HarvardX: PH125.9x Data Science
Leukemia Classification
CYO Project

Spyridon Ntokos

May 26, 2020

Contents

1 Overview

This project is related to the CYO (Choose Your Own) Project of the HarvardX: PH125.9x Data Science: Capstone course. The objective of this project is to build a classification model of a leukemia patients' gene expression dataset to be evaluated by measuring accuracy.

1.1 Introduction

Cancer is one of the leading causes of deaths today. Being able to correctly classify it, leads to the appropriate selection of therapeutic approaches and the most efficient treatment.

This Gene Expression Dataset (Golub et al.) contains data on 72 patients suffering from either Acute Myeloid Leukemia (AML) or Acute Lymphoblastic Leykemia (ALL). Thus, through the help of sufficient data analysis and machine learning techniques, an attempt to accurately classify the patients between the two cancer types is made.

1.2 Aim of the project

The aim of this project is to develop a machine learning algorithm using the inputs in training subset (~52% of the patients' data) to predict cancer type in the independent (validation) subset (the remaining ~47% of them). Several machine learning algorithms have been used and results have been compared to get maximum possible accuracy in prediction.

This report contains problem definition, data ingestion, data preparation / cleansing, exploratory analysis, modeling and data analysis and results. Finally the report ends with some concluding remarks.

1.3 Problem definition

This capstone project on "Leukemia Classification" predicts the cancer type of a leukemia patient based on their gene expression values. The dataset used for this purpose can be found in the following link:

- [Gene expression dataset (Golub et al.)] https://www.kaggle.com/crawford/gene-expression/data
- [Gene expression dataset (Golub et al.) - zip file] https://storage.googleapis.com/kaggle-data-sets/
1868/3249/bundle/archive.zip?GoogleAccessId=web-data@kaggle-161607.iam.gserviceaccount.com&
Expires=1590574751&Signature=pNKbnTCWdotyK7OrAVw%2FRlxbTyU2%2BbtseYhX3yd0ahsPrCubhMpDU1U5G
2FaO0jTZzQ0dcXTBOYtR%2BrnUrM3Gcg3N%2FIrWJvw8EP%2FThEllTskhx6MsmeynQ%
2BR8cyt6vWWR9S6%2Fm73OBLs7E1xBo6wpurikEP%2Fq%2FHx9Q5ugdIMtPO57Kw6N4pkzDBWMyPmKBlOXj3v
2FbPn1jt2C%2BB5ksmnV2ICcbTIQ9kCn6ayiY%2B3Eai1tCyLTasS18E2diMEZ95qnY5vQxw5lDXJwq3G%
2F4HQ1o90%2FDmtngJ9YI1hidl%2Fk3Xb%2BiyWYg%3D%3D&response-content-disposition=
attachment%3B+filename%3Dgene-expression.zip

The challenge is not so easy given that there are thousands of genes under evaluation of their expression level present in the dataset, while the number of patients in the training dataset is really low to make trustworthy accurate predictions.

The main idea is to develop an ensemble algorithm based on several different kinds of machine learning algorithms to most effectively predict patients' cancer type in the validation subset according to gene expression values of patients in the training subset.

2 Preparation Stage

2.1 Data ingestion

Data is downloaded from kaggle's website, using the appropriate URL. The three contained files also stored locally, while they are loaded for the required analysis. Required libraries for upcoming analysis are loaded, unless they are not yet installed.

```
################################################
# Create train & validation (indepedent) sets #
################################################

if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(ggplot2)) install.packages("ggplot2", repos = "http://cran.us.r-project.org")
if(!require(naniar)) install.packages("naniar", repos = "http://cran.us.r-project.org")
if(!require(matrixStats)) install.packages("matrixStats", repos = "http://cran.us.r-project.org")
if(!require(gridExtra)) install.packages("gridExtra", repos = "http://cran.us.r-project.org")


# Gene expression dataset (Golub et al.):
# https://www.kaggle.com/crawford/gene-expression/data

dl <- tempfile()
download.file("https://storage.googleapis.com/kaggle-data-sets/1868/3249/bundle/archive.zip?GoogleAcces
              dl)

# Extract .csv files
actual <- read.csv(unzip(dl, "actual.csv"), stringsAsFactors = FALSE)
train_set <- read.csv(unzip(dl, "data_set_ALL_AML_train.csv"), stringsAsFactors = FALSE)
validation <- read.csv(unzip(dl, "data_set_ALL_AML_independent.csv"), stringsAsFactors = FALSE)

rm(dl)
```

Note: Anything no longer used will be considered unnecessary and manually removed (keep workspace clean & tidy).

2.2 Data understanding

To understand the data, we need to take a glimpse of it and determine its dimensions.

Actual dataset:

```
##   patient cancer
## 1       1    ALL
## 2       2    ALL
## 3       3    ALL
## 4       4    ALL
## 5       5    ALL
## 6       6    ALL
```

```
## [1] 72  2
```

This set contains the data for all 72 patients and their cancer type.

Training subset:

```
## Warning: ‘as.tibble()‘ is deprecated as of tibble 2.0.0.
## Please use ‘as_tibble()‘ instead.
## The signature and semantics have changed, see ‘?as_tibble‘.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_warnings()‘ to see where this warning was generated.
```

```
## # A tibble: 6 x 15
##   Gene.Description Gene.Accession.~    X1 call     X2 call.1    X3 call.2    X4
##   <chr>            <chr>            <int> <chr> <int> <chr>  <int> <chr>  <int>
## 1 AFFX-BioB-5_at ~ AFFX-BioB-5_at    -214 A      -139 A        -76 A       -135
## 2 AFFX-BioB-M_at ~ AFFX-BioB-M_at    -153 A       -73 A        -49 A       -114
## 3 AFFX-BioB-3_at ~ AFFX-BioB-3_at     -58 A        -1 A       -307 A        265
## 4 AFFX-BioC-5_at ~ AFFX-BioC-5_at      88 A       283 A        309 A         12
## 5 AFFX-BioC-3_at ~ AFFX-BioC-3_at    -295 A      -264 A       -376 A       -419
## 6 AFFX-BioDn-5_at~ AFFX-BioDn-5_at   -558 A      -400 A       -650 A       -585
## # ... with 6 more variables: call.3 <chr>, X5 <int>, call.4 <chr>, X6 <int>,
## #   call.5 <chr>, X7 <int>
```

```
## [1] 1648   78
```

Every row in the set represents a gene, where all *X* columns represent the expression value for every patient and *call* columns give more information about the expression level.

Validation subset:

```
## # A tibble: 6 x 15
##   Gene.Description Gene.Accession.~   X39 call    X40 call.1   X42 call.2   X47
##   <chr>            <chr>            <int> <chr> <int> <chr>  <int> <chr>  <int>
## 1 AFFX-BioB-5_at ~ AFFX-BioB-5_at    -342 A       -87 A         22 A       -243
## 2 AFFX-BioB-M_at ~ AFFX-BioB-M_at    -200 A      -248 A       -153 A       -218
## 3 AFFX-BioB-3_at ~ AFFX-BioB-3_at      41 A       262 A         17 A       -163
## 4 AFFX-BioC-5_at ~ AFFX-BioC-5_at     328 A       295 A        276 A        182
## 5 AFFX-BioC-3_at ~ AFFX-BioC-3_at    -224 A      -226 A       -211 A       -289
## 6 AFFX-BioDn-5_at~ AFFX-BioDn-5_at   -427 A      -493 A       -250 A       -268
## # ... with 6 more variables: call.3 <chr>, X48 <int>, call.4 <chr>, X49 <int>,
## #   call.5 <chr>, X41 <int>
```

```
## [1] 7129   70
```

Similar to training set, with the main difference in the observed genes number.

2.3 Reshaping the data

Proceeding with data preparation / cleansing:

- **Actual dataset:**

The actual dataset is to be divided into two parts, each holding the data of patients in the training and validation subsets, respectively.

```r
actual_train <- actual[1:38, ]
actual_validation <- actual[39:72, ]
```

- **Training dataset:**

Both training and validation sets contain *call* columns, which have 3 possible values: Present (P), Absent (A), and Marginal (M). This article (found at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1409797/) sums it up nicely by mentioning that the DNA microarray manufacturer Affymetrix (now a subsidiary of Thermo Fisher Scientific):

"[…] uses a non-parametric statistical test (Wilcoxon signed rank test) of whether significantly more perfect matches show more hybridization signal than their corresponding mismatches to produce the detection call (Absent (A), Present (P) or Marginal (M))[…]".

However, the values don't provide any additional information in this data analysis, so we are dropping them:

```r
call_columns <- grep(pattern = "call", x = names(train_set))
train_set <- train_set[,-call_columns]
```

Next, the dataset is to be reshaped so that every row corresponds to patients' data, rather than to every observed gene:

```r
# Store 'Gene Access Name' column as new column headers
gene_id <- as.vector(t(train_set[,2]))

# Keep only numeric data
train_set <- train_set[, - c(1,2)]

# Transpose table and name new columns according to 'gene_id'
# and rows according to patient ID
train_set <- as.data.frame(t(train_set))
colnames(train_set) <- gene_id
train_set <- cbind(patient = row.names(train_set),
                   cancer_type = actual_train$cancer,
                   train_set)
```

As the patient IDs in the training set are preceded by an 'X', simply clear it, in order to match with the *patient* column in the respective actual set:

```r
train_set <- train_set %>%
  separate(patient, c(NA, "patient"), sep = "X")
```

Any missing values must be reported:

```r
anyNA(train_set)
```

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

```r
try(gg_miss_upset(train_set))
```

```
## Error : upset plots for missing data requre at least two variables to have missing data, only one va
```

Spot the missing values in reported column and identify the reason:

```r
train_set[, 'Var.1650']
```

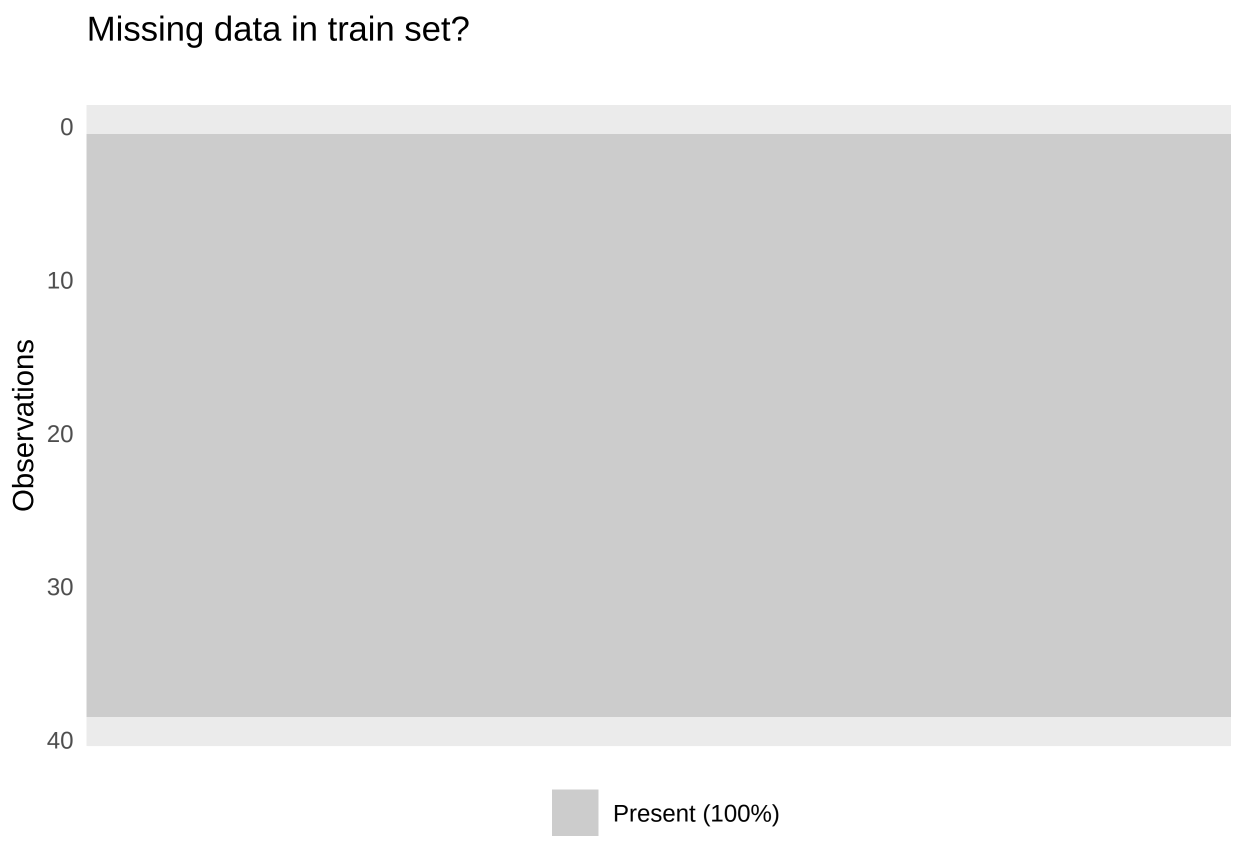
```
##  [1] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
## [26] NA NA NA NA NA NA NA NA NA NA NA NA NA NA
```

```r
gene_id[1649]
```

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

As seen above, the whole last column is comprised of NAs and doesn't correspond to any gene, thus its removal is necessary:

```r
train_set <- train_set[, - c(1650)]
vis_miss(train_set) +
  theme(axis.text.x = element_blank()) +
  ggtitle("Missing data in train set?")
```



As observed in the previous plot, no NAs are included in the dataset anymore, so we sum up our data cleansing by confirming our changes:

```r
head(train_set[, 1:15]) %>% as_tibble()
```

```
## # A tibble: 6 x 15
##   patient cancer_type `AFFX-BioB-5_at` `AFFX-BioB-M_at` `AFFX-BioB-3_at`
##   <chr>   <fct>                  <int>            <int>            <int>
## 1 1       ALL                     -214             -153              -58
## 2 2       ALL                     -139              -73               -1
## 3 3       ALL                      -76              -49             -307
## 4 4       ALL                     -135             -114              265
## 5 5       ALL                     -106             -125              -76
## 6 6       ALL                     -138              -85              215
## # ... with 10 more variables: `AFFX-BioC-5_at` <int>, `AFFX-BioC-3_at` <int>,
## #   `AFFX-BioDn-5_at` <int>, `AFFX-BioDn-3_at` <int>, `AFFX-CreX-5_at` <int>,
## #   `AFFX-CreX-3_at` <int>, `AFFX-BioB-5_st` <int>, `AFFX-BioB-M_st` <int>,
## #   `AFFX-BioB-3_st` <int>, `AFFX-BioC-5_st` <int>
```

- **Validation dataset:**

Firstly, previously was mentioned the fact that the validation set is observing more genes than the training dataset, so it must be made sure that the two datasets' observed genes match:

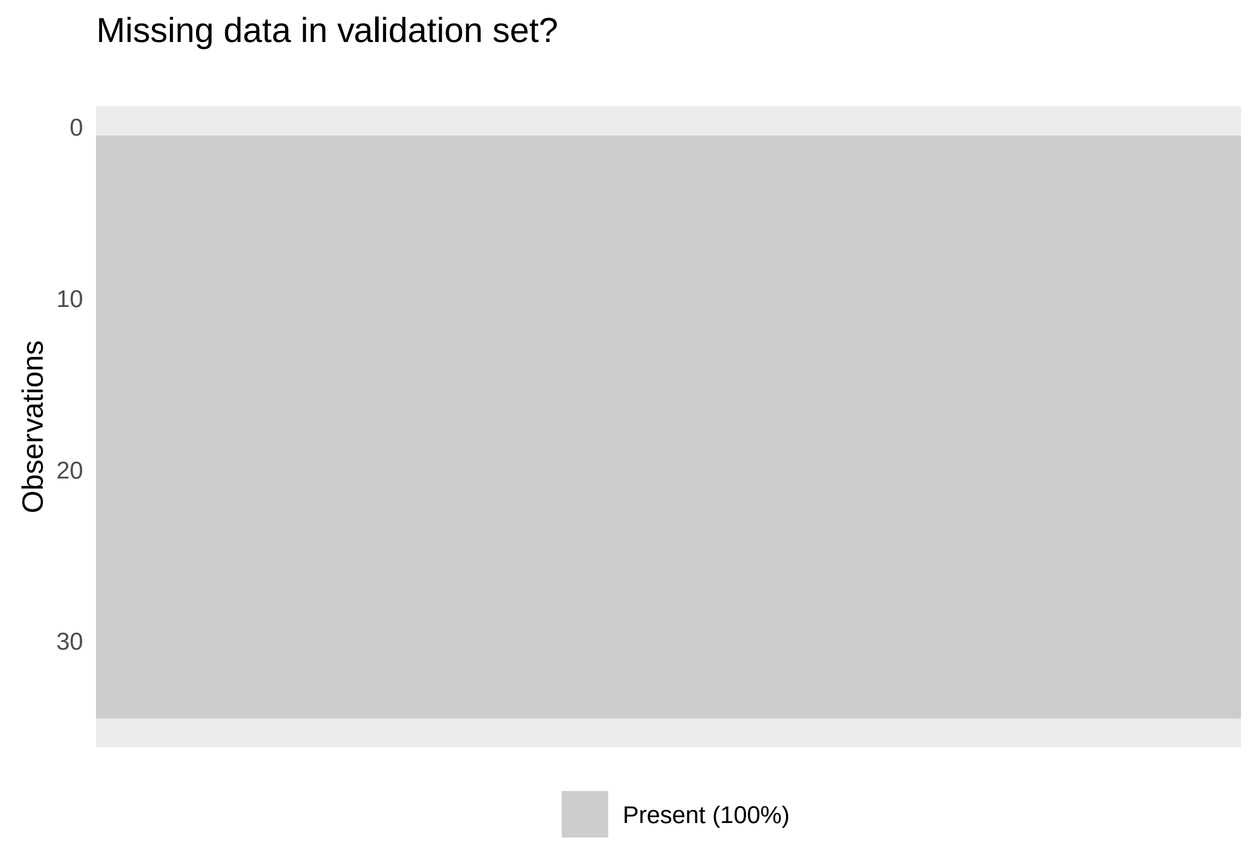
```
validation <- validation %>%
  semi_join(original_train, by = "Gene.Accession.Number")
```

Afterwards, the same procedure is followed as in training set's case:

```
# Removing irrelevant 'call' columns
call_columns <- grep(pattern = "call", x = names(validation))
validation <- validation[,-call_columns]

# Store 'Gene Access Name' column as new column headers
gene_id <- as.vector(t(validation[,2]))

# Keep only numeric data
validation <- validation[, - c(1,2)]

# Transpose table and name new columns according to 'gene_id'
# and rows according to patient ID
validation <- as.data.frame(t(validation))
colnames(validation) <- gene_id
validation <- cbind(patient = row.names(validation),
                    cancer_type = actual_validation$cancer,
                    validation)

# Drop 'X' from patient IDs
validation <- validation %>%
  separate(patient, c(NA, "patient"), sep = "X")
```

Once again, check for missing values:

```
vis_miss(validation) +
  theme(axis.text.x = element_blank()) +
  ggtitle("Missing data in validation set?")
```



Proceed by confirming applied changes:

```
head(validation[, 1:15]) %>% as_tibble()
```

```
## # A tibble: 6 x 15
##   patient cancer_type `AFFX-BioB-5_at` `AFFX-BioB-M_at` `AFFX-BioB-3_at`
##   <chr>   <fct>                  <int>            <int>            <int>
## 1 39      ALL                     -342             -200               41
## 2 40      ALL                      -87             -248              262
## 3 42      ALL                       22             -153               17
## 4 47      ALL                     -243             -218             -163
## 5 48      ALL                     -130             -177              -28
## 6 49      ALL                     -256             -249             -410
## # ... with 10 more variables: `AFFX-BioC-5_at` <int>, `AFFX-BioC-3_at` <int>,
## #   `AFFX-BioDn-5_at` <int>, `AFFX-BioDn-3_at` <int>, `AFFX-CreX-5_at` <int>,
## #   `AFFX-CreX-3_at` <int>, `AFFX-BioB-5_st` <int>, `AFFX-BioB-M_st` <int>,
## #   `AFFX-BioB-3_st` <int>, `AFFX-BioC-5_st` <int>
```

3 Exploratory Data Analysis

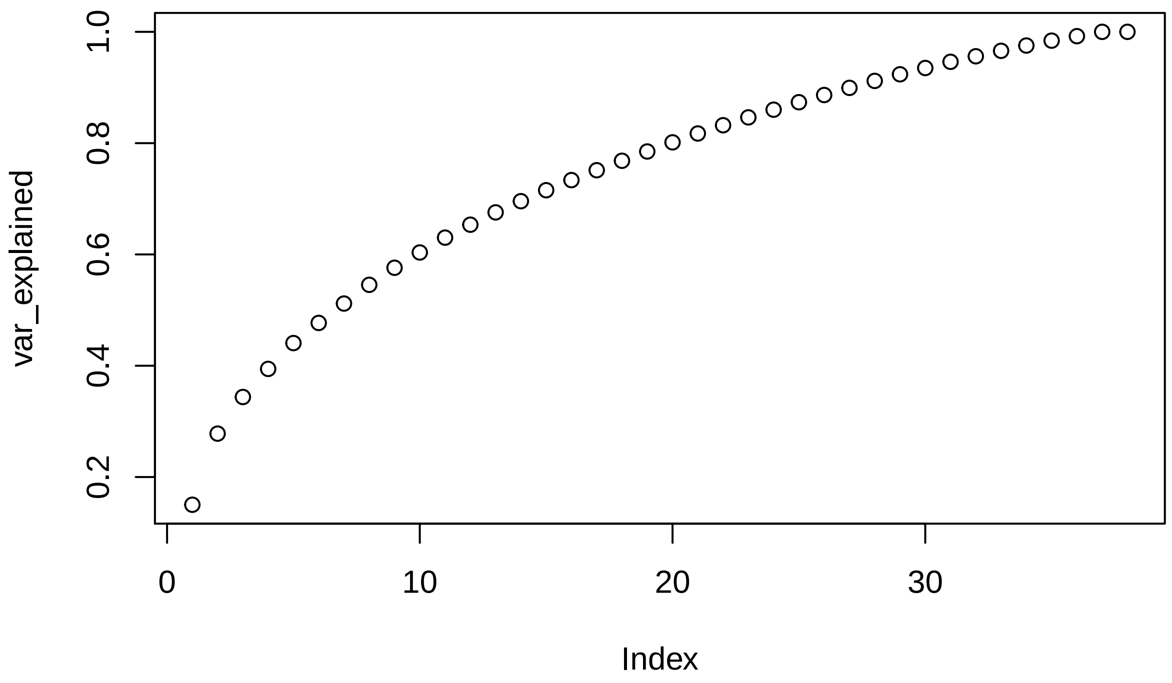
3.1 Principal Component Analysis (PCA)

PCA analysis is used to categorize data into different clusters based on their similarities. In our case though, it is not really needed as the categories / clusters are already known (AML or ALL) but after a short PCA analysis, the most important principal components will be determined and patients will be visually clustered according to them.

```
# Matrix transformation
x <- as.matrix(train_set[, - c(1,2)])
x_centered <- sweep(x, 2, colMeans(x), FUN = "-")
x_scaled <- sweep(x_centered, 2, colSds(x), FUN = "/")

# Proportion of variance
pca <- prcomp(x_scaled)
summary(pca)
```

```
## Importance of components:
##                            PC1     PC2      PC3     PC4     PC5     PC6     PC7
## Standard deviation     15.7288 14.5086 10.41144 9.11575 8.74839 7.71875 7.57369
## Proportion of Variance  0.1502  0.1278  0.06582 0.05045 0.04647 0.03617 0.03483
## Cumulative Proportion   0.1502  0.2780  0.34383 0.39428 0.44075 0.47693 0.51176
##                            PC8     PC9    PC10    PC11    PC12    PC13    PC14
## Standard deviation     7.44523 7.10708 6.72542 6.62613 6.19152 6.03239 5.76067
## Proportion of Variance 0.03366 0.03067 0.02746 0.02666 0.02328 0.02209 0.02015
## Cumulative Proportion  0.54541 0.57608 0.60354 0.63020 0.65348 0.67557 0.69572
##                           PC15    PC16    PC17    PC18    PC19   PC20    PC21
## Standard deviation     5.69762 5.46600 5.41090 5.28405 5.24635 5.1973 5.11985
## Proportion of Variance 0.01971 0.01814 0.01778 0.01695 0.01671 0.0164 0.01592
## Cumulative Proportion  0.71543 0.73357 0.75135 0.76830 0.78501 0.8014 0.81733
##                           PC22    PC23   PC24    PC25    PC26    PC27    PC28
## Standard deviation     4.94964 4.82042 4.7682 4.70064 4.62361 4.59421 4.54197
## Proportion of Variance 0.01487 0.01411 0.0138 0.01342 0.01298 0.01282 0.01253
## Cumulative Proportion  0.83220 0.84631 0.8601 0.87353 0.88651 0.89933 0.91185
##                           PC29    PC30    PC31    PC32   PC33   PC34   PC35
## Standard deviation     4.42187 4.31586 4.28001 4.04707 4.0183 3.9559 3.8073
## Proportion of Variance 0.01187 0.01131 0.01112 0.00994 0.0098 0.0095 0.0088
## Cumulative Proportion  0.92372 0.93503 0.94616 0.95610 0.9659 0.9754 0.9842
##                           PC36    PC37      PC38
## Standard deviation     3.63834 3.57422 8.185e-15
## Proportion of Variance 0.00804 0.00776 0.000e+00
## Cumulative Proportion  0.99224 1.00000 1.000e+00
```

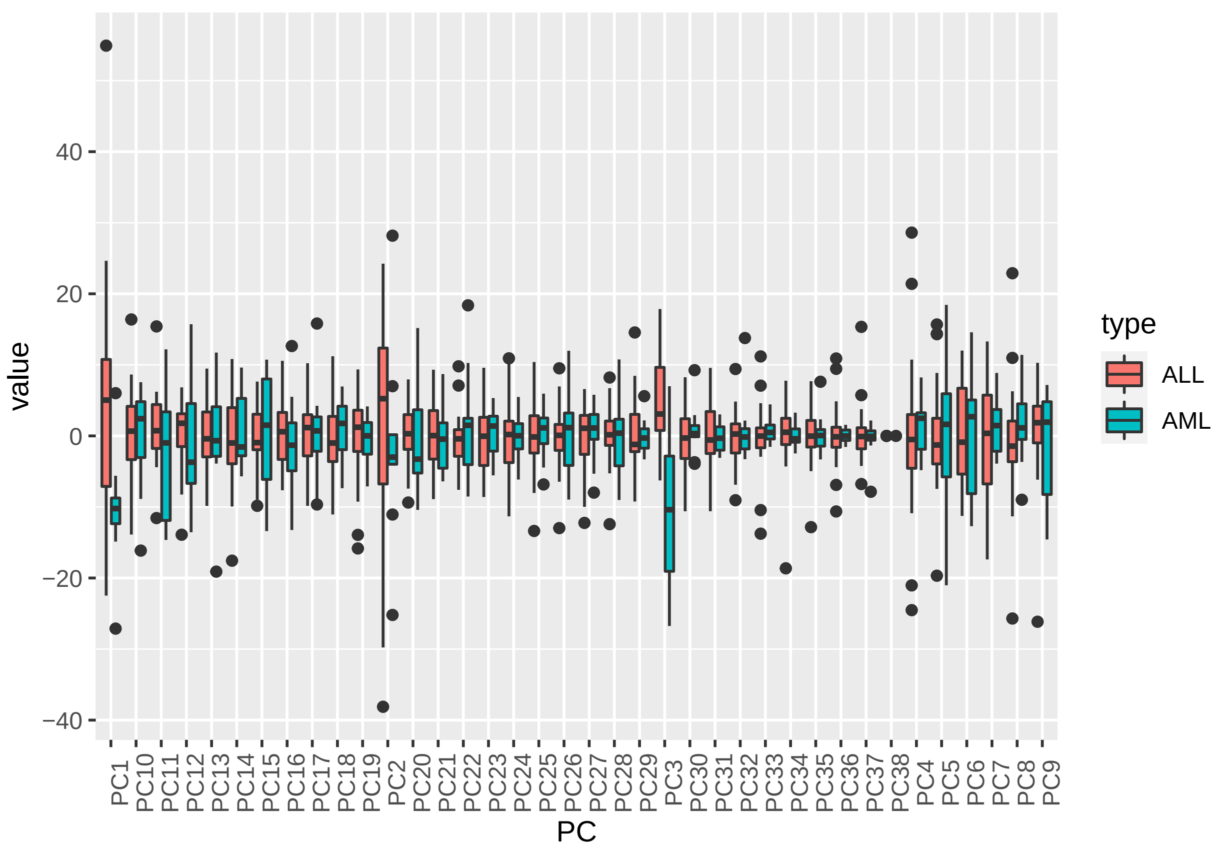
```
var_explained <- cumsum(pca$sdev^2/sum(pca$sdev^2))
plot(var_explained)
```



According to the plot above, one can see that almost 95% of the variance is explained by the first 30 columns.

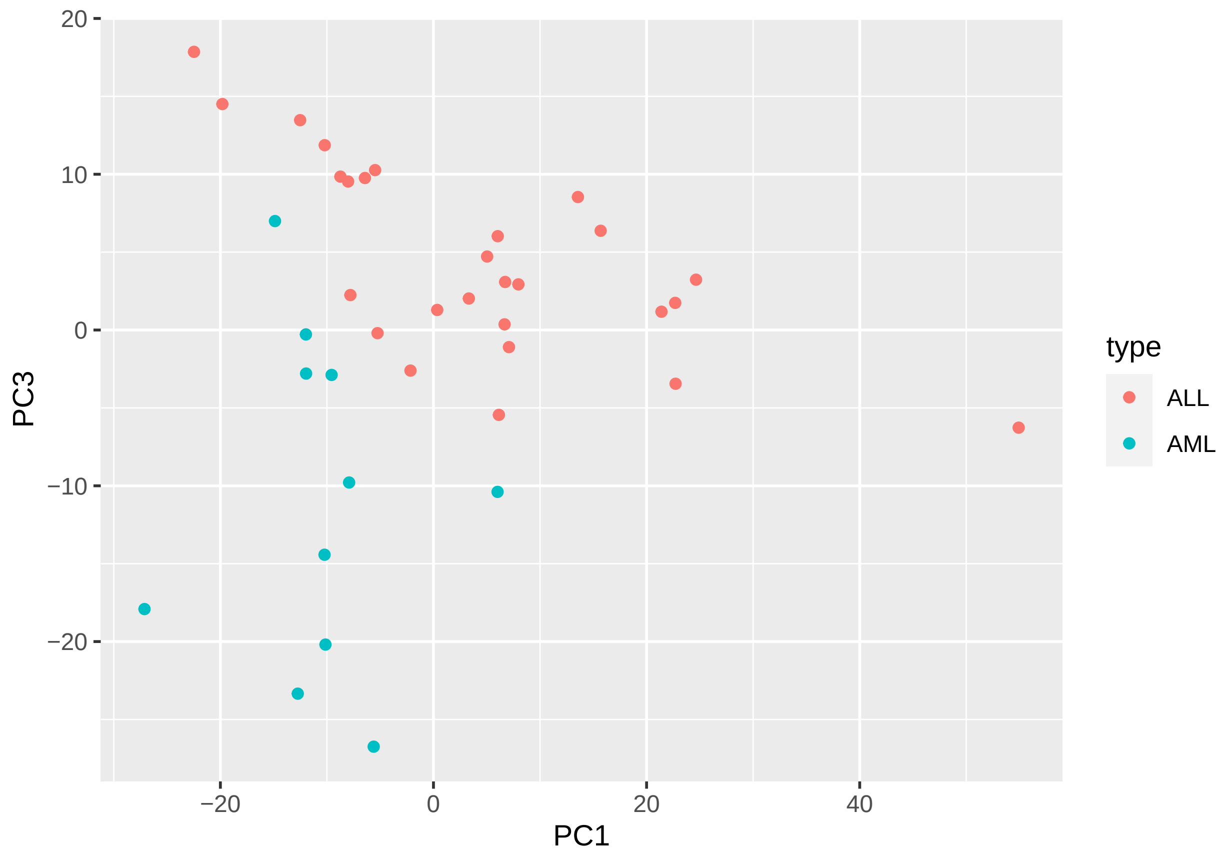
Next, try to figure out which PCs' difference is so great that they can be used to cluster our patients (visually the boxplots in these PCs won't overlap each other):

```
data.frame(type = actual_train$cancer, pca$x) %>%
  gather(key = "PC", value = "value", -type) %>%
  ggplot(aes(PC, value, fill = type)) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 90))
```



The detected non-overlapping PCs are PC1 and PC3:

```
data.frame(pca$x[,c(1,3)], type = actual_train$cancer) %>%
  ggplot(aes(PC1, PC3, color = type)) +
  geom_point()
```



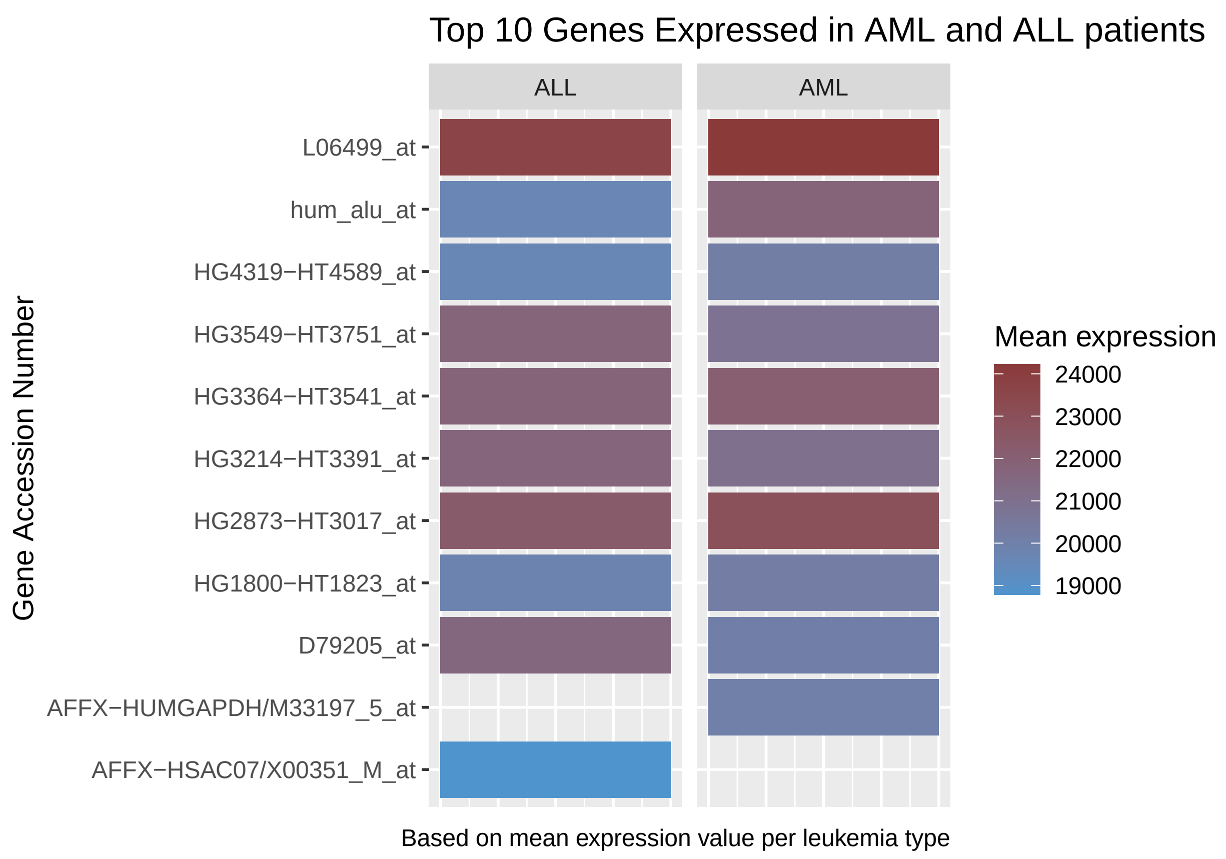
The above plot explains that AML patients tend to have negative values for both PC1 and PC3, where the exact opposite holds for the ALL patients.

3.2 Data visualization

Due to the high amount of observed genes in the dataset, we 'll focus on just a few of them.

3.2.1 Top 10 genes expressed by leukemia type

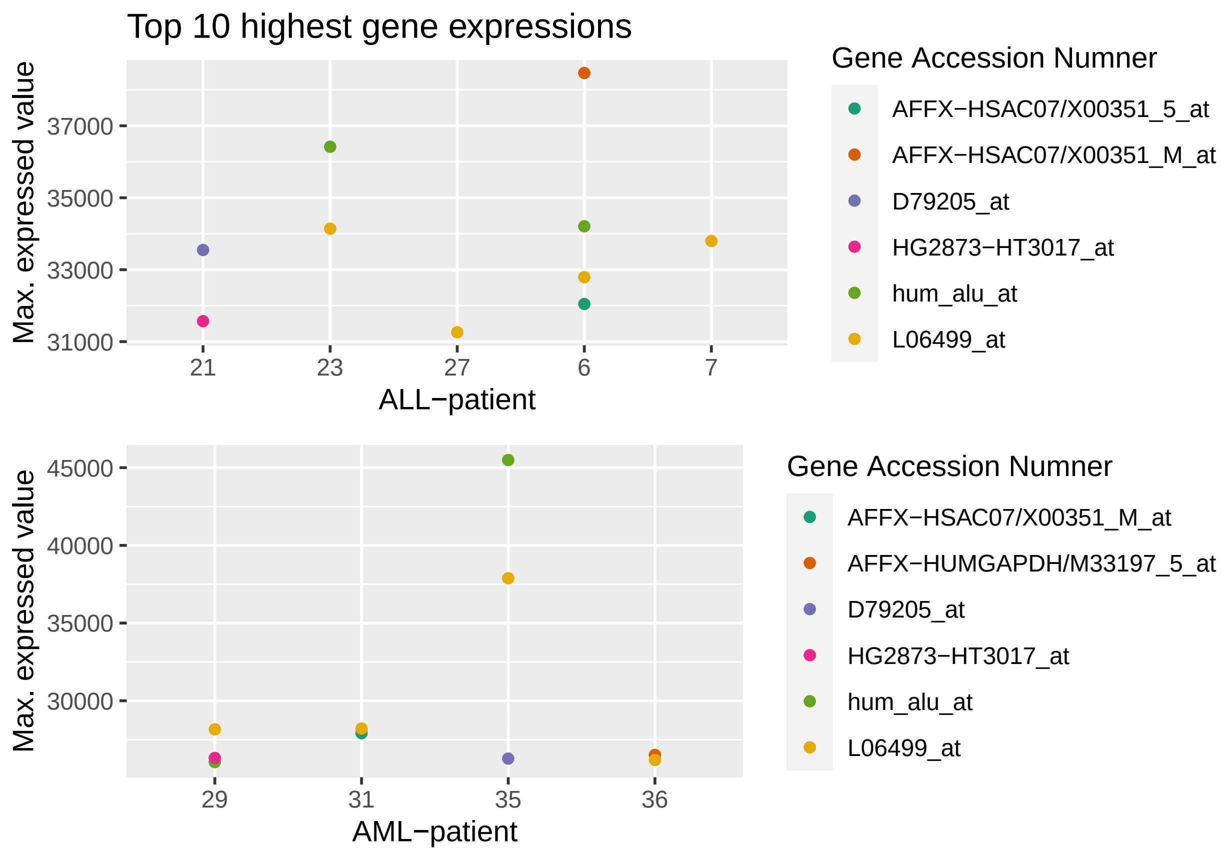
```
train_set %>%
  select(-patient) %>%
  group_by(cancer_type) %>%
  summarise_all(list(mean)) %>%
  gather("gene", "mean", - cancer_type) %>%
  group_by(cancer_type) %>%
  top_n(10) %>%
  ggplot(aes(gene, fill = mean)) +
  geom_bar() +
  facet_wrap(~cancer_type) +
  coord_flip() +
  ggtitle("Top 10 Genes Expressed in AML and ALL patients") +
  labs(x = "Gene Accession Number",
       caption = "Based on mean expression value per leukemia type") +
  scale_fill_continuous("Mean expression", low = "steelblue3", high = "indianred4") +
  theme(axis.title.x = element_blank(),
        axis.text.x = element_blank(),
        axis.ticks.x = element_blank())
```



These are the top 10 most expressed genes based on their average expression value as seen in ALL and AML patients in the training dataset.

3.2.2 Top 10 highest gene expressions

```
# ...in ALL patients
all_top_genes <- train_set %>%
  filter(cancer_type == "ALL") %>%
  gather("gene", "value", - c(patient, cancer_type)) %>%
  top_n(10, value) %>%
  ggplot(aes(patient, value)) +
  geom_point(aes(col = gene)) +
  xlab("ALL-patient") +
  ylab("Max. expressed value") +
  scale_color_brewer("Gene Accession Numner", palette = "Dark2") +
  ggtitle("Top 10 highest gene expressions")

# ...in AML patients
aml_top_genes <- train_set %>%
  filter(cancer_type == "AML") %>%
  gather("gene", "value", - c(patient, cancer_type)) %>%
  top_n(10, value) %>%
  ggplot(aes(patient, value)) +
  geom_point(aes(col = gene)) +
  xlab("AML-patient") +
  ylab("Max. expressed value") +
  scale_color_brewer("Gene Accession Numner", palette = "Dark2")

grid.arrange(all_top_genes, aml_top_genes, ncol = 1)
```



From the plot above, it is clearly seen that in both cancer types many genes are mutually expressed in high levels, but there are genes that are expressed higher only in ALL patients, such as the gene with "AFFX-HSAC07/X00351_5_at" accession number in patients 6 and 23, and others that are expressed higher only in AML patients, such as the gene with "AFFX-HUMGAPDH/M33197_5_at" in patient 36.

4 Model Training

4.1 Model preparation

Different kinds of machine learning algorithms are going to be implemented, measured and compared, in order to pick the most efficient ones for the construction of an esemble algorithm.

Loading needed libraries:

```r
if(!require(naivebayes)) install.packages("naivebayes")
if(!require(kernlab)) install.packages("kernlab")
if(!require(RSNNS)) install.packages("RSNNS")
if(!require(deepnet)) install.packages("deepnet")
if(!require(caTools)) install.packages("caTools")
if(!require(gbm)) install.packages("gbm")
if(!require(C50)) install.packages("C50")
if(!require(plyr)) install.packages("plyr")
```

Note: some libraries will mask some needed functions from libraries already loaded, such as dplyr's and caret's functions. Following code specifies which masked functions are used.

Model preparation:

The training set will be used for the model creation, where the validation set will be used to accurately measure the final model's efficiency. Thus the training set is to be partitioned into two subsets:

- train subset: comprising of 85% of original training dataset's entries
- test subset: comprising of the remaining 15%

```r
train_canc <- train_set[, !(colnames(train_set) == "patient")]

set.seed(1998, sample.kind = "Rounding")
test_index <- createDataPartition(train_canc$cancer_type,
                                  p = 0.15, list = FALSE)
test_canc <- train_canc[test_index, ] %>%
  mutate(cancer_type = as.factor(cancer_type))
train_canc <- train_canc[-test_index, ] %>%
  mutate(cancer_type = as.factor(cancer_type))
```

Note 1: the subsets had their "patient" columns dropped for easiness in model training Note 2: a seed was set for the recreation of the code

Setup metric and control:

Metric used for the tuning parameter's selection will be the accuracy and control is chosen as 10-fold cross-validation with 3 separate repeats:

```r
control <- trainControl(method = "repeatedcv",
                        number = 10, repeats = 3)
metric <- "Accuracy"
```

4.2 Model setup

4.2.1 Naive Bayes

```
set.seed(1998, sample.kind = "Rounding")

fit.naiveBayes <-caret::train(cancer_type ~ .,
                              data = train_canc,
                              method = "naive_bayes",
                              trControl = control,
                              metric = metric)

preds.naiveBayes <- predict(fit.naiveBayes, test_canc)

caret::confusionMatrix(preds.naiveBayes, test_canc$cancer_type)$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##     1.00000000     1.00000000     0.59038360     1.00000000     0.71428571
## AccuracyPValue  McnemarPValue
##     0.09486451            NaN
```

4.2.2 k-Nearest Neighbors

```
set.seed(1998, sample.kind = "Rounding")

fit.knn <- caret::train(cancer_type ~ .,
                        data = train_canc,
                        method = "knn",
                        trControl = control,
                        metric = metric,
                        tuneGrid = data.frame(k = seq(5, 31, 2)))

fit.knn$bestTune
```

```
##   k
## 1 5
```

```
preds.knn <- predict(fit.knn, test_canc)

caret::confusionMatrix(preds.knn, test_canc$cancer_type)$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##     1.00000000     1.00000000     0.59038360     1.00000000     0.71428571
## AccuracyPValue  McnemarPValue
##     0.09486451            NaN
```

4.2.3 Support Vector Machines

```r
set.seed(1998, sample.kind = "Rounding")

fit.svm <- caret::train(cancer_type ~ .,
                        data = train_canc,
                        method = "svmRadial",
                        trControl = control,
                        metric = metric)

preds.svm <- predict(fit.svm, test_canc)

caret::confusionMatrix(preds.svm, test_canc$cancer_type)$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.7142857      0.0000000      0.2904209      0.9633074      0.7142857
## AccuracyPValue  McnemarPValue
##      0.6792299      0.4795001
```

4.2.4 Multilayer Perceptrons

```r
set.seed(1998, sample.kind = "Rounding")

fit.mlp <- caret::train(cancer_type ~ .,
                        data = train_canc,
                        method = "mlp",
                        trControl = control,
                        metric = metric)

preds.mlp <- predict(fit.mlp, test_canc)

caret::confusionMatrix(preds.mlp, test_canc$cancer_type)$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.7142857      0.0000000      0.2904209      0.9633074      0.7142857
## AccuracyPValue  McnemarPValue
##      0.6792299      0.4795001
```

4.2.5 Deep Neural Network

```r
set.seed(1998, sample.kind = "Rounding")

fit.dnn <- caret::train(cancer_type ~ .,
                        data = train_canc,
                        method = "dnn",
                        trControl = control,
                        metric = metric)

preds.dnn <- predict(fit.dnn, test_canc)

caret::confusionMatrix(preds.dnn, test_canc$cancer_type)$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.7142857      0.0000000      0.2904209      0.9633074      0.7142857
## AccuracyPValue  McnemarPValue
##      0.6792299      0.4795001
```

4.2.6 Generalized Linear Model

```
set.seed(1998, sample.kind = "Rounding")

fit.glm <- caret::train(cancer_type ~ .,
                        data = train_canc,
                        method = "glm",
                        trControl = control,
                        metric = metric)

preds.glm <- predict(fit.glm, test_canc)

caret::confusionMatrix(preds.glm, test_canc$cancer_type)$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##     0.42857143    -0.40000000     0.09898828     0.81594843     0.71428571
## AccuracyPValue  McnemarPValue
##     0.97672496     1.00000000
```

4.2.7 Boosted Logistic Regression

```
set.seed(1998, sample.kind = "Rounding")

fit.lb <- caret::train(cancer_type ~ .,
                       data = train_canc,
                       method = "LogitBoost",
                       trControl = control,
                       metric = metric)

preds.lb <- predict(fit.lb, test_canc)

caret::confusionMatrix(preds.lb, test_canc$cancer_type)$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##     1.00000000     1.00000000     0.59038360     1.00000000     0.71428571
## AccuracyPValue  McnemarPValue
##     0.09486451            NaN
```

4.2.8 Stochastic Gradient Boosting

```
set.seed(1998, sample.kind = "Rounding")
```

```
gbm_grid <- expand.grid(
  n.trees = 5,
  interaction.depth = 3,
  shrinkage = 0.03,
  n.minobsinnode = 2)

fit.gbm <- caret::train(cancer_type ~ .,
                        data = train_canc,
                        method = "gbm",
                        trControl = control,
                        metric = metric,
                        tuneGrid = gbm_grid,
                        verbose = FALSE)

preds.gbm <- predict(fit.gbm, test_canc)

caret::confusionMatrix(preds.gbm, test_canc$cancer_type)$overall
```

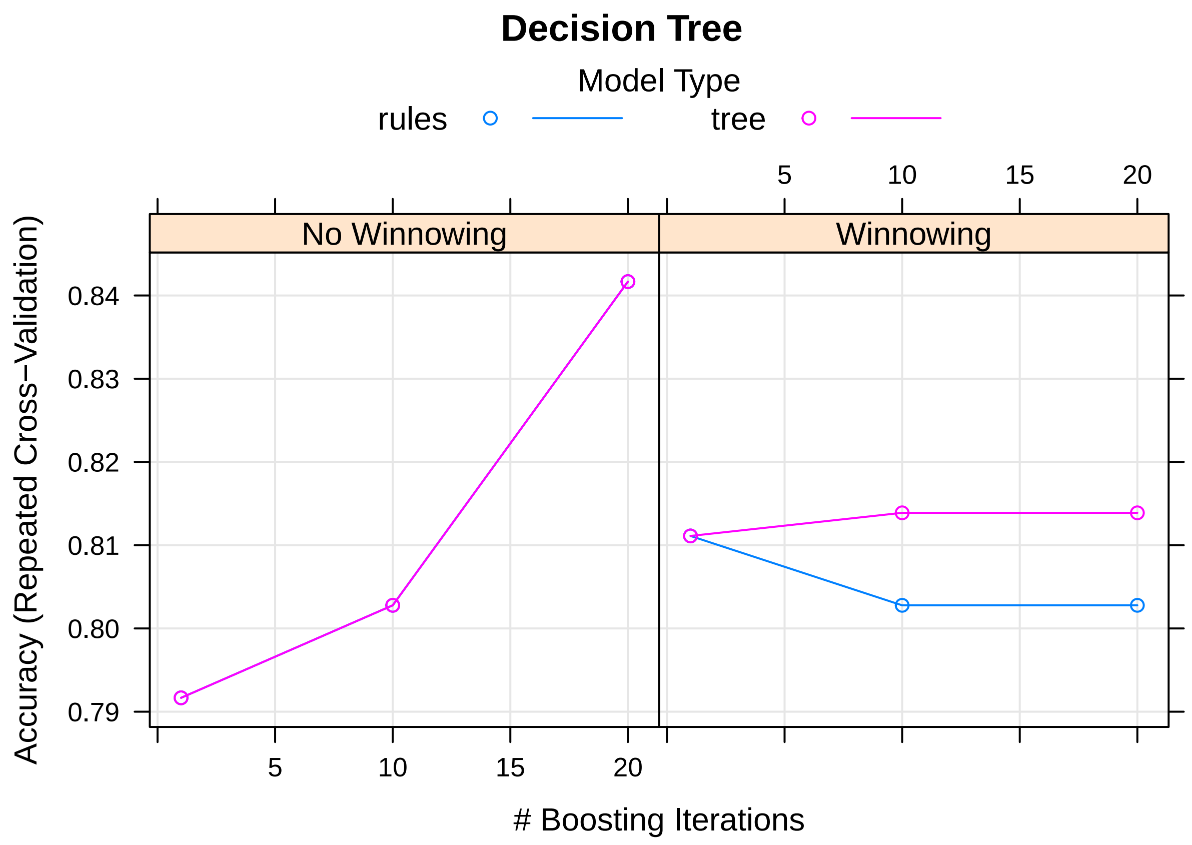
```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.7142857      0.0000000      0.2904209      0.9633074      0.7142857
## AccuracyPValue  McnemarPValue
##      0.6792299      0.4795001
```

4.2.9 C5.0 (Decision Tree algorithm)

```
set.seed(1998, sample.kind = "Rounding")

fit.c5 <- caret::train(cancer_type ~ .,
                       data = train_canc,
                       method = "C5.0",
                       trControl = control,
                       metric = metric,
                       verbose = FALSE)

preds.c5 <- predict(fit.c5, test_canc)

caret::confusionMatrix(preds.c5, test_canc$cancer_type)$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##     1.00000000     1.00000000     0.59038360     1.00000000     0.71428571
## AccuracyPValue  McnemarPValue
##     0.09486451            NaN
```

```
plot(fit.c5, main = "Decision Tree")
```

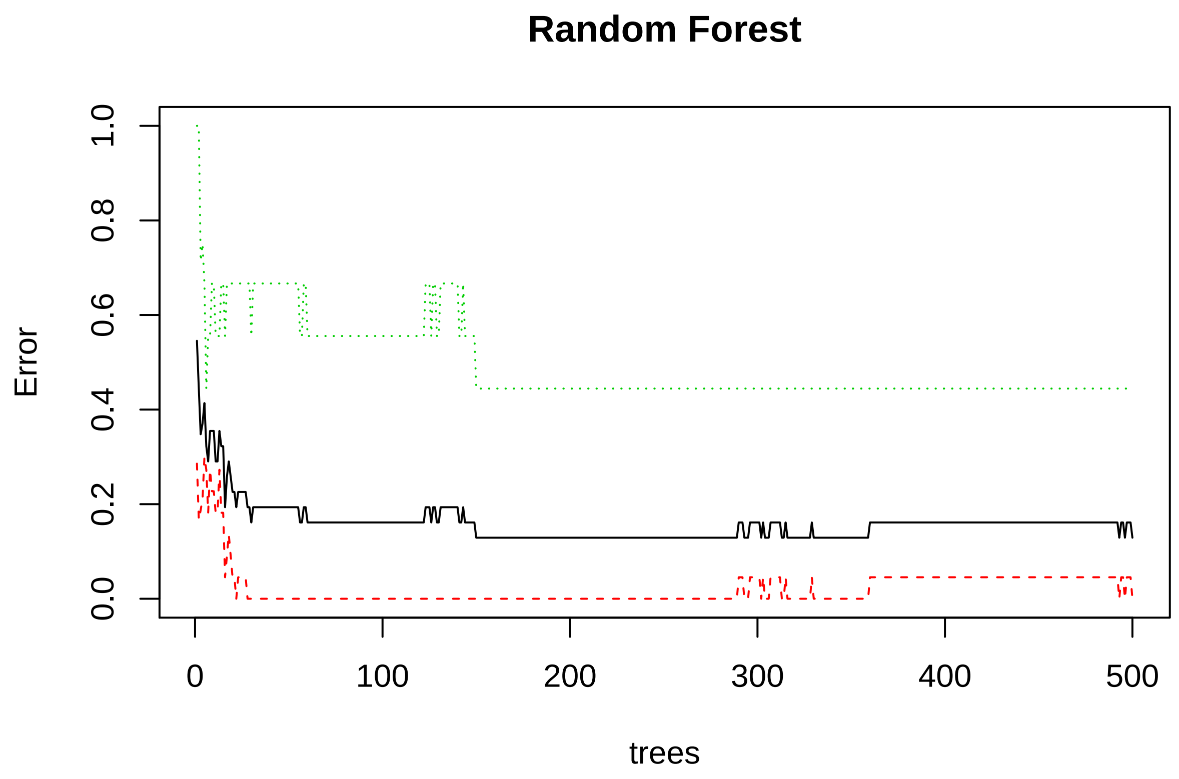


4.2.10 Random Forest

```
set.seed(1998, sample.kind = "Rounding")

fit.rf <- caret::train(cancer_type ~ .,
                       data = train_canc,
                       method = "rf",
                       trControl = control,
                       metric = metric,
                       verbose = FALSE)

preds.rf <- predict(fit.rf, test_canc)

caret::confusionMatrix(preds.rf, test_canc$cancer_type)$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##     1.00000000     1.00000000     0.59038360     1.00000000     0.71428571
## AccuracyPValue  McnemarPValue
##     0.09486451            NaN
```

```
plot(fit.rf$finalModel, main = "Random Forest")
```



4.2.11 K-means Clustering

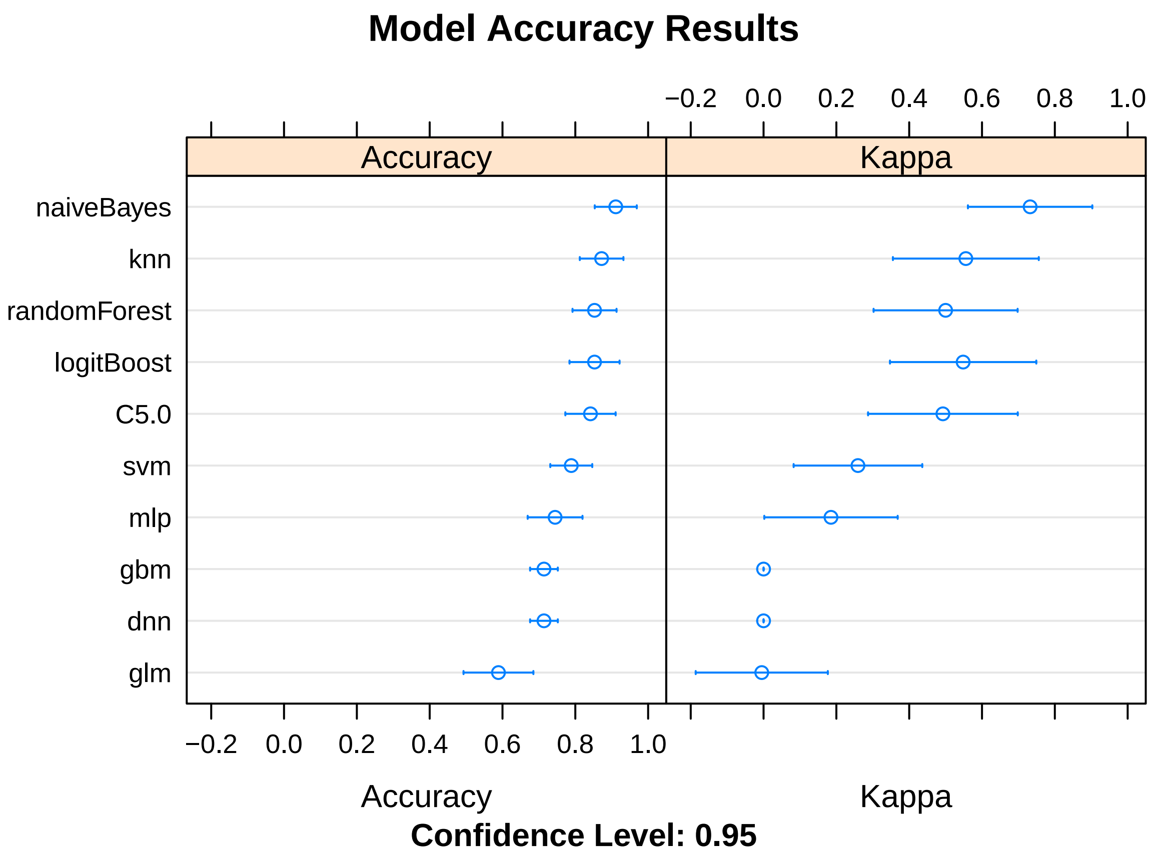
```
predict_kmeans <- function(x, k) {
  centers <- k$centers    # extract cluster centers

  # calculate distance to cluster centers
  distances <- sapply(1:nrow(x), function(i){
    apply(centers, 1, function(y) dist(rbind(x[i,], y)))
  })
  max.col(-t(distances))  # select cluster with min distance to center
}

train_canc_m <- as.matrix(train_canc[, -1])
test_canc_m <- as.matrix(test_canc[, -1])

set.seed(1998, sample.kind = "Rounding")
k <- kmeans(train_canc_m, centers = 2)
kmeans_preds <- ifelse(predict_kmeans(test_canc_m, k) == 1, "ALL", "AML")

caret::confusionMatrix(as.factor(kmeans_preds), as.factor(test_canc$cancer_type))$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##     0.42857143     0.12500000     0.09898828     0.81594843     0.71428571
## AccuracyPValue  McnemarPValue
##      0.97672496     0.13361440
```

This algorithm is immediately rejected due its low accuracy (<50%).

4.2.12 Model results

```r
results <- resamples(list(naiveBayes = fit.naiveBayes,
                          knn = fit.knn,
                          svm = fit.svm,
                          mlp = fit.mlp,
                          dnn = fit.dnn,
                          glm = fit.glm,
                          logitBoost = fit.lb,
                          gbm = fit.gbm,
                          C5.0 = fit.c5,
                          randomForest = fit.rf))

dotplot(results, main = "Model Accuracy Results")
```



It is obvious that first 5 algorithms perform extremely well considering their accuracies, and their Kappa measurements. Nevertheless, the least efficient algorithms that will not be included in the ensemble algorithm will be the last three. They are excluded due to the fact that their Kappa measurements is equal to zero (gbm, dnn) or it can take negative values (glm).

As Dr. Stelios Kampakis describes: "[. . .] Cohen's kappa is always less than or equal to 1. Values of 0 or less, indicate that the classifier is useless. There is no standardized way to interpret its values. Landis and Koch (1977) provide a way to characterize values. According to their scheme a value < 0 is indicating no agreement , 0–0.20 as slight, 0.21–0.40 as fair, 0.41–0.60 as moderate, 0.61–0.80 as substantial, and 0.81–1 as almost perfect agreement. [. . .]".

4.2.13 Ensemble

```
ensemble <- cbind(knn = preds.knn == "ALL",
                  naiveBayes = preds.naiveBayes == "ALL",
                  randomForest = preds.rf == "ALL",
                  logitBoost = preds.lb == "ALL",
                  C5.0 = preds.c5 == "ALL",
                  svm = preds.svm == "ALL",
                  mlp = preds.mlp == "ALL")

preds.ensemble <- ifelse(rowMeans(ensemble) > 0.5, "ALL", "AML")

caret::confusionMatrix(as.factor(preds.ensemble),
                       test_canc$cancer_type)$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##     1.00000000     1.00000000     0.59038360     1.00000000     0.71428571
## AccuracyPValue  McnemarPValue
##     0.09486451            NaN
```

Our final model seems to work extremely well over the test subset of the original dataset. Now, it is ready to be tested over the validation set.

4.2.14 Final model

Every training model included in the ensemble algorithm has to predict the validation's patients' cancer type:

```
preds.knn <- predict(fit.knn, validation)
results <- data.frame(Method = "k-Nearest Neighbors",
                      Accuracy = caret::confusionMatrix(preds.knn, validation$cancer_type)$overall["Accu

preds.naiveBayes <- predict(fit.naiveBayes, validation)
results <- dplyr::bind_rows(results,
                            data.frame(Method = "Naive Bayes",
                                       Accuracy = caret::confusionMatrix(preds.naiveBayes, validation$ca

preds.rf <- predict(fit.rf, validation)
results <- dplyr::bind_rows(results,
                            data.frame(Method = "Random Forest",
                                       Accuracy = caret::confusionMatrix(preds.rf, validation$cancer_typ

preds.lb <- predict(fit.lb, validation)
results <- dplyr::bind_rows(results,
                            data.frame(Method = "Boosted Logistic Regression",
                                Accuracy = caret::confusionMatrix(preds.lb, validation$cancer_ty

preds.c5 <- predict(fit.c5, validation)
results <- dplyr::bind_rows(results,
                            data.frame(Method = "C5.0 (Decision Tree Algorithm)",
                                       Accuracy = caret::confusionMatrix(preds.c5, validation$cancer_ty

preds.svm <- predict(fit.svm, validation)
results <- dplyr::bind_rows(results,
                            data.frame(Method = "Support Vector Machines",
                                       Accuracy = caret::confusionMatrix(preds.svm, validation$cancer_t

preds.mlp <- predict(fit.mlp, validation)
results <- dplyr::bind_rows(results,
                            data.frame(Method = "Multilayer Perceptons",
                                       Accuracy = caret::confusionMatrix(preds.mlp, validation$cancer_t
```

The results table was overwritten to hold the new predicted accuracy per method used.

```
ensemble <- cbind(knn = preds.knn == "ALL",
                  naiveBayes = preds.naiveBayes == "ALL",
                  randomForest = preds.rf == "ALL",
                  logitBoost = preds.lb == "ALL",
                  C5.0 = preds.c5 == "ALL",
                  svm = preds.svm == "ALL",
                  mlp = preds.mlp == "ALL")

preds.ensemble <- ifelse(rowMeans(ensemble) > 0.5, "ALL", "AML")

results <- dplyr::bind_rows(results,
                            data.frame(Method = "Ensemble",
                                       Accuracy = caret::confusionMatrix(as.factor(preds.ensemble), val
```

Lastly, the results of the final model and the sub-methods are shown in the following table:

```
results %>% knitr::kable()
```

Method	Accuracy
k-Nearest Neighbors	0.5882353
Naive Bayes	0.6470588
Random Forest	0.6176471
Boosted Logistic Regression	0.6176471
C5.0 (Decision Tree Algorithm)	0.5588235
Support Vector Machines	0.6470588
Multilayer Perceptons	0.5882353
Ensemble	0.6176471

Comparison with the actual data:

```r
data.frame(PatientID = actual_validation$patient,
           Predicted = preds.ensemble,
           Actual = actual_validation$cancer)
```

```
##    PatientID Predicted Actual
## 1         39       ALL    ALL
## 2         40       ALL    ALL
## 3         41       ALL    ALL
## 4         42       ALL    ALL
## 5         43       ALL    ALL
## 6         44       ALL    ALL
## 7         45       ALL    ALL
## 8         46       ALL    ALL
## 9         47       ALL    ALL
## 10        48       ALL    ALL
## 11        49       ALL    ALL
## 12        50       ALL    AML
## 13        51       ALL    AML
## 14        52       ALL    AML
## 15        53       ALL    AML
## 16        54       ALL    AML
## 17        55       ALL    ALL
## 18        56       ALL    ALL
## 19        57       ALL    AML
## 20        58       ALL    AML
## 21        59       ALL    ALL
## 22        60       AML    AML
## 23        61       AML    AML
## 24        62       AML    AML
## 25        63       ALL    AML
## 26        64       ALL    AML
## 27        65       AML    AML
## 28        66       ALL    AML
## 29        67       ALL    ALL
## 30        68       AML    ALL
## 31        69       ALL    ALL
## 32        70       AML    ALL
## 33        71       AML    ALL
## 34        72       ALL    ALL
```

It is obvious, that there are some minor number of ALL cases predicted as AML and vice versa.

5 Conclusion

The “Method-Accuracy” result table shows the calculated accuracies of the final ensemble model and its sub-methods. The results are not really satisfiable due to the small sample size of patients used in the training process of the final model (only 31 patients).

The analysis has plenty of room for further improvement:

- More machine learning algorithms can be used to achieve a higher-accuracy final model.
- In case of new leukemia patients to be added into the dataset, the code should be re-trained using all 72 already existed patient entries to better predict new entries in the dataset.
- Visualization can be also further improved.

6 Appendix - Enviroment

```
print("Operating System:")
```

```
## [1] "Operating System:"
```

```
version
```

```
##                _
## platform       x86_64-w64-mingw32
## arch           x86_64
## os             mingw32
## system         x86_64, mingw32
## status
## major          3
## minor          6.3
## year           2020
## month          02
## day            29
## svn rev        77875
## language       R
## version.string R version 3.6.3 (2020-02-29)
## nickname       Holding the Windsock
```