on. I.e., there is no more missing data. At this point, we have iterated through one 'cycle.'
6. Repeat step 5 until the values imputed values converge, or the chosen number of cycles has been reached.

Ten cycles have been performed [4]. The method used is predictive mean matching (PMM). PMM calculates values for each target feature 'V.' For each missing data value in V, 10 (it can be any specified value) values are calculated to choose from. One value from the 10 is chosen at random, and this is the value which is imputed. The distribution of the missing data point is assumed to be the same as that of the observed data of the 10 values.

```r
# Implement MICE, using predictive mean matching (PMM)
set.seed(321)
imputed_data <- mice(data, m=10, method = "pmm")
```

```
##
##  iter imp variable
##   1   1  ALB  ALP  ALT  CHOL  PROT
##   1   2  ALB  ALP  ALT  CHOL  PROT
##   1   3  ALB  ALP  ALT  CHOL  PROT
##   1   4  ALB  ALP  ALT  CHOL  PROT
##   1   5  ALB  ALP  ALT  CHOL  PROT
##   1   6  ALB  ALP  ALT  CHOL  PROT
##   1   7  ALB  ALP  ALT  CHOL  PROT
##   1   8  ALB  ALP  ALT  CHOL  PROT
##   1   9  ALB  ALP  ALT  CHOL  PROT
##   1   10  ALB  ALP  ALT  CHOL  PROT
##   2   1  ALB  ALP  ALT  CHOL  PROT
##   2   2  ALB  ALP  ALT  CHOL  PROT
##   2   3  ALB  ALP  ALT  CHOL  PROT
##   2   4  ALB  ALP  ALT  CHOL  PROT
##   2   5  ALB  ALP  ALT  CHOL  PROT
##   2   6  ALB  ALP  ALT  CHOL  PROT
##   2   7  ALB  ALP  ALT  CHOL  PROT
##   2   8  ALB  ALP  ALT  CHOL  PROT
##   2   9  ALB  ALP  ALT  CHOL  PROT
##   2   10  ALB  ALP  ALT  CHOL  PROT
##   3   1  ALB  ALP  ALT  CHOL  PROT
##   3   2  ALB  ALP  ALT  CHOL  PROT
##   3   3  ALB  ALP  ALT  CHOL  PROT
```

```
##   3   4  ALB  ALP  ALT  CHOL  PROT
##   3   5  ALB  ALP  ALT  CHOL  PROT
##   3   6  ALB  ALP  ALT  CHOL  PROT
##   3   7  ALB  ALP  ALT  CHOL  PROT
##   3   8  ALB  ALP  ALT  CHOL  PROT
##   3   9  ALB  ALP  ALT  CHOL  PROT
##   3   10  ALB  ALP  ALT  CHOL  PROT
##   4   1  ALB  ALP  ALT  CHOL  PROT
##   4   2  ALB  ALP  ALT  CHOL  PROT
##   4   3  ALB  ALP  ALT  CHOL  PROT
##   4   4  ALB  ALP  ALT  CHOL  PROT
##   4   5  ALB  ALP  ALT  CHOL  PROT
##   4   6  ALB  ALP  ALT  CHOL  PROT
##   4   7  ALB  ALP  ALT  CHOL  PROT
##   4   8  ALB  ALP  ALT  CHOL  PROT
##   4   9  ALB  ALP  ALT  CHOL  PROT
##   4   10  ALB  ALP  ALT  CHOL  PROT
##   5   1  ALB  ALP  ALT  CHOL  PROT
##   5   2  ALB  ALP  ALT  CHOL  PROT
##   5   3  ALB  ALP  ALT  CHOL  PROT
##   5   4  ALB  ALP  ALT  CHOL  PROT
##   5   5  ALB  ALP  ALT  CHOL  PROT
##   5   6  ALB  ALP  ALT  CHOL  PROT
##   5   7  ALB  ALP  ALT  CHOL  PROT
##   5   8  ALB  ALP  ALT  CHOL  PROT
##   5   9  ALB  ALP  ALT  CHOL  PROT
##   5   10  ALB  ALP  ALT  CHOL  PROT
```

```
## Warning: Number of logged events: 2
```

```
# Check to see that the imputing process has worked -- it has
imp.data <- complete(imputed_data,10)
data[542,]
```

```
## # A tibble: 1 x 13
##   Category      Age Sex     ALB   ALP   ALT   AST   BIL   CHE  CHOL  CREA   GGT
##   <chr>       <dbl> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 1=Hepatitis    19 m        41    NA    87    67    12  7.55   3.9    62    65
## # i 1 more variable: PROT <dbl>
```

```
imp.data[542,]
```

```
##        Category Age Sex ALB  ALP ALT AST BIL  CHE CHOL CREA GGT PROT
## 542 1=Hepatitis  19   m  41 59.5  87  67  12 7.55  3.9   62  65   75
```

```
# Here, we see that there are no more NA's
sapply(imp.data, function(x) sum(is.na(x)))
```

```
## Category      Age      Sex      ALB      ALP      ALT      AST      BIL
##        0        0        0        0        0        0        0        0
##      CHE     CHOL     CREA      GGT     PROT
##        0        0        0        0        0
```

It is useful to create binary dummy variables for the male/female categorical data. This allows use to perform PCA, and oversampling, for example.

```
# Convert male or female data to numeric type
imp.data$Sex <- ifelse(imp.data$Sex=="m", 1, 0)
```

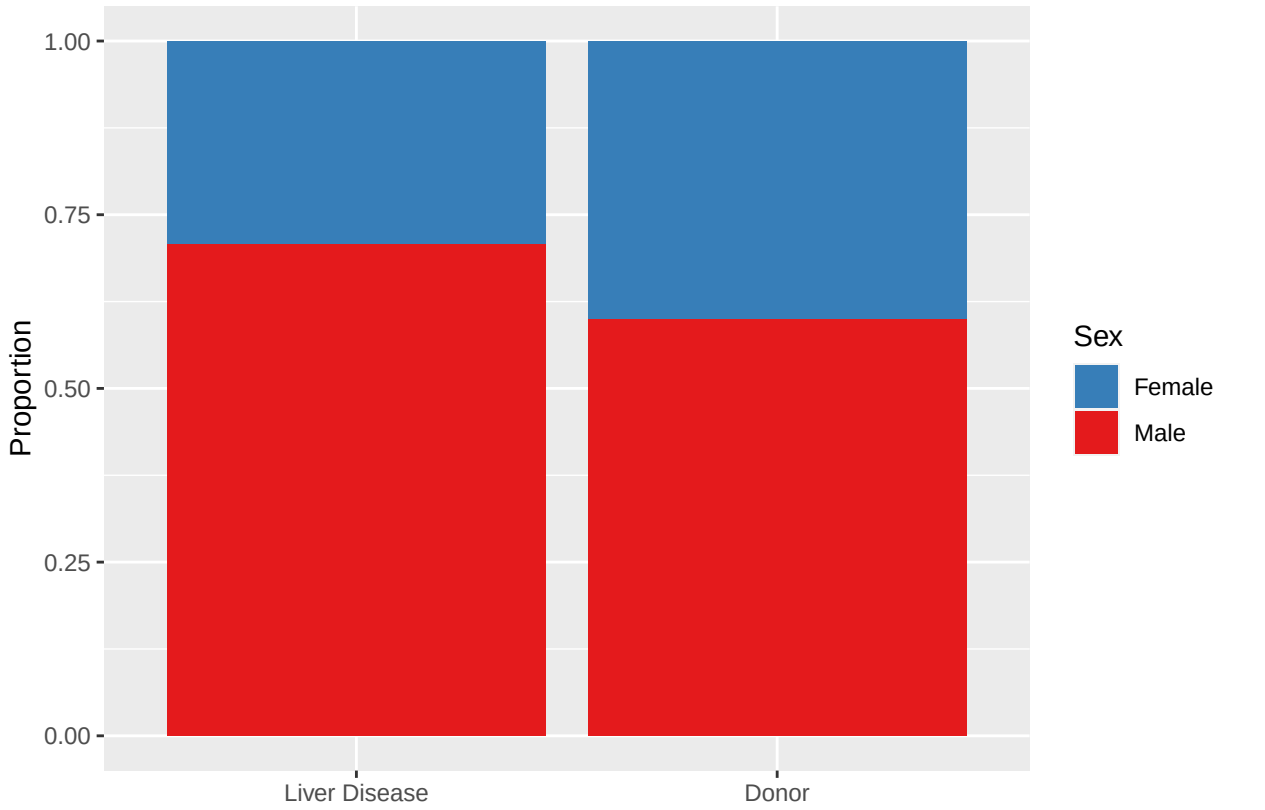
```r
# Swap columns
imp.data <- imp.data %>% relocate(Category, Sex)
```

Let us explore the relationship (if any) between the presence of liver disease (hepatitis C), and sex.

```r
# Split data into 'Donor' and 'Diseased'
data1 <- imp.data %>%
  mutate(Category = if_else(str_detect(Category, "Donor"), "Donor",
                            "Liver Disease")) %>%
  mutate(Category = factor(Category, levels = c("Liver Disease", "Donor"))) %>%
  relocate(Category, .before = Category)

# Create copy of data including the non-encoded categorical feature for use later
data2 <- data1 %>%
  mutate(Sex = if_else(str_detect(Sex, "1"), "Male",
                            "Female"))

# See how sex and liver disease are related
# The ratio of male to females is weighted more towards males in the patients
# with liver disease, as opposed to the donor category.
sex.num <- data2 %>% group_by(Category, Sex) %>% summarise(N = n())
```

```
## `summarise()` has grouped output by 'Category'. You can override using the
## `.groups` argument.
```

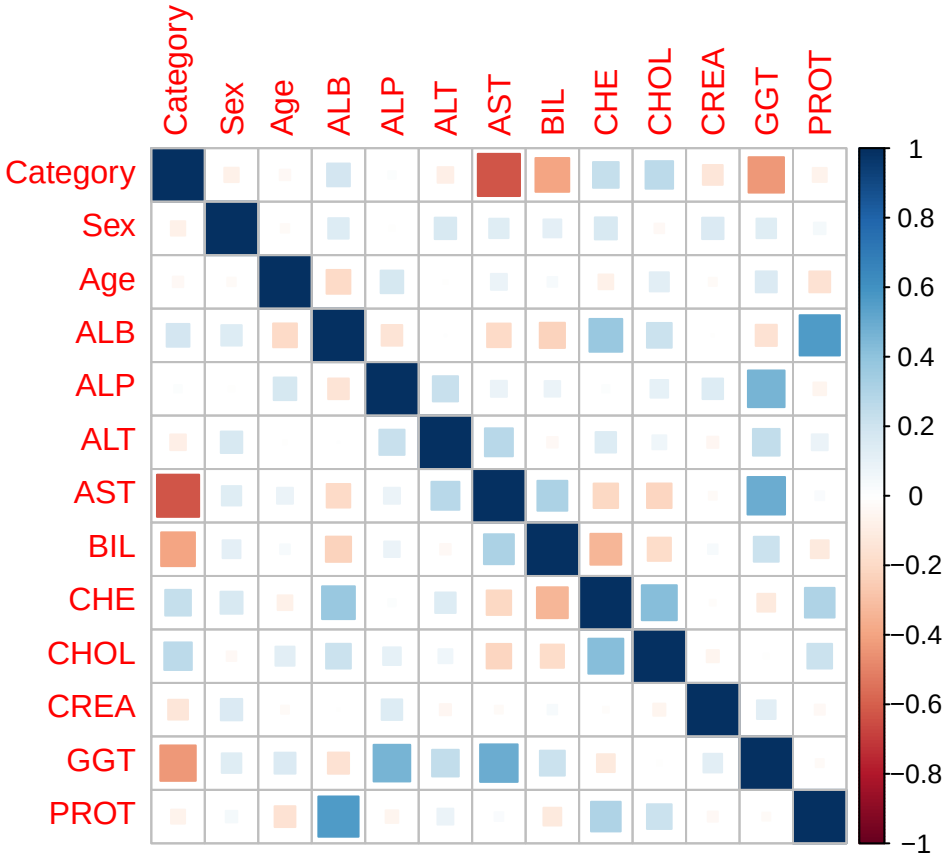
```r
ggplot(sex.num, aes(Category, N)) +
  geom_col(aes(fill = Sex), position = "fill") +
  scale_fill_brewer(palette = "Set1", direction = -1) +
  xlab("") +
  ylab("Proportion")
```



From the chart above, we can see that there is a higher proportion of liver disease among males than females. This supports data analysis on the subject, which found that 'HCV RNA positivity is significantly higher in males than females in adults.'[5]

Now, let us create a correlation plot of the data.

```
explore <- data1
explore$Category <- as.numeric(explore$Category)
cor_matrix <- cor(explore)
library(corrplot)
corrplot(cor_matrix, method="square", diag=TRUE)
```



We can see that there is a fairly strong negative correlation between AST, and Category (of disease). Then a slightly weaker, but somewhat marked correlation between BIL and Category (of disease), and GGT and Category.

Creation of training data, and test data. We shall explore the data more from the training set.

Next, we shall create a 'training' dataset, on which we train our classifiers, and a 'testing' dataset, on which we shall test the performance of our classifiers later. The 70% to 30% split of training data to test data allows us to maintain enough data for the classifiers to learn from, while also making sure all the categories are represented within the datasets. This split is stratified, because the target class is imbalanced. In this way, we have made sure that each of the subcategories, or classes, (blood donor, suspect blood donor, hepatitis, fibrosis, and cirrhosis) are represented in the split test and train datasets in proportions representative of their presence in the entire data. Furthermore, let us make note of the highly imbalanced nature of this dataset, with respect to the minority classes – something we shall return to later.

```
library(caret)
trainIndex <- createDataPartition(imp.data$Category, p = 0.7,
                                  list = FALSE,
                                  times = 1)
# Sub-setting into training data
train <- imp.data[ trainIndex,]

# Sub-setting into testing data
test <- imp.data[-trainIndex,]

# Let us inspect these new dataframes using frequency tables
```

```r
as.data.frame(table(train$Category))
```

```
##                      Var1 Freq
## 1          0=Blood Donor  374
## 2 0s=suspect Blood Donor    5
## 3            1=Hepatitis   17
## 4             2=Fibrosis   15
## 5            3=Cirrhosis   21
```

```r
as.data.frame(table(test$Category))
```

```
##                      Var1 Freq
## 1          0=Blood Donor  159
## 2 0s=suspect Blood Donor    2
## 3            1=Hepatitis    7
## 4             2=Fibrosis    6
## 5            3=Cirrhosis    9
```

Principal Component Analysis (PCA)

PCA is one the primary methods for use in multivariate techniques.[6] The main aim of PCA is to take high-dimensional datasets and reduce the dimensions (features) [7] – a process known as feature extraction, or feature reduction; the most important features are extracted, or the least important are removed. PCA attempts to find the fewest features that explain the maximum variance. PCA uses eigen-decomposition; eigenvectors and eigenvalues are used to convert a set of observations to a set of linearly uncorrelated variables – these are known as principal components (PCs); these PCs maximise variance in a monotonic decreasing sequence. I.e., the first PC explains the most variance, and then subsequent ones explain less, and so on. PCA is commonly used in clinical studies [8]. PCA has also the aim of aiding in the interpretation of data, while also trying to minimise any information loss.[9]

The use of PCA is only really appropriate when we have datasets of continuous variables. If the data have categorical features, the common idea is to replace these values with hot-encoded (often binary) dummy variables. From here, one could perform PCA on this changed data. This can result in bias, since the dummy or binary variables do not have distributions comparable to those of the continuous features.[10] To remedy this problem, we can use factor analysis of mixed data (FAMD). This method is similar to PCA, but it can data that contain both categorical and continuous features. FAMD uses a unique method of scaling in the multiple correspondence analysis (MCA) to equally balance the contributions of both categorical and continuous data, while it forms the principal components.[11]

Although PCA can be used on one-hot encoded binary data, that does not mean that it will be successful. There is a debate as to whether or not PCA should be used on binary categorical data (e.g., data the is one-hot encoded). We can certainly apply PCA to data that has features with binary values, but that does not mean we should. As we have noted before, PCA is designed to be implemented on data with continuous features. PCA is often naively employed, with little knowledge of better alternatives for mixed numerical and nominal data, such as FAMD. PCA tries to maximise variance (squared deviations), while minimising columns. When we use one-hot encoded categorical features, the notion of squared deviations breaks down. Although one may use these features in PCA, these results may not mean very much. For this reason, we shall remove the sex feature during the use of PCA. This is of course not ideal, since we are losing data, but we shall explore this method anyway for completion.

Now, we want to mean-centre, and normalise, the data for use in classification, as well as PCA. It makes sense to do this to the continuous features. This can help certain classifiers (such as SVM) to converge faster. We normalise and mean-centre the data after splitting the data into the train dataset and test dataset. This is because we want to avoid any information seeping from our untouched test dataset into the train dataset. Otherwise, we might have completely biased results at the end, which can lead to predictions that make the

performance of the classifier seem far better than it truly is. Note that, if we don't normalise the data, the model will be dominated by any features that have a much wider scale discrepancy – this can damage the model.

Mean centre and normalise the test and train data

```
train1 <- subset(train, select = -c(Category, Sex))
test1 <- subset(test, select = -c(Category, Sex))

# Mean-centre and normalise the 11 features, doing nothing to the target
# category, of course
preProcValues <- preProcess(train1, method = c("center", "scale"))

# Now, we transform the test data set using the same transformation parameters
# that we used on the training set -- this is important.
train.transformed <- predict(preProcValues, train1)
test.transformed <- predict(preProcValues, test1)

# Recombine into full scaled datasets
train.scaled <- cbind(subset(train, select = c(Category, Sex)), train.transformed)
test.scaled <- cbind(subset(test, select = c(Category, Sex)), test.transformed)
```

At this stage, it is useful for future applications – e.g. creating classifiers – to change the datatype in Category from character to factor.

Furthermore, we are interested in predicting if the patient is diseased or not – i.e., that they fall into one of two categories: 'Donor' or 'Liver Disease' – and so we shall split between 'Donor,' and 'Liver Disease.'

```
train2 <- train.scaled %>%
  mutate(Category = if_else(str_detect(Category, "Donor"), "Donor",
                            "Liver Disease")) %>%
  mutate(Category = factor(Category, levels = c("Liver Disease", "Donor"))) %>%
  relocate(Category, .before = Category)

test2 <- test.scaled %>%
  mutate(Category = if_else(str_detect(Category, "Donor"), "Donor",
                            "Liver Disease")) %>%
  mutate(Category = factor(Category, levels = c("Liver Disease", "Donor"))) %>%
  relocate(Category, .before = Category)

# Split our data into useful subsets. This is a common syntax.
x.train <- subset(train2, select = -Category)
y.train <- subset(train2, select = Category)
x.test <- subset(test2, select = -Category)
y.test <- subset(test2, select = Category)
```

Now, we perform PCA with scaled, and mean-centred, values

```
# We remove the target column, since we do not need this for PCA
data.pca <- subset(train2, select = -c(Category, Sex))
pca <- prcomp(data.pca)
pca$rotation
```

```
##              PC1         PC2         PC3         PC4          PC5          PC6
## Age   0.15233369  0.14870439  0.52602417  0.29854946  0.423633444  0.445860119
```

```
## ALB  -0.43671865  0.19676481 -0.27387272 -0.14688288  0.239713711  0.069814134
## ALP   0.24537503  0.43589140  0.27396719 -0.25400139 -0.096676737 -0.387480490
## ALT   0.09819969  0.42119825 -0.16318352  0.23555961 -0.593580265 -0.008785240
## AST   0.34596628  0.24057348 -0.41349625  0.17886702  0.129769342  0.409190902
## BIL   0.33299107 -0.04195284 -0.25611530 -0.02452510  0.469042369 -0.552088652
## CHE  -0.39694069  0.28493702  0.11968621  0.03311952 -0.140850546 -0.004036112
## CHOL -0.27757149  0.32657935  0.36398621  0.15602816  0.221773253 -0.273810223
## CREA  0.06049616  0.05011149  0.12947738 -0.83500165 -0.009747801  0.303658545
## GGT   0.34635433  0.46654836 -0.05001128 -0.10534637  0.093572316  0.064982290
## PROT -0.35695170  0.32265150 -0.37941397 -0.05409634  0.294404897  0.049576392
##              PC7         PC8         PC9         PC10         PC11
## Age  -0.05845786  0.43259165  0.13284593 -0.001830683 -0.048861589
## ALB   0.21716568  0.28622147  0.01896290  0.559252153  0.411626875
## ALP   0.38591176  0.14304910  0.16805148 -0.319361058  0.388720748
## ALT  -0.35370556  0.44926955 -0.18287106  0.121548970 -0.057010785
## AST  -0.17190214 -0.36317614  0.06519816 -0.183661472  0.485543310
## BIL  -0.43320259  0.20167103  0.21949039  0.121207941 -0.071796022
## CHE  -0.26980080 -0.26557026  0.75422121  0.018473213 -0.116022404
## CHOL -0.32647003 -0.36723117 -0.51731963  0.044224392  0.155875028
## CREA -0.41716999  0.05137104 -0.10270473 -0.016959162 -0.005264344
## GGT   0.31057643 -0.30645851 -0.04985801  0.422411089 -0.514297927
## PROT  0.09114674  0.18473046 -0.13556952 -0.583977176 -0.361422943
```

```
# The main source of inspiration for this code was found here:
# http://www.sthda.com/english/articles/31-principal-component-methods-in-r-practical-guide/112-pca-pri

# calculate total variance explained by each principal component
var.explained = pca$sdev^2 / sum(pca$sdev^2)

# Let us inspect the variance explained
# This should be monotonically decreasing -- and it is
var.explained
```

```
##  [1] 0.23541951 0.16048929 0.11869403 0.09996845 0.08599788 0.06748224
##  [7] 0.06344932 0.05411995 0.04998558 0.03565848 0.02873527
```

```
#install.packages(c("FactoMineR", "factoextra"))
library("FactoMineR")
library("factoextra")

# If we look at the cumulative proportion, we can see that 90% of the total
# variance is explained by the first 9 principal components.
summary(pca)
```

```
## Importance of components:
##                           PC1    PC2    PC3     PC4    PC5     PC6     PC7
## Standard deviation     1.6092 1.3287 1.1426 1.04864 0.9726 0.86157 0.83543
## Proportion of Variance 0.2354 0.1605 0.1187 0.09997 0.0860 0.06748 0.06345
## Cumulative Proportion  0.2354 0.3959 0.5146 0.61457 0.7006 0.76805 0.83150
##                           PC8     PC9    PC10    PC11
## Standard deviation     0.77157 0.74151 0.62629 0.56222
## Proportion of Variance 0.05412 0.04999 0.03566 0.02874
## Cumulative Proportion  0.88562 0.93561 0.97126 1.00000
```

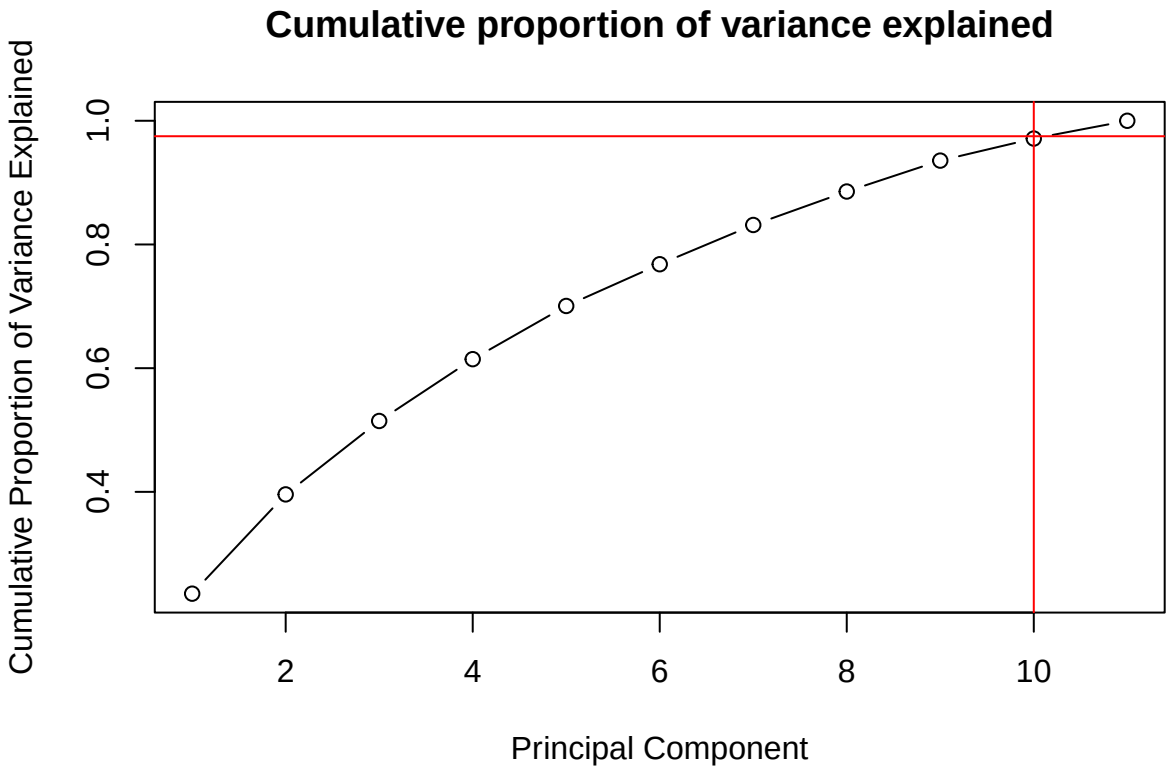
```
# We can also get this by the following:
cumsum(var.explained)
```

```
##  [1] 0.2354195 0.3959088 0.5146028 0.6145713 0.7005692 0.7680514 0.8315007
##  [8] 0.8856207 0.9356063 0.9712647 1.0000000
```

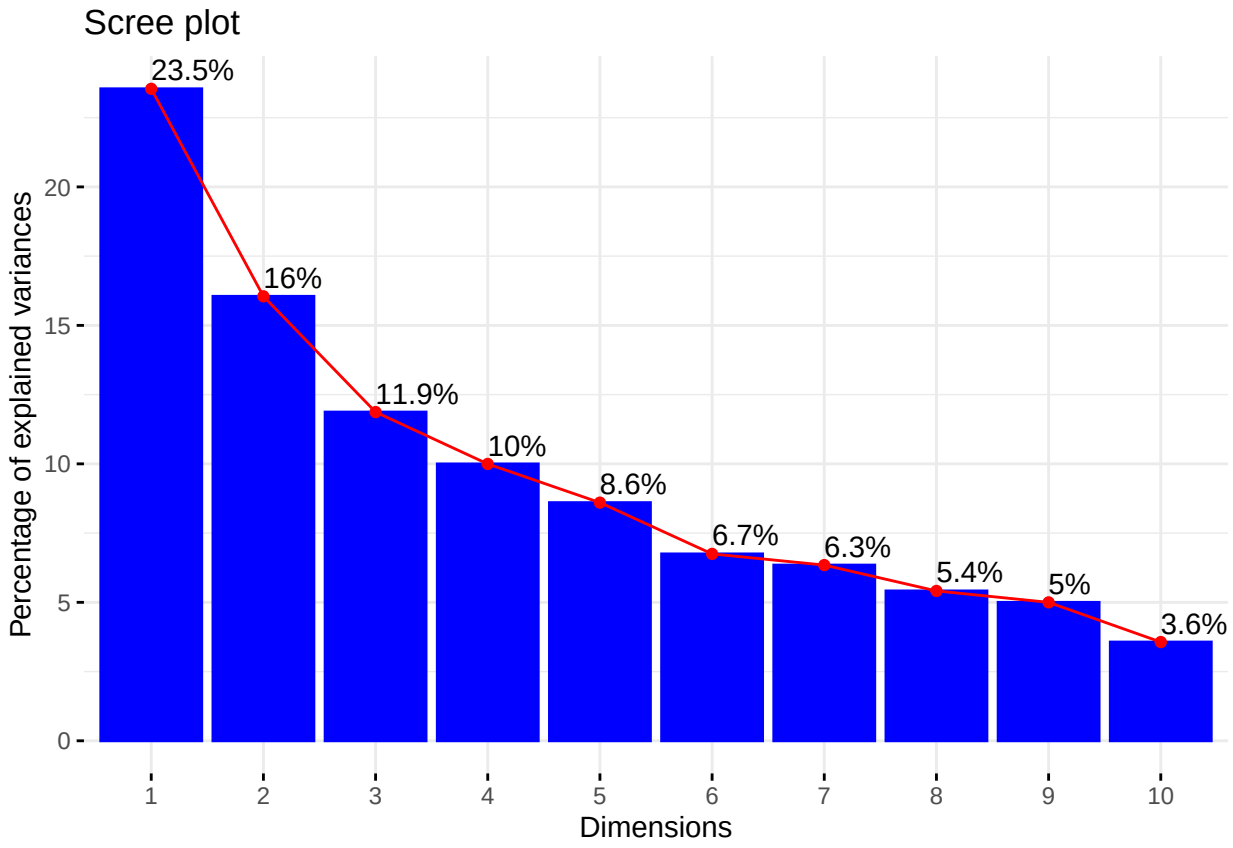
```
# We see the eigenvalues, as well as variance percent, and cumulative variance
# percent
eig.val <- get_eigenvalue(pca)
eig.val
```

```
##        eigenvalue variance.percent cumulative.variance.percent
## Dim.1   2.5896146        23.541951                    23.54195
## Dim.2   1.7653822        16.048929                    39.59088
## Dim.3   1.3056344        11.869403                    51.46028
## Dim.4   1.0996530         9.996845                    61.45713
## Dim.5   0.9459767         8.599788                    70.05692
## Dim.6   0.7423047         6.748224                    76.80514
## Dim.7   0.6979425         6.344932                    83.15007
## Dim.8   0.5953194         5.411995                    88.56207
## Dim.9   0.5498413         4.998558                    93.56063
## Dim.10  0.3922432         3.565848                    97.12647
## Dim.11  0.3160880         2.873527                   100.00000
```

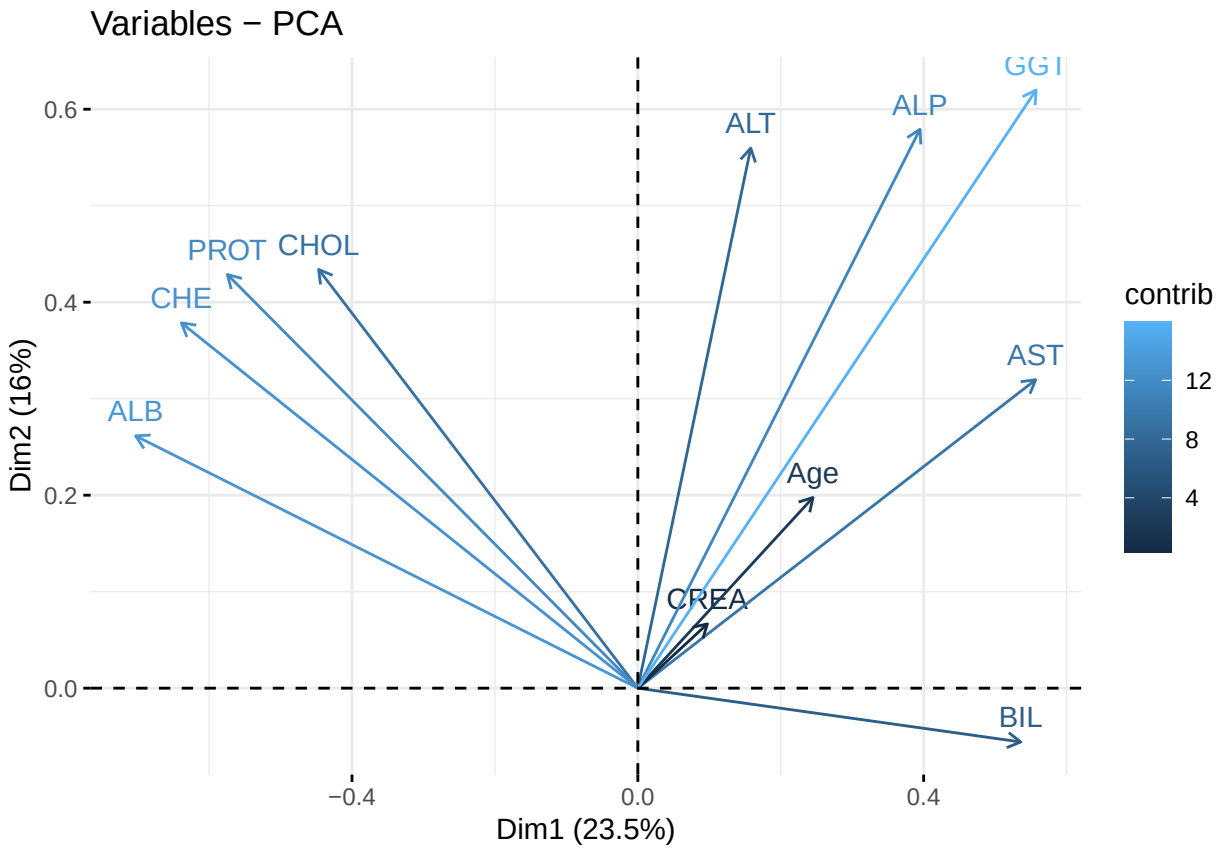
```
# Now, let us plot the cumulative proportion of variance explained

cumulative.plot.data <- cumsum(var.explained)

plot(cumulative.plot.data, xlab = "Principal Component",
     ylab = "Cumulative Proportion of Variance Explained",
     type = "b",
     main = "Cumulative proportion of variance explained")
abline(h = 0.975, col="red", v=10)
```



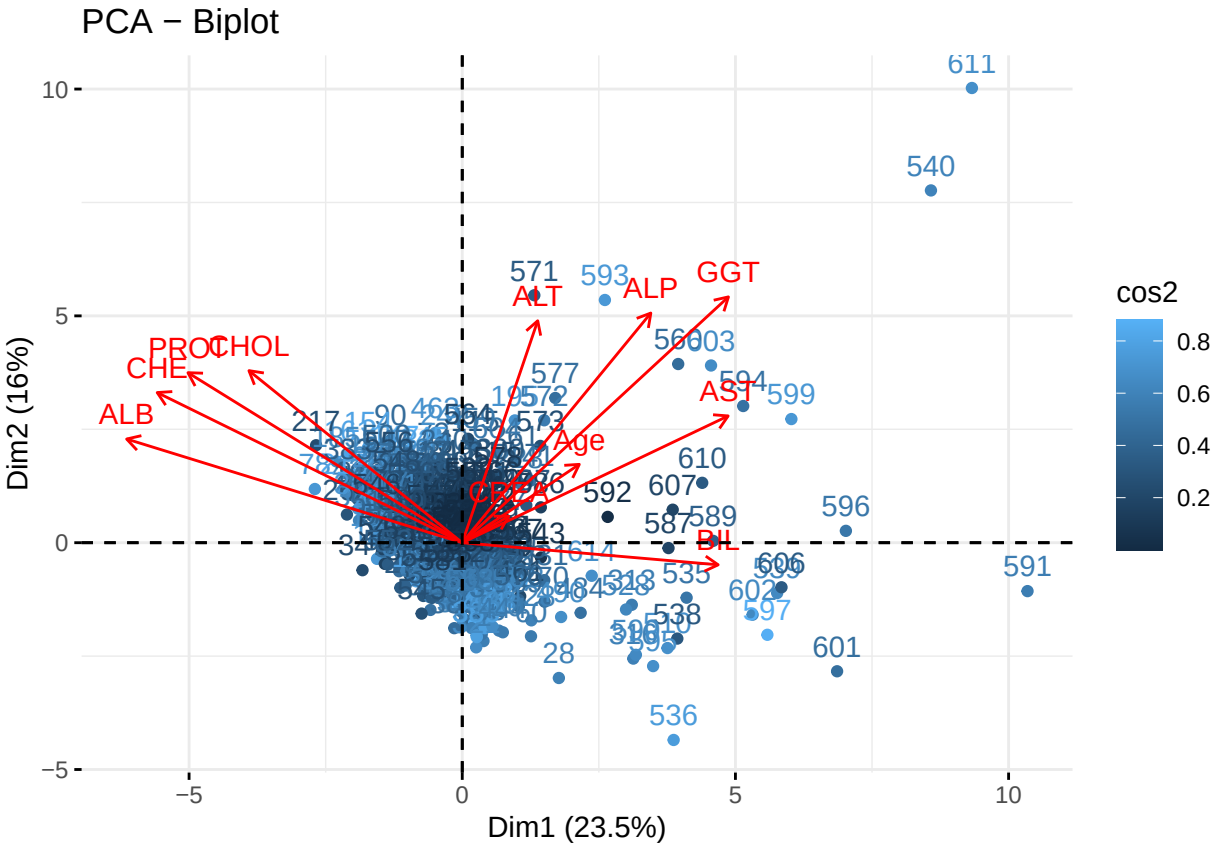
```r
# Plot graphs for visuals

# Screeplot
fviz_screeplot(pca, addlabels = TRUE, barfill = "blue", barcolor = "blue",
               linecolor = "red")
```




```r
# Correlation circle
fviz_pca_var(pca, col.var = "contrib")
```




```
# Biplot
fviz_pca_biplot(pca, col.ind="cos2", col.var = "red")
```



After performing PCA on the training set, we can see that ~97% of the variance is explained by 10/11 PCs. We can see this graphically in the plot of cumulative proportion of variance explained; the red horizontal line indicates 97.5% of variance explained – this is seen by PC10. This seems to indicate that PCA isn't very useful for these data as a method for dimension reduction. Note further that, after the first two principal components, the scree plot is fairly downwardly linear, with a slight overall concavity.

Looking at the correlation circle, and bi-plot, we can see the correlations of features with respect to the first to principal components. It is a helpful tool to see the importance of each explanatory variable. The direction of each arrow tell us if the correlation is positive or negative, and the lengths indicate the relative strength of the correlation.

Finding the elbow point

The elbow point refers to the point at which the variance explained starts to level out. To calculate the point at which the PCs begin to 'elbow,' we take the maximum of the following two values:

1. The point at which the PCs begin to explain only 5% of standard deviation, and the PCs up to that point explain 90% of the standard deviation.
2. The point at which the percentage change in variation between one PC and the next PC is below 0.1%.

It is worth noting that this 'elbow' method is subjective, and it can be unreliable. Often, the 'elbow' point is ambiguous, especially when the scree plot does not have a distinct elbow.[12] What it does seem to show is that PCA did not result in dimension reduction. In the scree plot above, we can see that there is no definitive point at which the variance explained starts to level out – it happens very gradually.

```
# Define the function from the source code:
# https://rdrr.io/bioc/PCAtools/src/R/findElbowPoint.R

# We use this because 'package "PCAtools" is not available for the most current
```

```
# version of R.

findElbowPoint <- function(variance) {
  if (is.unsorted(-variance)) {
    stop("'variance' should be sorted in decreasing order")
  }

  # Finding distance from each point on the curve to the diagonal.
  dy <- -diff(range(variance))
  dx <- length(variance) - 1
  l2 <- sqrt(dx^2 + dy^2)
  dx <- dx/l2
  dy <- dy/l2

  dy0 <- variance - variance[1]
  dx0 <- seq_along(variance) - 1

  parallel.l2 <- sqrt((dx0 * dx)^2 + (dy0 * dy)^2)
  normal.x <- dx0 - dx * parallel.l2
  normal.y <- dy0 - dy * parallel.l2
  normal.l2 <- sqrt(normal.x^2 + normal.y^2)

  # Picking the maximum normal that lies below the line.
  # If the entire curve is above the line, we just pick the last point.
  below.line <- normal.x < 0 & normal.y < 0
  if (!any(below.line)) {
      length(variance)
  } else {
      which(below.line)[which.max(normal.l2[below.line])]
  }
}

findElbowPoint(var.explained)
```

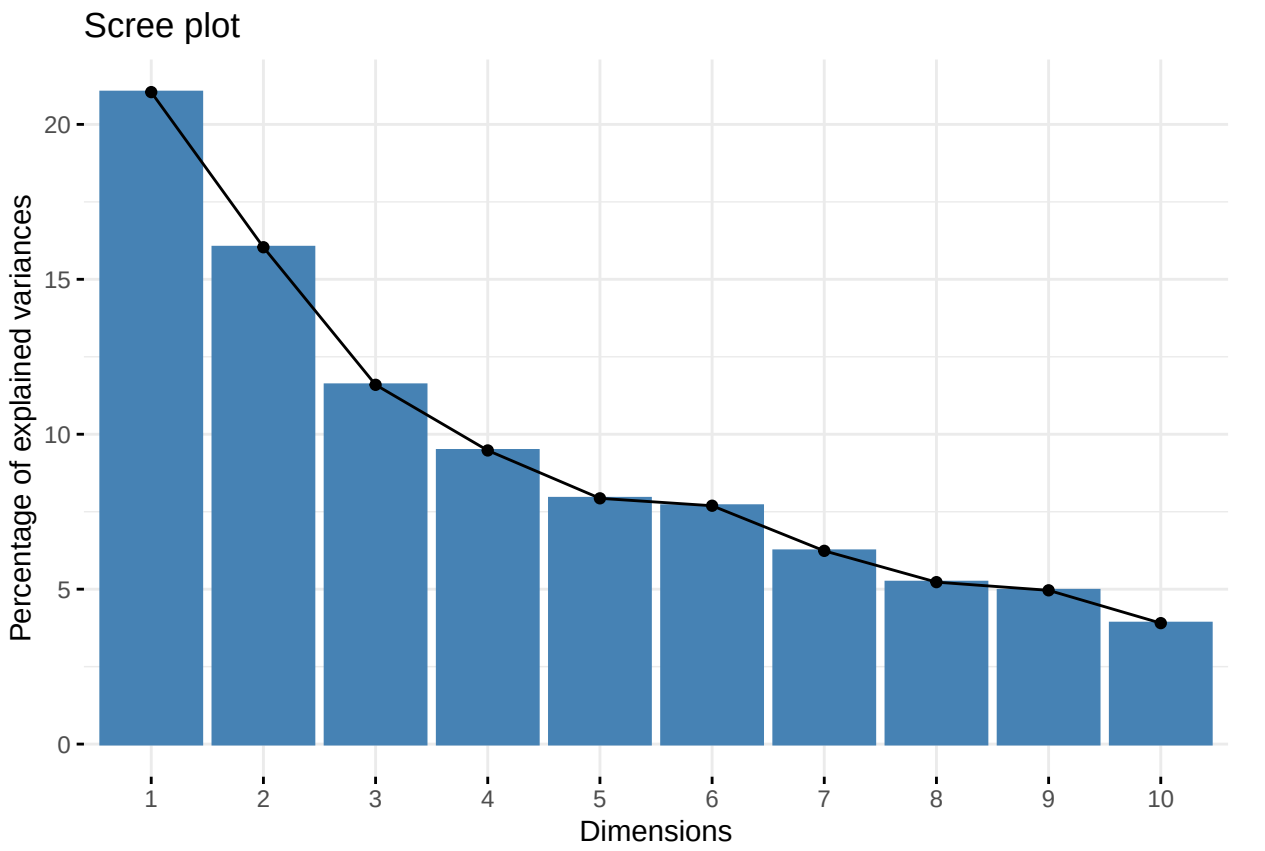
```
## [1] 11
```

Factor Analysis of Mixed Data (FAMD)

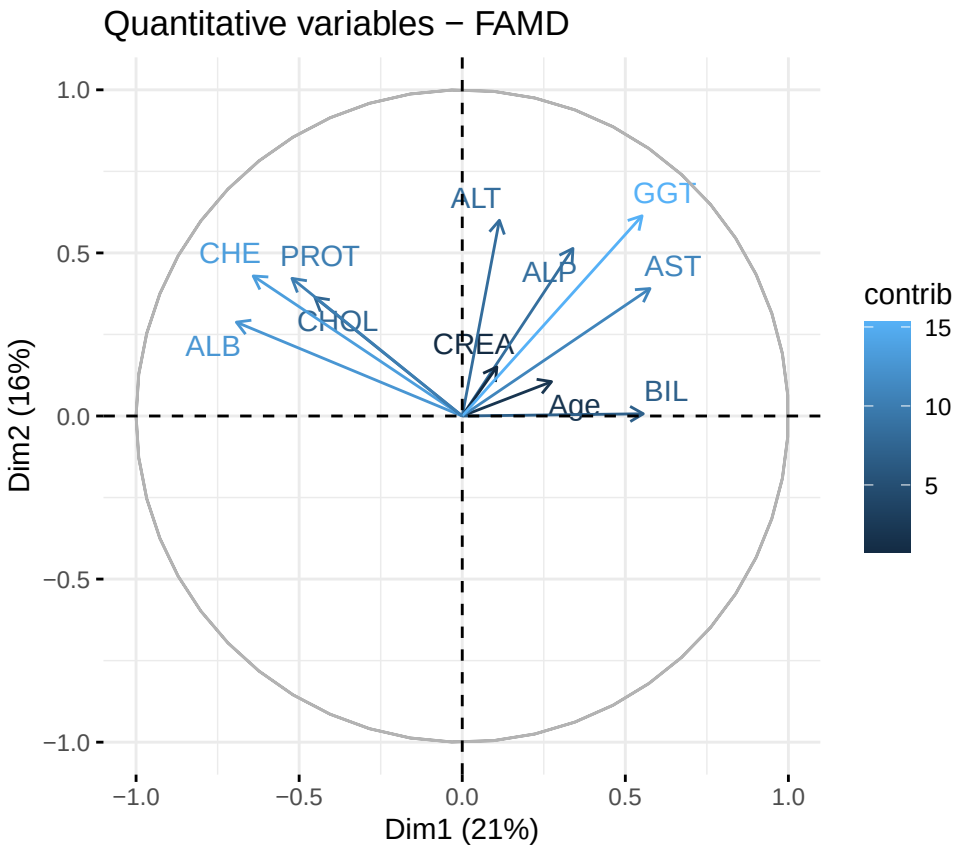
FAMD is a method of exploring data, and computing principal components, on data that consis of both categrical and continuous features. The continuous features are scaled so that their valiance is one, and the nominal features are transformed into a 'disjunctive data table.' These features are then scaled using the same method used in machine capability analysis (MCA). Let us implement FAMD.

```
# The main source of inspiration for this code was found here:
# http://www.sthda.com/english/articles/31-principal-component-methods-in-r-practical-guide/115-famd-fa

library("FactoMineR")

data.famd <- subset(data2, select = -c(Category))

# The dataset data.famd includes the categorical data that hasn't been hot,
# encoded, minus the target variable.

res.famd <- FAMD(data.famd,
```

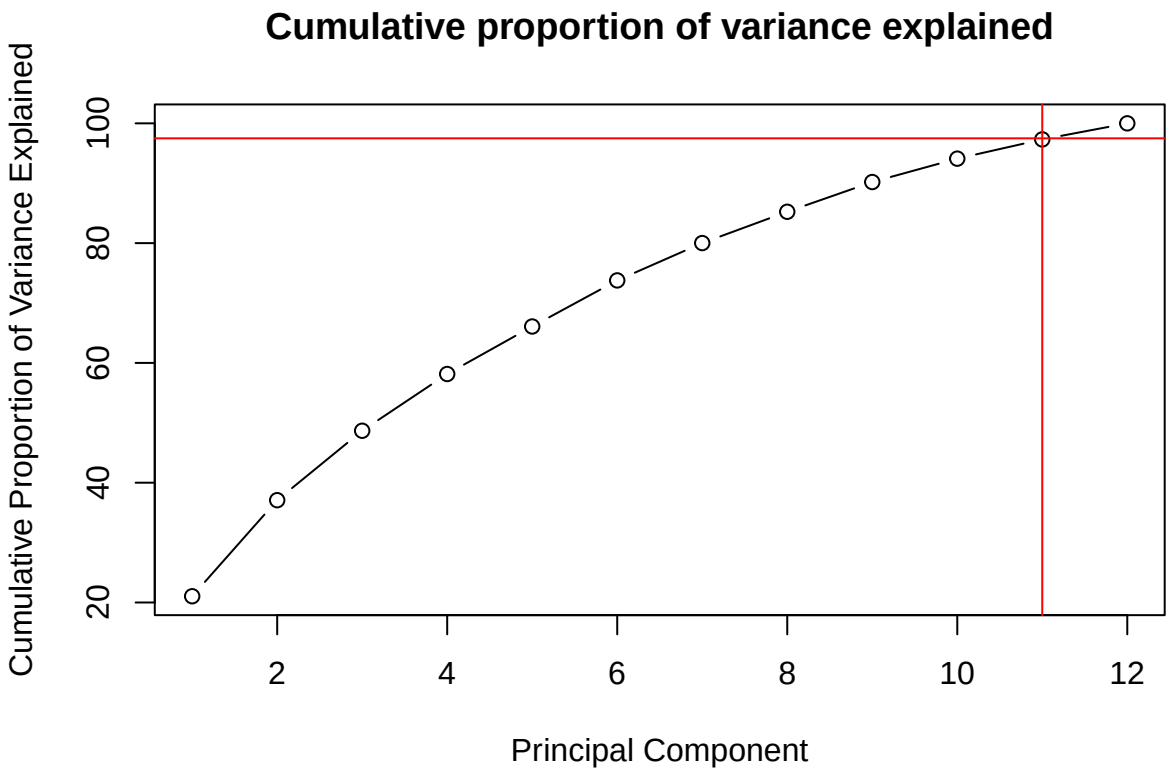
```
                graph = FALSE,
                sup.var = NULL,
                ncp=12)

eig.val <- get_eigenvalue(res.famd)

fviz_screeplot(res.famd)
```




```
var <- get_famd_var(res.famd)

fviz_famd_var(res.famd, "quanti.var", repel = TRUE,
              col.var = "contrib")
```




```r
# Variance explained
cumulative.plot.data1 <- eig.val[,3]

plot(cumulative.plot.data1, xlab = "Principal Component",
     ylab = "Cumulative Proportion of Variance Explained",
     type = "b",
     main = "Cumulative proportion of variance explained")
abline(h = 97.5, col="red", v=11)
```




```
# Calculate elbow point
# Cumulative variance explained must be sorted in decreasing order
findElbowPoint(rev(eig.val[,3]))
```

```
## [1] 12
```

Note that from the scree plot, and the plot of 'cumulative proportion of variance explained,' there seems to be not much possibility of feature reduction again. The elbow point is calculated as 12 – this further indicates that we shouldn't employ feature reduction.

Looking at the correlation circle, we can see the correlations of features with respect to the first to principal components; note that these arrows are generally in similar directions to the ones found in PCA. It is a helpful tool to see the importance of each explanatory feature.

Models

The two models that will be used for classification are random forest (RF), and support vector machine (SVM).

Performance metrics

Some definitions:

1. True positive (TP): Result where the test correctly predicts the presence of disease.
2. True negative (TN): Result where the test correctly predicts the absence of disease.
3. False positive (FP): Also known as Type I error, this is a result where a test incorrectly predicts the presence of disease.
4. False negative (FN): Also known as Type II error, this is a results where a test incorrectly predicts the absence of disease.

The metrics that will be used to assess the performance of the models will be accuracy, sensitivity, and specificity – these are all related to the values TP, TN, FP, and FN.

The *accuracy* is simply a total measure of how many observations, or instances of positive and negative results, were correctly predicted by the classifier.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Sensitivity (or the true positive rate) is a measure of a classifier's ability to predict that a patient who truly has the disease is 'positive.' It takes values between 0 and 1. The higher the value of sensitivity, the fewer instances of false negative results are predicted by the classifier. This means that the classifier is missing fewer instances of the disease among the population. Sensitivity is given by the following equation. A low value is not good, and it means the classifier is failing to pick up disease in truly diseased patients.

$$Sensitivity = \frac{TP}{TP + FN}$$

Specificity (or the true negative rate) is a measure of a classifier's ability to predict that a patient who truly does not have the disease is 'negative.' The higher the value of specificity, the fewer instances of false positive results are predicted by the classifier. A test (or classifier) with a low value of specificity is quite bad for diagnostic purposes; in such tests, many of the predictions will result in false positive – i.e., they do not have the disease, but the test classifies them as positive, and they might subsequently undergo treatment, or further diagnostic tests, that are completely unnecessary.

$$Specificity = \frac{TN}{TN + FP}$$

Ideally, in our classifier, we should like there to be simultaneously high values of sensitivity and specificity; however, there is often an asymmetry, or trade-off, between these two values. When a classifier has a high sensitivity, it is less likely to predict false negatives. In the general population, not all patients fall into strict categories, such as positive or negative. Sometimes, diseases are more nuanced, and some patients may lie within, as it were, a grey-area in between the two binary categories. Therefore, a decision must be made as to how much weight one puts on either one of the values when creating a suitable test.

Random Forest (RF)

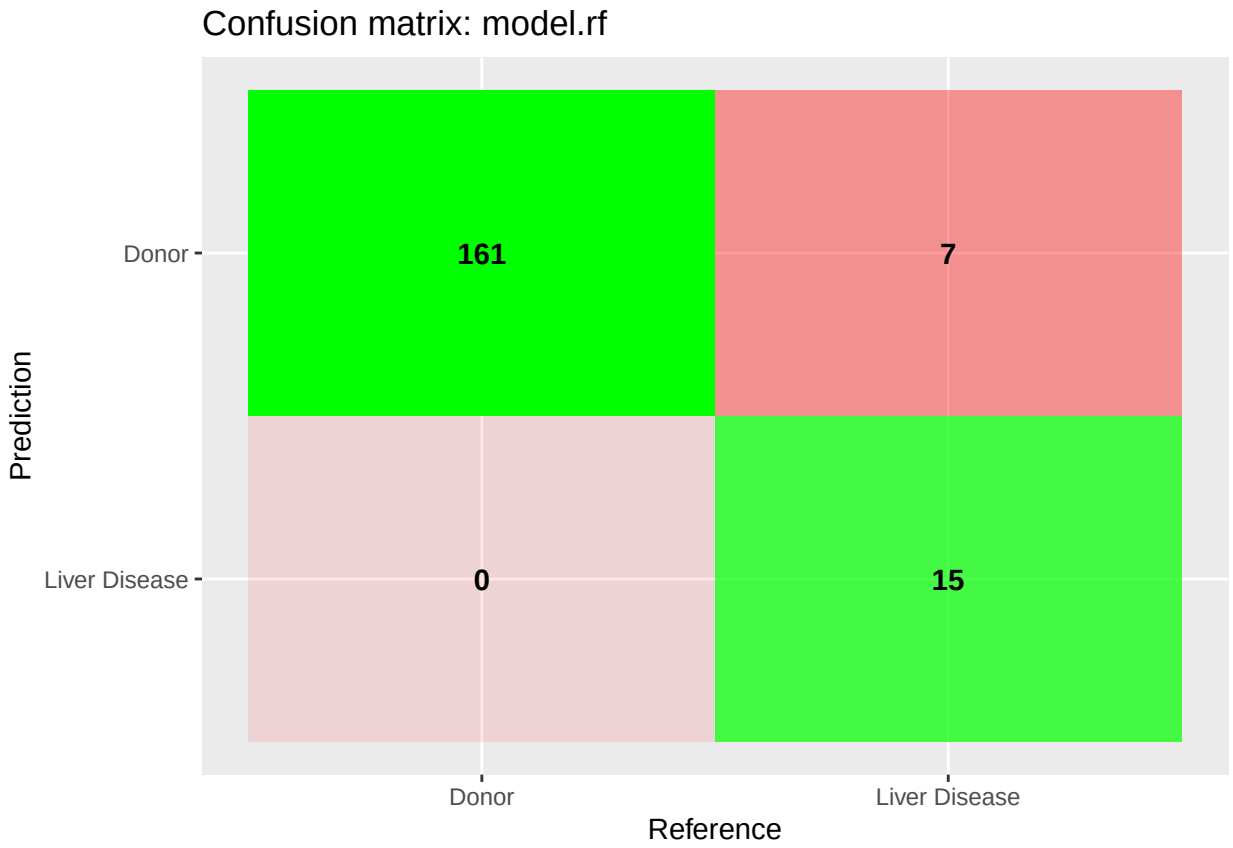
A random forest (RF) classifier involves a combination of many single classification trees. Each classifier is created using an independently-sampled random vector. Each individual tree gives a unit vote indicating the class which is most popular, and this is used to classify an input vector. Random forests are commonly used classifiers in medical fields for disease detection [13].

```
# Create a random forest classifier that uses cross-validation

train.data <- train2
control <- trainControl(method="repeatedcv", number=10, repeats=3)
model.rf <- train(Category~., data=train.data, method="rf", metric="Accuracy", trControl=control)

# Test the model on unseen (scaled) test data
pred_test <- predict(model.rf, x.test)
cm <- confusionMatrix(pred_test, y.test$Category)
cm
```

```
## Confusion Matrix and Statistics
##
##                Reference
```

```
## Prediction      Liver Disease Donor
##   Liver Disease            15     0
##   Donor                     7   161
##
##                Accuracy : 0.9617
##                  95% CI : (0.9228, 0.9845)
##     No Information Rate : 0.8798
##     P-Value [Acc > NIR] : 9.848e-05
##
##                   Kappa : 0.7904
##
##  Mcnemar's Test P-Value : 0.02334
##
##             Sensitivity : 0.68182
##             Specificity : 1.00000
##          Pos Pred Value : 1.00000
##          Neg Pred Value : 0.95833
##              Prevalence : 0.12022
##          Detection Rate : 0.08197
##    Detection Prevalence : 0.08197
##       Balanced Accuracy : 0.84091
##
##        'Positive' Class : Liver Disease
##
```

```r
# Link to code useful for plotting confusion matrix
# https://stackoverflow.com/questions/37897252/plot-confusion-matrix-in-r-using-ggplot

library(dplyr)
table <- data.frame(cm$table)
plot <- table %>%
  mutate(goodbad = ifelse(table$Prediction == table$Reference, "good", "bad")) %>%
  group_by(Reference) %>%
  mutate(prop = Freq/sum(Freq))

# fill alpha relative to sensitivity/specificity by proportional outcomes within reference groups (see
ggplot(data = plot, mapping = aes(x = Reference, y = Prediction, fill = goodbad, alpha = prop)) +
  geom_tile() +
  ggtitle("Confusion matrix: model.rf") +
  geom_text(aes(label = Freq), vjust = .5, fontface  = "bold", alpha = 1) +
  scale_fill_manual(values = c(good = "green", bad = "red")) +
  theme(legend.position="none") +
  xlim(rev(levels(table$Reference)))
```



The results show that there is an accuracy of 96.2%, a sensitivity of 0.682, and a specificity of 1.00.

Tuning/Hyper-parameter Optimisation

Let us try to tune the hyper-parameter, and see if we can do any better. The hyper-parameter we are trying to optimise is 'mtry.' It is defined as 'the number of randomly drawn candidate variables out of which each split is selected when growing a tree.'[14] Smaller values of this parameter create more varied, and uncorrelated, trees. Such forests often pay more attention to features that have a smaller correlation with the target variable. These features are often superseded by those with a stronger correlation, had they been candidates out of which the split is selected. A drawback is that these trees often perform worse on average – variables that might not be useful are chosen instead.

As a default, in classification, the mtry value is usually put to be the square root of the number of feature columns.[15] However, better values can be found using trials. The mtry is bounded by the number of features, since it determines the number of candidates that are chosen at random at each stage of RF iteration.

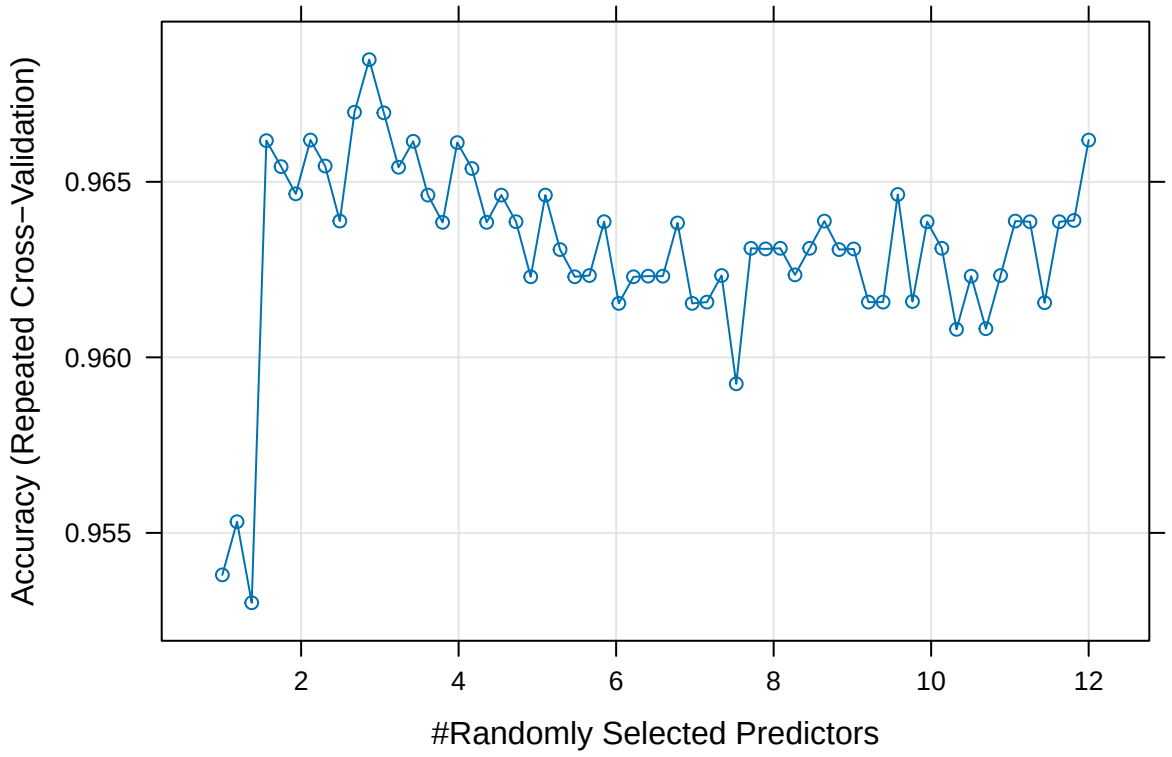
```
# Hyper-parameter optimisation

tunegrid <- expand.grid(.mtry=seq(1, 12, length = 60))
model.rf2 <- train(Category~., data=train.data, method="rf", metric="Accuracy",
                   tuneGrid=tunegrid, trControl=control)
model.rf2
```

```
## Random Forest
##
## 432 samples
##  12 predictor
##   2 classes: 'Liver Disease', 'Donor'
##
```

```
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 389, 389, 389, 389, 388, 388, ...
## Resampling results across tuning parameters:
##
##   mtry       Accuracy   Kappa
##    1.000000  0.9538038  0.7306141
##    1.186441  0.9553181  0.7371828
##    1.372881  0.9530102  0.7232576
##    1.559322  0.9661725  0.8149165
##    1.745763  0.9654326  0.8118573
##    1.932203  0.9646574  0.8085949
##    2.118644  0.9661901  0.8176176
##    2.305085  0.9654502  0.8160090
##    2.491525  0.9638822  0.8035960
##    2.677966  0.9669830  0.8271423
##    2.864407  0.9684805  0.8375860
##    3.050847  0.9669645  0.8287156
##    3.237288  0.9654149  0.8231239
##    3.423729  0.9661541  0.8226245
##    3.610169  0.9646213  0.8230107
##    3.796610  0.9638453  0.8215223
##    3.983051  0.9661180  0.8287929
##    4.169492  0.9653789  0.8255062
##    4.355932  0.9638453  0.8161671
##    4.542373  0.9646213  0.8248727
##    4.728814  0.9638637  0.8200371
##    4.915254  0.9622949  0.8162086
##    5.101695  0.9646213  0.8248727
##    5.288136  0.9630701  0.8184736
##    5.474576  0.9622949  0.8104316
##    5.661017  0.9623310  0.8156102
##    5.847458  0.9638637  0.8215408
##    6.033898  0.9615373  0.8109791
##    6.220339  0.9622949  0.8162086
##    6.406780  0.9623125  0.8144497
##    6.593220  0.9623125  0.8139361
##    6.779661  0.9638276  0.8208506
##    6.966102  0.9615373  0.8121847
##    7.152542  0.9615734  0.8125663
##    7.338983  0.9623310  0.8144620
##    7.525424  0.9592470  0.8042602
##    7.711864  0.9631061  0.8200669
##    7.898305  0.9630885  0.8178320
##    8.084746  0.9631061  0.8189531
##    8.271186  0.9623486  0.8159795
##    8.457627  0.9631061  0.8185927
##    8.644068  0.9638813  0.8216185
##    8.830508  0.9630701  0.8181407
##    9.016949  0.9630877  0.8214639
##    9.203390  0.9615734  0.8132264
##    9.389831  0.9615734  0.8140181
##    9.576271  0.9646389  0.8256370
##    9.762712  0.9615910  0.8137084
```

```
##    9.949153  0.9638629  0.8250534
##   10.135593  0.9631061  0.8201412
##   10.322034  0.9607982  0.8091996
##   10.508475  0.9623125  0.8162490
##   10.694915  0.9608158  0.8121863
##   10.881356  0.9623310  0.8178599
##   11.067797  0.9638813  0.8227403
##   11.254237  0.9638629  0.8223236
##   11.440678  0.9615549  0.8147482
##   11.627119  0.9638637  0.8244487
##   11.813559  0.9638990  0.8235600
##   12.000000  0.9661893  0.8329618
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.864407.
```

```r
# Plot optimised model
plot(model.rf2)
```

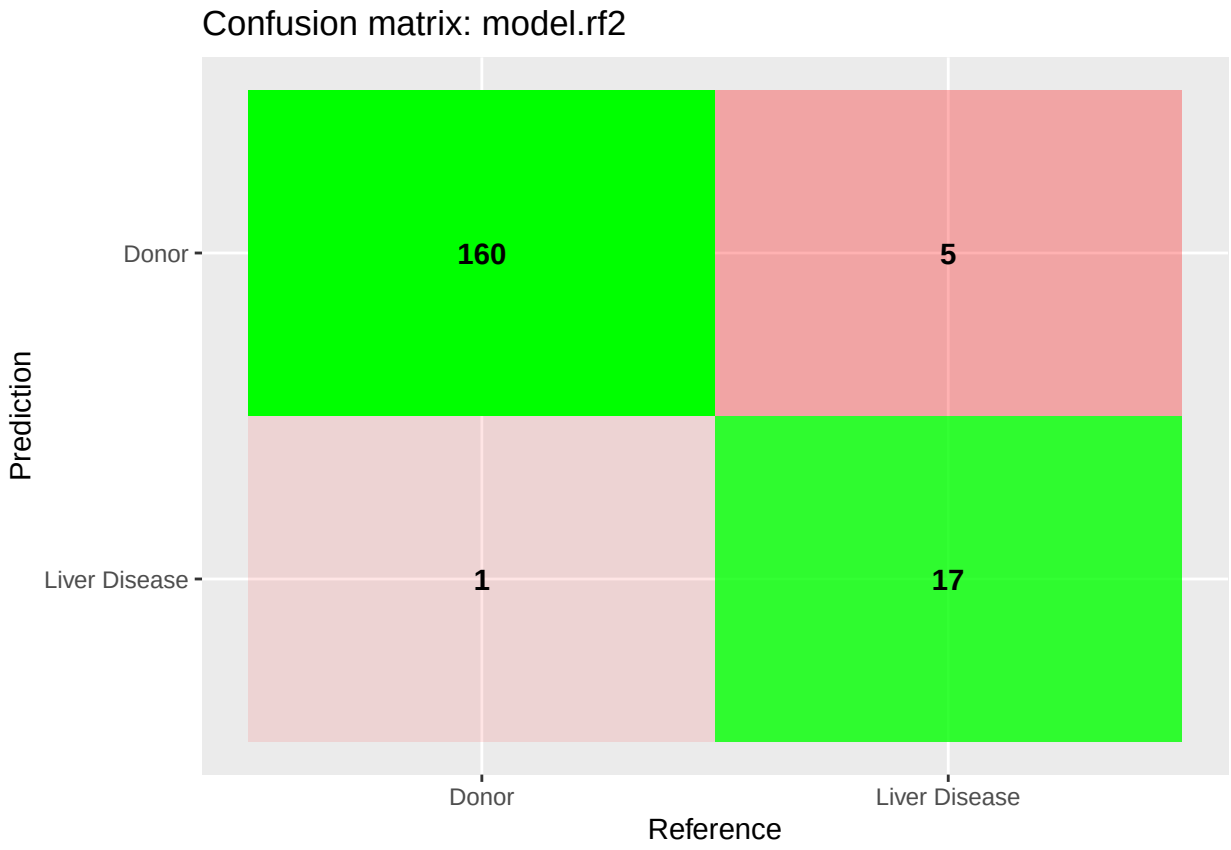



```r
# Check the performance of the model
pred_test0 <- predict(model.rf2, x.test)
cm0 <- confusionMatrix(pred_test0, y.test$Category)
cm0
```

```
## Confusion Matrix and Statistics
##
##                Reference
## Prediction      Liver Disease Donor
##   Liver Disease            17     1
##   Donor                     5   160
##
```

```
##                Accuracy : 0.9672
##                  95% CI : (0.93, 0.9879)
##     No Information Rate : 0.8798
##     P-Value [Acc > NIR] : 2.702e-05
##
##                   Kappa : 0.8318
##
##  Mcnemar's Test P-Value : 0.2207
##
##             Sensitivity : 0.77273
##             Specificity : 0.99379
##          Pos Pred Value : 0.94444
##          Neg Pred Value : 0.96970
##              Prevalence : 0.12022
##          Detection Rate : 0.09290
##    Detection Prevalence : 0.09836
##       Balanced Accuracy : 0.88326
##
##        'Positive' Class : Liver Disease
##
```

```r
# Plot confusion matrix
table <- data.frame(cm0$table)
plot <- table %>%
  mutate(goodbad = ifelse(table$Prediction == table$Reference, "good", "bad")) %>%
  group_by(Reference) %>%
  mutate(prop = Freq/sum(Freq))

# fill alpha relative to sensitivity/specificity by proportional outcomes within reference groups (see
ggplot(data = plot, mapping = aes(x = Reference, y = Prediction, fill = goodbad, alpha = prop)) +
  geom_tile() +
  ggtitle("Confusion matrix: model.rf2") +
  geom_text(aes(label = Freq), vjust = .5, fontface  = "bold", alpha = 1) +
  scale_fill_manual(values = c(good = "green", bad = "red")) +
  theme(legend.position="none") +
  xlim(rev(levels(table$Reference)))
```



Support vector machine (SVM)

SVM is a machine learning classification algorithm. It works by transforming the data in an optimal way such that these transformations determine boundaries within data, and between clusters of data based on already-defined features, labels, and target features. SVM is used in fields such as healthcare [16], as well as fields involving image classification. SVM algorithms map data to a feature space with increased dimension in order to categorise the data. This is done even when the data are not explicitly linearly separable. When such a boundary is found between the features, the algorithm transforms the data so that the boundary can be represented as a hyperplane. Next, the features in this new data are responsible for giving predictions, such as in which group new data should belong.

```
train.control <- trainControl(method="repeatedcv", number=10, repeats=3)
# Fit the model
svm1 <- train(Category ~., data = train.data, method = "svmLinear", trControl = train.control) # ,  pre
#View the model
svm1
```

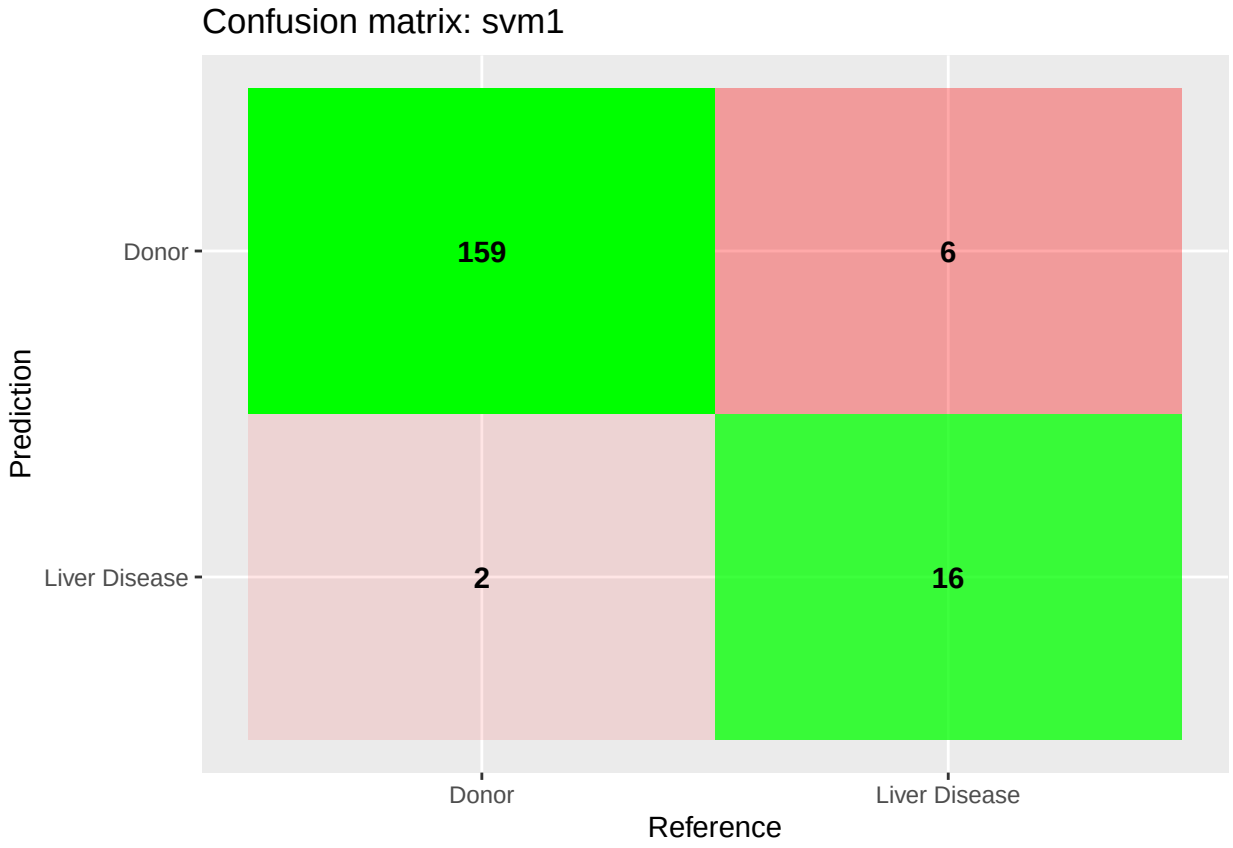
```
## Support Vector Machines with Linear Kernel
##
## 432 samples
##  12 predictor
##   2 classes: 'Liver Disease', 'Donor'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 389, 389, 389, 389, 388, 388, ...
## Resampling results:
##
```

```
##   Accuracy   Kappa
##   0.9575028  0.7792655
##
## Tuning parameter 'C' was held constant at a value of 1
```

```
# Check the performance of the model
pred_test1 <- predict(svm1, x.test)
cm1 <- confusionMatrix(pred_test1, y.test$Category)
cm1
```

```
## Confusion Matrix and Statistics
##
##                Reference
## Prediction      Liver Disease Donor
##   Liver Disease            16     2
##   Donor                     6   159
##
##                Accuracy : 0.9563
##                  95% CI : (0.9157, 0.9809)
##     No Information Rate : 0.8798
##     P-Value [Acc > NIR] : 0.0003133
##
##                   Kappa : 0.7757
##
##  Mcnemar's Test P-Value : 0.2888444
##
##             Sensitivity : 0.72727
##             Specificity : 0.98758
##          Pos Pred Value : 0.88889
##          Neg Pred Value : 0.96364
##              Prevalence : 0.12022
##          Detection Rate : 0.08743
##    Detection Prevalence : 0.09836
##       Balanced Accuracy : 0.85743
##
##        'Positive' Class : Liver Disease
##
```

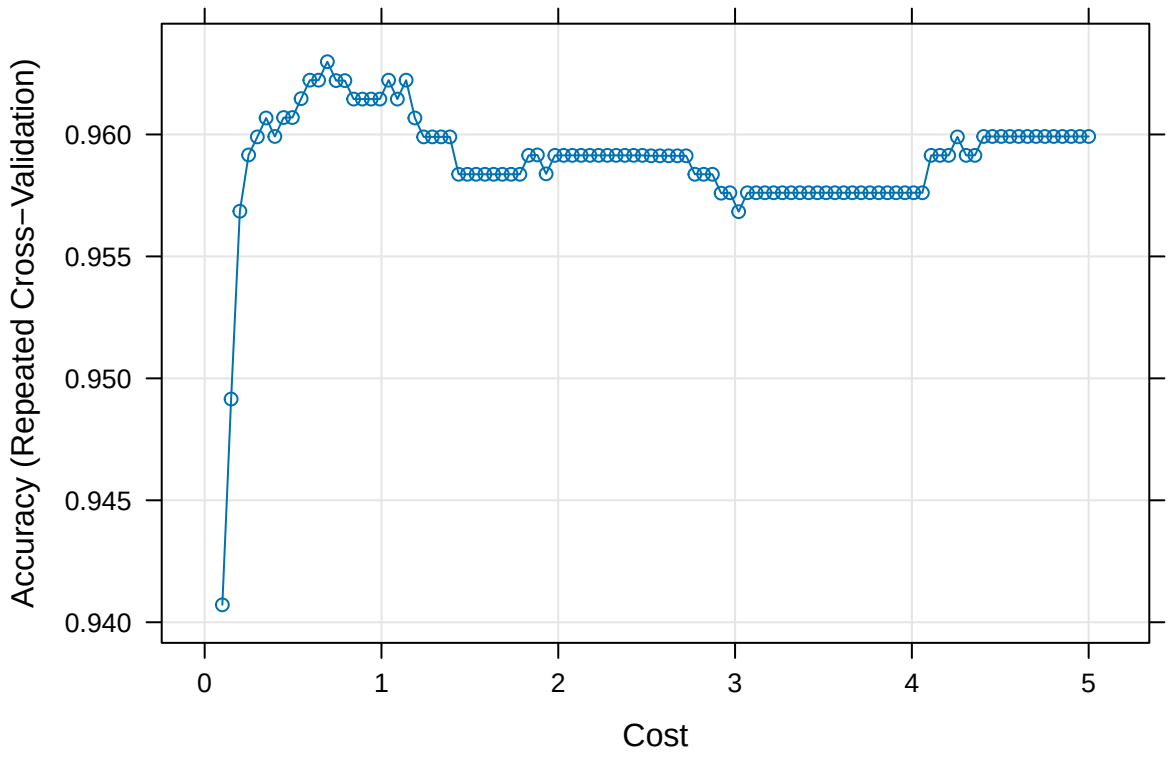
```
# Plot confusion matrix
table <- data.frame(cm1$table)
plot <- table %>%
  mutate(goodbad = ifelse(table$Prediction == table$Reference, "good", "bad")) %>%
  group_by(Reference) %>%
  mutate(prop = Freq/sum(Freq))

# fill alpha relative to sensitivity/specificity by proportional outcomes within reference groups (see
ggplot(data = plot, mapping = aes(x = Reference, y = Prediction, fill = goodbad, alpha = prop)) +
  geom_tile() +
  ggtitle("Confusion matrix: svm1") +
  geom_text(aes(label = Freq), vjust = .5, fontface  = "bold", alpha = 1) +
  scale_fill_manual(values = c(good = "green", bad = "red")) +
  theme(legend.position="none") +
  xlim(rev(levels(table$Reference)))
```



Tuning/hyper-parameter optimisation

The hyper-parameter 'C,' in SVM, is often referred to as the 'cost' of miss-classifying. A high value would indicate that we wish the model to greatly avoid miss-classification in training. You can imagine that the flexible kernel will bend around more values – this can result in over-fitting, rendering a classifier useless on new unseen data. Therefore, there is a trade-off between high and low values with such a parameter; the classifier needs to learn where to place the hyper-plane boundary so as to maximise this trade-off. Note that a very small cost could result in the miss-classification of data that is linearly separable.

```
# Hyperparameter optimisation
svm2 <- train(Category ~., data = train.data, method = "svmLinear",
              trControl = train.control,
              tuneGrid = expand.grid(C = seq(0.1, 5, length = 100)))

plot(svm2)
```




```
svm2$bestTune
```

```
##            C
## 13 0.6939394
```

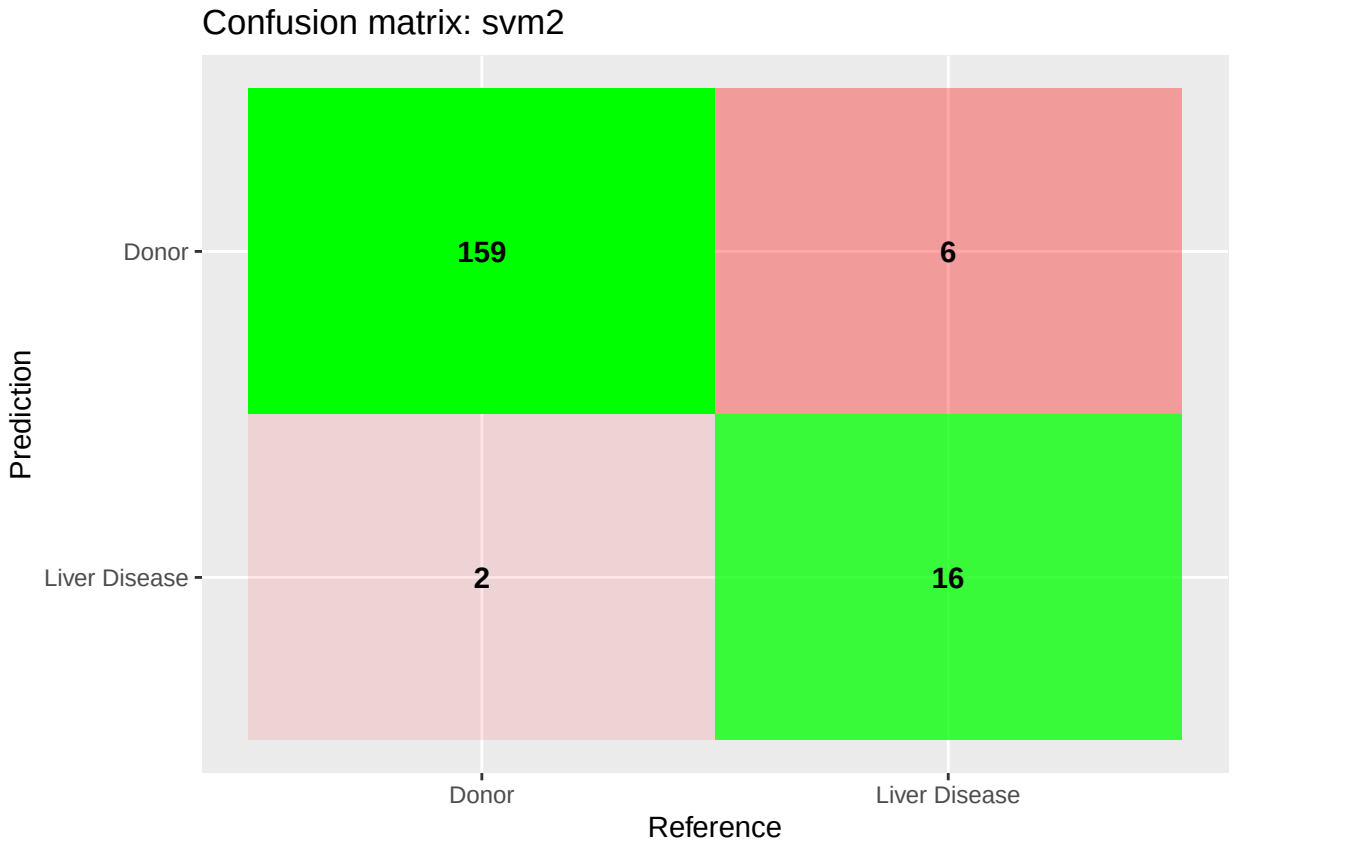
We can see that as the 'C' value initially increases, it causes the accuracy of the classifier to increase a considerable amount until we start to see diminishing returns – this is the point where accuracy of the classifier begins to plateau.

```
# Check the performance of the model
pred_test2 <- predict(svm2, x.test)
cm2 <- confusionMatrix(pred_test2, y.test$Category)
cm2
```

```
## Confusion Matrix and Statistics
##
##                Reference
## Prediction      Liver Disease Donor
##   Liver Disease            16     2
##   Donor                     6   159
##
##                Accuracy : 0.9563
##                  95% CI : (0.9157, 0.9809)
##     No Information Rate : 0.8798
##     P-Value [Acc > NIR] : 0.0003133
##
##                   Kappa : 0.7757
##
##  Mcnemar's Test P-Value : 0.2888444
##
##             Sensitivity : 0.72727
##             Specificity : 0.98758
```

```
##          Pos Pred Value : 0.88889
##          Neg Pred Value : 0.96364
##              Prevalence : 0.12022
##          Detection Rate : 0.08743
##    Detection Prevalence : 0.09836
##       Balanced Accuracy : 0.85743
##
##        'Positive' Class : Liver Disease
##
```

```r
# Plot confusion matrix
table <- data.frame(cm2$table)
plot <- table %>%
  mutate(goodbad = ifelse(table$Prediction == table$Reference, "good", "bad")) %>%
  group_by(Reference) %>%
  mutate(prop = Freq/sum(Freq))

# fill alpha relative to sensitivity/specificity by proportional outcomes within
# reference groups
ggplot(data = plot, mapping = aes(x = Reference, y = Prediction, fill = goodbad, alpha = prop)) +
  geom_tile() +
  ggtitle("Confusion matrix: svm2") +
  geom_text(aes(label = Freq), vjust = .5, fontface  = "bold", alpha = 1) +
  scale_fill_manual(values = c(good = "green", bad = "red")) +
  theme(legend.position="none") +
  xlim(rev(levels(table$Reference)))
```



Run models with PCA-transformed data – this is mainly academic, since we know that PCA wasn't an ideal method for dimension reduction in this case.

```
training.data <- cbind(subset(train2, select = c(Category)), pca$x)

# Second number is the number of PCs - 1
training.data <- training.data[,1:12]

# Perform the same transformation to test data as was done of train data,
# including mean centring and scaling.
# We select the first 11 PCs, since PCA did not lead to dimension reduction
testing.data <- as.data.frame(predict(pca, newdata = x.test))[,1:11]
```

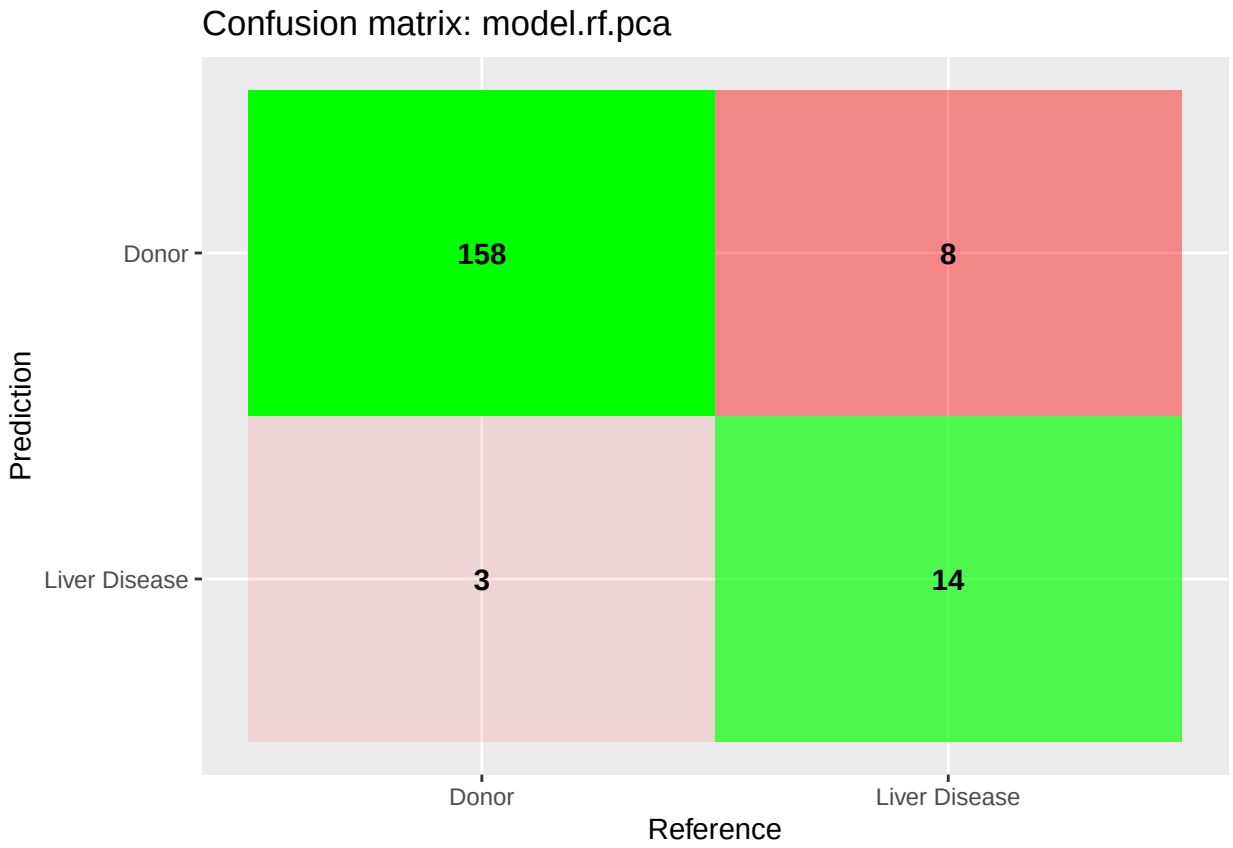
```
# Train classifiers on pca-transformed data

train.control <- trainControl(method="repeatedcv", number=10, repeats=3)
tunegrid <- expand.grid(.mtry=sqrt(ncol(training.data)))
model.rf.pca <- train(Category~., data=training.data, method="rf", metric="Accuracy",
                  tuneGrid=tunegrid, trControl=control)

# Fit the model
svm.pca <- train(Category ~., data = training.data, method = "svmLinear", trControl = train.control)

pred_test3 <- predict(model.rf.pca, testing.data)
cm3 <- confusionMatrix(pred_test3, y.test$Category)
cm3
```

```
## Confusion Matrix and Statistics
##
##                Reference
## Prediction      Liver Disease Donor
##   Liver Disease            14     3
##   Donor                     8   158
##
##                Accuracy : 0.9399
##                  95% CI : (0.895, 0.9696)
##     No Information Rate : 0.8798
##     P-Value [Acc > NIR] : 0.005158
##
##                   Kappa : 0.6849
##
##  Mcnemar's Test P-Value : 0.227800
##
##             Sensitivity : 0.6364
##             Specificity : 0.9814
##          Pos Pred Value : 0.8235
##          Neg Pred Value : 0.9518
##              Prevalence : 0.1202
##          Detection Rate : 0.0765
##    Detection Prevalence : 0.0929
##       Balanced Accuracy : 0.8089
##
##        'Positive' Class : Liver Disease
##
```

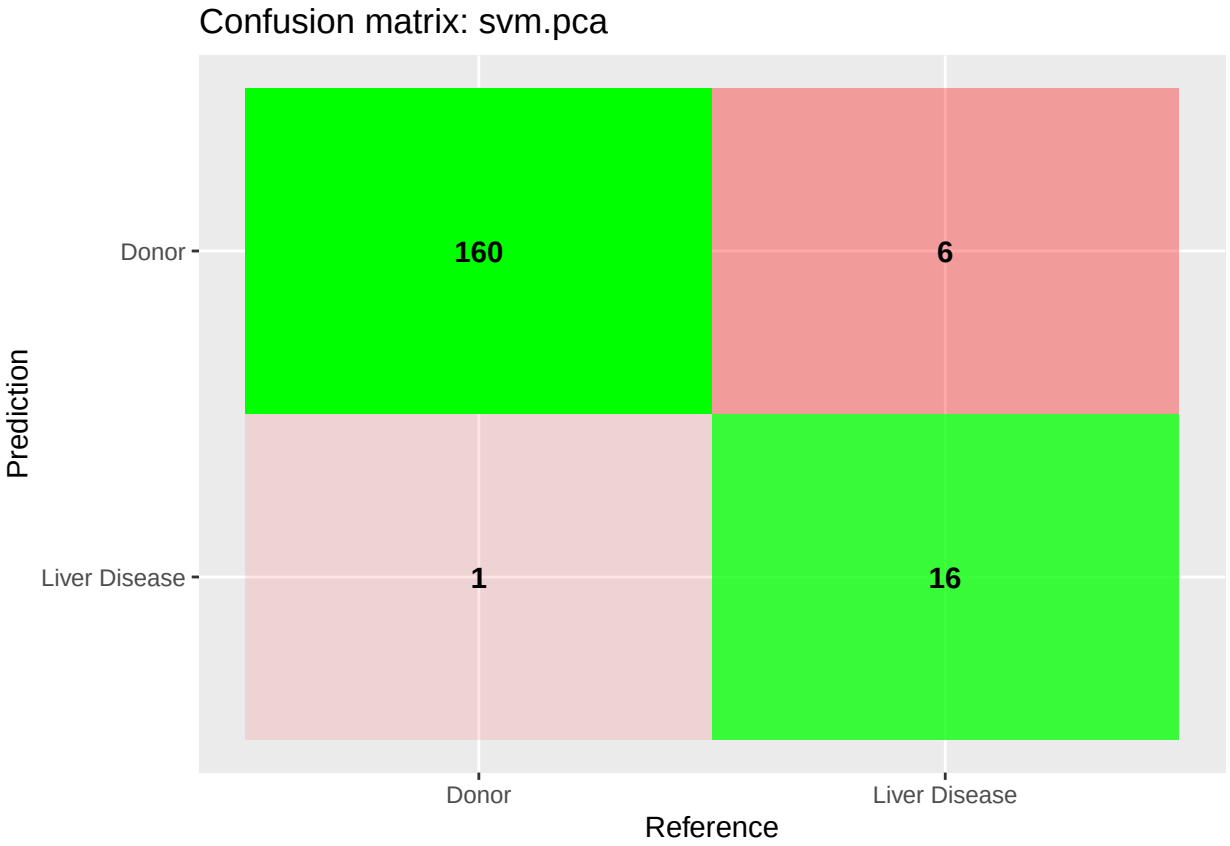
```r
pred_test4 <- predict(svm.pca, testing.data)
cm4 <- confusionMatrix(pred_test4, y.test$Category)
cm4
```

```
## Confusion Matrix and Statistics
##
##                Reference
## Prediction      Liver Disease Donor
##   Liver Disease            16     1
##   Donor                     6   160
##
##                Accuracy : 0.9617
##                  95% CI : (0.9228, 0.9845)
##     No Information Rate : 0.8798
##     P-Value [Acc > NIR] : 9.848e-05
##
##                   Kappa : 0.7995
##
##  Mcnemar's Test P-Value : 0.1306
##
##             Sensitivity : 0.72727
##             Specificity : 0.99379
##          Pos Pred Value : 0.94118
##          Neg Pred Value : 0.96386
##              Prevalence : 0.12022
##          Detection Rate : 0.08743
##    Detection Prevalence : 0.09290
##       Balanced Accuracy : 0.86053
##
##        'Positive' Class : Liver Disease
##
```

```r
# Plot confusion matrix
table <- data.frame(cm3$table)
plot <- table %>%
  mutate(goodbad = ifelse(table$Prediction == table$Reference, "good", "bad")) %>%
  group_by(Reference) %>%
  mutate(prop = Freq/sum(Freq))

# fill alpha relative to sensitivity/specificity by proportional outcomes within
# reference groups
ggplot(data = plot, mapping = aes(x = Reference, y = Prediction, fill = goodbad, alpha = prop)) +
  geom_tile() +
  ggtitle("Confusion matrix: model.rf.pca") +
  geom_text(aes(label = Freq), vjust = .5, fontface  = "bold", alpha = 1) +
  scale_fill_manual(values = c(good = "green", bad = "red")) +
  theme(legend.position="none") +
  xlim(rev(levels(table$Reference)))
```




```
# Plot confusion matrix
table <- data.frame(cm4$table)
plot <- table %>%
  mutate(goodbad = ifelse(table$Prediction == table$Reference, "good", "bad")) %>%
  group_by(Reference) %>%
  mutate(prop = Freq/sum(Freq))

# fill alpha relative to sensitivity/specificity by proportional outcomes within
# reference groups
ggplot(data = plot, mapping = aes(x = Reference, y = Prediction, fill = goodbad, alpha = prop)) +
  geom_tile() +
  ggtitle("Confusion matrix: svm.pca") +
  geom_text(aes(label = Freq), vjust = .5, fontface  = "bold", alpha = 1) +
  scale_fill_manual(values = c(good = "green", bad = "red")) +
  theme(legend.position="none") +
  xlim(rev(levels(table$Reference)))
```



With the PCA-transformed data, the accuracy has been reduced in the RF model, and marginally increased in the SVM model. In general, there is no statistical benefit to using PCA data in classification models – this we have shown, and it is consistent with the results. PCA is also used for reasons other than feature reduction, such as increasing computational efficiency, and interpreting data during EDA.

Over-sampling

From the results about, we can see that, in all the models, the sensitivity is much lower than the specificity. Recall that a lower sensitivity means that the model is predicting a marked number false negatives, and therefore some instances of the disease are being missing by the classifier. The specificity is high in both models, which is good. So, what could account for this imbalance? One theory we shall test is the notion that, due to the highly imbalanced nature of the target classes, the models are biased to predicting that a patient who truly does not have the disease is 'negative.' Therefore, the model is missing some instances of the disease among the population. A method to attempt to rectify this issue is the process of over-sampling.

We use over-sampling when the amount of our data is either too small, or highly imbalanced. When the data are more balanced in the training set, the models have a greater chance to learn how to correctly classify patients. SMOTE (synthetic minority over-sampling technique) is a method for over-sampling, but it doesn't deal with categorical data [17].

The technique we shall use in this study is SMOTE-NC (synthetic minority over-sampling technique – nominal continuous). Compared to random over-sampling, SMOTE-NC offers an improvement. Furthermore, SMOTE-NC has the ability of handling categorical data, such as the male/female class. Simply one-hot encoding these features would, in some algorithms, be interpreted as continuous data. Values between 0 and 1 may be be assigned. Furthermore, we should not naively round these, thinking that the algorithm is telling us that this person is predicted as 'more female' than male; this would have to same effect as simply choosing random values between 0 and 1 – this is not what we wish to do. SMOTE-NC forms synthetic data by randomly sampling from minority classes. In our case, these are the classes with disease (hepatitis,

fibrosis, and cirrhosis). SMOTE-NC has been used, with 'gradient boosting imputation based random forest classifiers' to predict the 'severity level of covid-19 patients with blood samples.' [18] Unfortunately, many of these over-sampling methods are not available in R, or the packages have been removed from CRAN. Therefore, we shall use Python to perform this over-sampling of the data.

Note that over-sampling has disadvantages, since it can introduce both 'noise' and 'bias' into the data. To avoid introducing much bias – as much as we can help it – we shall over-sample just the minority classes of hepatitis, fibrosis, and cirrhosis. Furthermore, we shall do this in a way such that the total number of instances of the classes of 'donor' and 'liver disease' remain fairly balanced. Otherwise, we might create a classifier than now skews more to high sensitivity, while simultaneously reducing specificity too much – the pendulum will have swung too far in the other direction. Our method here attempts to mitigate this issue.

We over-sample from already standardised data. Hence, the new synthetic data will also be standardised. In this way, we can just concatenate the SMOTE-NC data with the training dataset.

Please now see the smote.Rmd and smote.py files – this step is outlined in the README.md file

Results and Conclusion

We shall give the results for each model as follows: (accuracy, sensitivity, specificity).

Ranfom forest (RF)

RF, normalised and mean-centred data

No tuning: (96.2%, 0.682, 1.00) Tuning: (96.7%, 0.773, 0.994)

RF, PCA-transformed data

No tuning: (94.0%, 0.636, 0.981)

RF, SMOTE-NC over-sampled data

No tuning: (97.3%, 0.864, 0.988) Tuning: (97.8%, 0.863, 0.994)

Support vector machines (SVM)

SVM, normalised and mean-centred data

No tuning: (95.6%, 0.727, 0.988) Tuning: (95.6%, 0.727, 0.988)

SVM, PCA-transformed data

No tuning: (96.2%, 0.727, 0.994)

SVM, SMOTE-NC over-sampled data

No tuning: (95.6%, 0.773, 0.981) Tuning: (94.0%, 0.682, 0.975)

Conclusion

The application of machine learning algorithms to medical classification is a growing (but still emerging) field of study. It has been shown that the presence of disease can be successfully predicted, by the classifiers, using the features of the data provided. Ultimately, the use of PCA was not fruitful in the pursuit of dimension reduction, but it has been shown as a method for dimensional reduction in other medical classifiers [19]. FAMD was then used, but the dimensions were again not reduced. The performance of each model

was evaluated using the metric (accuracy, sensitivity, specificity). We adapted the data to create a binary classification problem, making the predictor choose between 'donor' and 'liver disease.'

The best performing model was the hyper-perameter tuned RF model with SMOTE-NC, with the results (97.8%, 0.863, 0.994). This is a marked increase over the original values of (96.2%, 0.682, 1.00). Note that the presence of SMOTE-NC has successfully increased the sensitivity from 0.682, to 0.863. We notice that the specificity has gone down slightly, but we expect this, since there is usually a trade-off between these two values. We should note the limitations of the model in generating a high sensitivity. This is likely because of the large imbalance between the majority and minority classes of disease level in the original data. Ideally, in the future, we should like to test this model again with more data so that a more robust conclusion can be made. Nevertheless, in an attempt to solve this problem, over-sampling using SMOTE-NC was successfully employed, and its use formed the best performing model.

In general, SMOTE-NC was successful at increasing the sensitivity of the classifiers in all but one case (the tuned SVM model). This model did not respond well to the use of over-sampled data. Commonly, using a different value of the hyper-parameter 'C' for each minority class is asymptotically equivalent to over-sampling by changing the class frequencies. Only one C value was tuned in the optimisation stage. This could have caused the reduced performance, but it is not entirely clear. Further analysis should be done into why this has happened.

In each model using the PCA-transformed data led to different conclusions. With RF, its use weakened the classifier by reducing its accuracy, sensitivity and specificity. However, its use in SVM strengthened it, and increased the accuracy, while only marginally reducing the specificity. More analysis is required to understand why this is. One possibility is that the PCA data uses fewer features/columns than in the scaled data, or the scaled over-sampled data.

As we can see, the tuning of hyper-parameters generally helped the models to perform better, except in the instance of SVM with SMOTE-NC. The method of hyper-parameter optimisation used in this project was very basic. In the future, this should be made more robust. A way of doing this could be the use of programs such as 'Optuna.'

It seems that the level of certain blood markers (proteins and amino acids) is a successful predictor of a patient's possibility of having liver disease, and it has been shown to be a good predictor of the absence of liver disease in patients – this is shown in the generally very high specificity values in the models. In the future, it would be interesting to test these models on datasets with different features, and also datasets from different diseases.

References

1. Lichtinghagen K Ralf, Hoffmann G (2020) HCV data

2. Barnard J, Meng X-L (1999) Applications of multiple imputation in medical studies: From AIDS to NHANES. Statistical Methods in Medical Research 8:17–36. https://doi.org/10.1177/096228029900800103

3. Azur MJ, Stuart EA, Frangakis C, Leaf PJ (2011) Multiple imputation by chained equations: What is it and how does it work? International Journal of Methods in Psychiatric Research 20:40–49. https://doi.org/10.1002/mpr.329

4. Raghunathan TE, Solenberger PW, Van Hoewyk J (2002) IVEware: Imputation and variance estimation software. Ann Arbor, MI: Survey Methodology Program, Survey Research Center, Institute for Social Research, University of Michigan

5. Abdel-Gawad M, Nour M, El-Raey F, et al (2023) Gender differences in prevalence of hepatitis c virus infection in egypt: A systematic review and meta-analysis. Scientific Reports 13: https://doi.org/10.1038/s41598-023-29262-z

6. Jolliffe I (2022) A 50-year personal journey through time with principal component analysis. Journal of Multivariate Analysis 188:104820. https://doi.org/10.1016/j.jmva.2021.104820

7. Uddin MdP, Mamun MdA, Hossain MdA (2020) PCA-based feature reduction for hyperspectral remote sensing image classification. IETE Technical Review 38:377–396. https://doi.org/10.1080/02564602.2020.1740615
8. Zhang Z, Castelló A (2017) Principal components analysis in clinical studies. Annals of Translational Medicine 5:351–351. https://doi.org/10.21037/atm.2017.07.12
9. Jolliffe IT, Cadima J (2016) Principal component analysis: A review and recent developments. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 374:20150202. https://doi.org/10.1098/rsta.2015.0202
10. Ran X (2019) Using the dimension reduction method FAMD in the data pre-processing step for risk prediction and for unsupervised clustering
11. Pagès J (2014) Multiple factor analysis by example using r
12. KETCHEN Jr. DJ, SHOOK CL (1996) THE APPLICATION OF CLUSTER ANALYSIS IN STRATEGIC MANAGEMENT RESEARCH: AN ANALYSIS AND CRITIQUE. Strategic Management Journal 17:441–458. https://doi.org/10.1002/(sici)1097-0266(199606)17:6%3C441::aid-smj819%3E3.0.co;2-g
13. Yaqoob MK, Ali SF, Bilal M, et al (2021) ResNet based deep features and random forest classifier for diabetic retinopathy detection. Sensors 21:3883. https://doi.org/10.3390/s21113883
14. Probst P, Wright M, Boulesteix A-L (2018) Hyperparameters and tuning strategies for random forest. https://doi.org/10.48550/ARXIV.1804.03515
15. Bernard S, Heutte L, Adam S (2009) Influence of hyperparameters on random forest accuracy. In: Lecture notes in computer science. Springer Berlin Heidelberg, pp 171–180
16. Zhou X, Li X, Zhang Z, et al (2022) Support vector machine deep mining of electronic medical records to predict the prognosis of severe acute myocardial infarction. Frontiers in Physiology 13: https://doi.org/10.3389/fphys.2022.991990
17. Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP (2011) SMOTE: Synthetic minority over-sampling technique. https://doi.org/10.48550/ARXIV.1106.1813
18. Gök EC, Olgun MO (2021) SMOTE-NC and gradient boosting imputation based random forest classifier for predicting severity level of covid-19 patients with blood samples. Neural Computing and Applications 33:15693–15707. https://doi.org/10.1007/s00521-021-06189-y
19. Gárate-Escamila AK, Hajjam El Hassani A, Andrès E (2020) Classification models for heart disease prediction using feature selection and PCA. Informatics in Medicine Unlocked 19:100330. https://doi.org/10.1016/j.imu.2020.100330