Coursework

Advanced Data Science (CMM536)

Amir Omicevic, 1615285@rgu.ac.uk

21 April 2018

Introduction

For the research part of the coursework, I have chosen to review "Stream mining for solar physics: Applications and implications for big solar data" conference paper. What first impressed me was just the name of the paper, that simply invited me to give it a quick read and assess its suitability for my coursework. As the paper deeply analyses clustering and classification techniques for data-stream mining that involves unprecedented data quantity, my first impression turned into a desire to read the paper fully and write my review of it. It had made a very interesting reading, along with the analysis of some of the references mentioned in the paper. What was really difficult, was fitting my reivew into a single A4 page, which I managed to complete having spent a considerable amount of time. But neverthelss, it was worth it. The paper is authored by Karl Battams, who works for Space Science Division, U.S. Naval Research Laboratory, Washington, D.C.

The paper can be downloaded from https://ieeexplore.ieee.org/document/7004400/

For the second part of the course, despite the well-meaning and respected advice from Eyad to choose a well-defined and purposeful dataset, after looking at a great number of datasets, I proceedeed with something that might not fully fit the aforementioned description. But it is a field that I am very interested in, I have read numerous articles and am currently reading books on the use of artificial intelligence in the finance industry in general, which is related to this exercise. [1][2].

I have chosen a data set that tries to classify whether the closing value of the stock market will go up or down depending on the 25 top news articles/headings current to that day. I realised straight away that this might not lead to a model that will be predicting the market movement with any significant success whatsoever, or even higher than the simple throw of a dice. A lot of inspiration has also come from an example of the stock market modelling in "An Introduction to Statistical Learning" by G. James [3].

But the curiosity was overwhelming and I wanted to give it my best shot to see how far I can take it. It gave me a platform to experiment, research and hopefully learn. As Andrew NG mentions in his new book "Machine Learning Yearning" (only a draft verision, released in batches), when faced with challenging tasks, it is best to start with a simple concept and build on that.[4] You can see the current draft at https://github.com/amiromicevic/CMM536AdvancedDataScience/blob/master/Ng_MLY02.pdf

I enjoyed it, despite being unable to deliver a model that I silently hoped I would, but neverthless, I learned a great deal, and have ideas on how to take it futher. The dataset is available at https://www.kaggle.com/aaron7sun/stocknews/data

Research - Problem Statement

In solar physics, we are now at a stage where our skill to capture data outweighs our ability to process it. The NASA Solar Dynamics Observatory launched in February 2010 generates 16MB of image data per second, totalling to 1.5Tb a day. This challenge renders present mining methods computationally impractical and requires new stream mining methods and reduction of data complexity.

Relevant Work

Heliophysics Event Knowledgebase online repository of solar events is part of SDO Event Detection System (EDS) effort that analyses solar data in near real-time using "feature-finding modules" contributed by the community. The Computer Aided CME Tracking, ARTEMIS and SEEDS algorithms detect coronal mass ejections, but the community is aware of the need for more advanced methods.

Discussion of the proposed methods

Traditional clustering methods requires the entire volume and are unable to compare clusters in streaming. CluStream algorithm separates the process into online micro-clustering and offline macro-clustering; the offline clustering utilises the summary statistics logged by the online clustering. Micro clusters can capture the evolution of the stream to flag concept drifts as well as provide a good summary of the incoming data.

In classification, one of the modifications to standard models utilizes Hoeffding Trees and Very Fast Decision Trees (VFDT). They balance the trade-off between accuracy and efficiency, learn in real-time, improve accuracy as data flows in, while real-time and offline results are very close. VFDT can delay decision and use user-defined error bounds to classify. The Adaptive Nearest Neighbor Classification for Data streams (ANNCAD) is another modified and data-size adaptable model in stream mining.

Time horizons approach retains data in memory to get accurate summaries, synopses, or classification models. Most basic one is simple sliding window, where a subset of the most recent data is kept, with the primary constraint being system resources. The pyramidal time window and Tilted timeframe models reduce granularity and summarise older data before exposing it to most recent data model. Synopsis methods such as Reservoir sampling and Haar wavelet manage large data streams well by reducing data size into chunks defined by coefficients vectors that can rebuild data at a later stage.

The Streaming Pattern dIscoverR in multIple Time-series (SPIRIT) algorithm is a framework that models the behaviour of a stream via principal components and weights and periodically updates them. This gives it a great ability for change detection via hidden variables that signify new trends in the stream. Algorithms must also handle concept drift for evolving data. Velocity density methods build total evolution of the data - a metric of the change of the nature of the data. They work well with micro-clustering and time horizons to detect short and long-term drifts.

Results Reported

SPIRIT scales up well to large streams, its computation demands are low: it only needs O(nk) floating point operations. It automatically estimates the number k of hidden variables to track, and it can automatically adapt, if k changes. It can plug in any forecasting method and handles missing values. Excellent performance.

Conclusion Discussion

There is no single one-stop solution, so a combination of the above algorithms with some novel approaches beyond simple concepts is required. The algorithms covered here are the most obvious applicable solutions to the problem, but we also must look for assistance and experience from big data communities.

Data Streams

As already mentioned, the chosen data set contains two data subsets which will be brought together into a sinle data set. The first data set contains the basic Dow Jones stock market information downloaded from Yahoo Finance. It has 7 features, as follows:

Date - date of the stock market record

Open - opening value of the stock market on the date

High - the highest value stock market reached on the date

Low - the lowest value stock market reached on the date

Close - the closing value of the stock markete on the date

Volume - the volume of shares traded on the date

Adj Close - adjusted closing value

The data set has 1989 rows of data.

The second data set contains the 25 top news items for each day from July 2016 going back to August 2008 obtained from Reddit social network. The 25 top news items were chosen by Reddit users. The data set has two features, as follows:

Date - date of the news

News - the actual news recorded on that date

The data is structured in such way that each date has 25 rows, where each row corresponds to a single news item. The data set has 73609 rows of data.

We will discuss the class distribution of the final data set once we have it built.

Data Exploration

Let us first load the required R packages.

```
library(tidyverse)      #Collection of R packages designed for data science
library(tidytext)       #Text mining for word processing and sentiment analysis
library(glue)           #An implementation of interpreted string literals, as in Python's Literal String
library(stringr)        #Cohesive set of functions designed to make working with strings

library(jsonlite)       #JSON parser and generator optimized for statistical data and the web
library(data.table)     #Fast processing of large amounts of data (i.e. 100GB in RAM)
library(RMOA)           #Massive Online Analysis for classification and regression models on streaming o
library(ROCR)           #Visualizing the Performance of Scoring Classifiers
library(stream)         #A framework for data stream modeling and data mining tasks such as clustering a

library(ROCR)            #Visualizing the Performance of Scoring Classifiers
library(caret)           #Streamline the process for creating predictive models
library(tm)              #Text mining package
library(SnowballC)       #Porter's word stemming algorithm
library(wordcloud)       #Word cloud library
library(RColorBrewer)    #Color schemes for maps and graphics
library(ggplot2)         #Plotting and graphics library
library(e1071)           #Naive Bayes implementation
library(gmodels)         #Cross table evaluation results amongst various model fitting tools
library(nnet)            #Feed-Forward Neural Networks and Multinomial Log-Linear Models
```

```
library(knitr)          #General-purpose tool for dynamic report generation in R

# Ensemble classification svm, slda, boosting, bagging, random forests, glmnet, decision trees,
# neural networks, maximum entropy
library(RTextTools)

# For graph and Rgraphviz, if not already installed, use the following 2 commands
# source("http://bioconductor.org/biocLite.R")
# biocLite(c("graph", "RBGL", "Rgraphviz"))

library(graph)          #A package to handle graph data structures
library(Rgraphviz)      #Provides plotting capabilities for R graph objects
```

The very first thing to do is to setup the working directory. We shall also turn off the warnings. It is ok to turn them off as all the scripts have been previosuly run with the warnings examined, so we do not want the PDF document to end up full of warnings.

```
options(warn=-1)

setwd('C:\\Users\\Amir\\Documents\\Coursework\\CMM536AdvancedDataScience')
```

Now we can load the news dataset as the R environment knows where to look for data files.

```
library(XLConnect)
news <- readWorksheetFromFile("RedditNewsExcel.xlsx", sheet=1, startRow = 1, endCol = 2)

# Swap the columns as it is very difficult in the R studio viewer to see the full content of the last c
news <- news[c("News", "Date")]
```

Let us now clean up the news data and show dimensions of the dataset.

```
# Remove the b's and backslashes
news$News <- gsub('b"|b\'|\\\\\|\\"', "", news$News)

# Remove punctuation except headline separators
news$News <- gsub("([<>])|[[:punct:]]", "\\1", news$News)

# Remove any dollar signs (they're special characters in R)
news$News <- gsub("\\$", "", news$News)

# Show the size of news data frame after cleanup
dim(news)
```

[1] 73615 5

Let us add two new features to the data frame to hold the sentiment values. The idea is to do the sentiment analysis for each news row to attempt to quantify the news items.

```
news["SentimentBing"] <- NA
news["SentimentAfinn"] <- NA
```

We shall use two sentiment lexicons and attempt to average out the final sentiment value, which will be set to either positive or negative.

```
newsRowsCount <- dim(news)[1]
for (i in 1:newsRowsCount)
{
  # Tokenize and remove any dollar signs (they're special characters in R)
```

```
  tokens <- data_frame(text = news$News[i]) %>% unnest_tokens(word, text)

  # First we use Bing Liu sentiment lexicon
  sentiment <- tokens %>%
    inner_join(get_sentiments("bing")) %>%
    count(sentiment) %>%
    spread(sentiment, n, fill = 0) #%>%

  news[i,3] <- ifelse(is.null(sentiment$negative), 'Pos', 'Neg')

  # Second time around we use Finn Årup Nielsen sentiment lexicon
  sentiment <- tokens %>%
    inner_join(get_sentiments("afinn")) %>%
    summarise(sentiment = sum(score))

  news[i,4] <- sentiment[1]
}
```

Let us have a look at the summary of SentimentAfinn column. This will help us determine some ordinal values for the final sentiment value.

```
summary(news$SentimentAfinn)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-25.000  -3.000  -1.000  -1.408   0.000  14.000
```

We can see that the mean of -1.408 indicates that the lexicon favours news to be of a negative sentiment. Only the 3rd quartile begins to show more positive semantics. The lowest value is -25, and the highest value is 14

Let us add now the new feature to the data frame to hold the final sentiment value. The ordinal values for the final sentiment will be calculated in the following way:

NegHigh where SentimentAfinn less or equal to -15.

NegMed where SentimentAfinn between -14 and -6 inclusive.

NegLow where SentimentAfinn between -5 and -1 inclusive.

PosLow where SentimentAfinn between 1 and 5 inclusive.

PosMed where SentimentAfinn between 6 and 10 inclusive.

PosHigh where SentimentAfinn greater than 10.

If SentimentAfinn = 0 it will be set to the value of PosLow if SentimentBing is Pos

If SentimentAfinn = 0 it will be set to the value of NegLow if SentimentBing is Neg

We are adding ordinal values in order to increase the meaning of the news items from only 2 class value (positive or negative) that will be provided to the classifier. Now is the time to populate the new features.

```
news["FinalSentiment"] <- NA

news$FinalSentiment[which(news$SentimentAfinn <= -15)] <- 'NegHigh'
news$FinalSentiment[which(news$SentimentAfinn > -15 & news$SentimentAfinn <= -6)] <- 'NegMed'
news$FinalSentiment[which(news$SentimentAfinn > -6 & news$SentimentAfinn < 0)] <- 'NegLow'

news$FinalSentiment[which(news$SentimentAfinn > 0 & news$SentimentAfinn <= 5)] <- 'PosLow'
news$FinalSentiment[which(news$SentimentAfinn > 5 & news$SentimentAfinn <= 10)] <- 'PosMed'
```

```
news$FinalSentiment[which(news$SentimentAfinn > 10)] <- 'PosHigh'

news$FinalSentiment[which(news$SentimentAfinn == 0 & news$SentimentBing == 'Pos')] <- 'PosLow'
news$FinalSentiment[which(news$SentimentAfinn == 0 & news$SentimentBing == 'Neg')] <- 'NegLow'
```

Let us now check that the FinalSentiment is fully populated without any gaps.

```
length(which(news$FinalSentiment != 'NegHigh' & news$FinalSentiment != 'NegMed'
             & news$FinalSentiment != 'NegLow'
             & news$FinalSentiment != 'PosLow'
             & news$FinalSentiment != 'PosMed' & news$FinalSentiment != 'PosHigh'
             & news$FinalSentiment != 'PosLow' & news$FinalSentiment != 'NegLow')) == 0
```

```
[1] TRUE
```

Now we shall load the DowJonesIndex data set to complete the creation of the final data set with all features.

```
dowJones <- read.csv("DowJonesIndex.csv")
dim(dowJones)
```

```
[1] 1989    7
```

```
# The column Close will be dropped as its purpose is served by the Adj.Close column
dowJones$Close <- NULL
```

Let us add the 25 news sentiment columns called News1 to News25. Then we shall populate them from news dataset.

```
for (i in 1:25)
{
  addNewCol = paste("dowJones$News",toString(i), "<- NA", collapse="", sep="")
  eval(parse(text = addNewCol))
}

# Let us now populate the news columns in dowJones from news dataset
dowJonesRowsCount <- dim(dowJones)[1]
for (i in 1:dowJonesRowsCount)
{
  date <- as.character(dowJones$Date[i])
  dowJones[i,7:31] <- news$FinalSentiment[which(news$Date == date)]
}
```

Perform a few checks (first, middle and last record) to ensure that dowJones is populated accurately.

```
dowJonesTest <- dowJones[1,7:31]
newsTest <- news$FinalSentiment[which(news$Date == '2016-07-01')]
all(dowJonesTest == newsTest)
```

```
[1] TRUE
```

```
dowJonesTest <- dowJones[1000,7:31]
newsTest <- news$FinalSentiment[which(news$Date == '2012-07-12')]
all(dowJonesTest == newsTest)
```

```
[1] TRUE
```

```
dowJonesTest <- dowJones[1989,7:31]
newsTest <- news$FinalSentiment[which(news$Date == '2008-08-08')]
all(dowJonesTest == newsTest)
```

```
[1] TRUE
```

Let us first check the distribution of the label, dowJones$Adj.Close. Then we shall add the final feature label to be set to 1 if Adj Close value rose or stayed as the same, and 0 when DJIA Adj Close value decreased.

```
# First, show the number of rows where value rose or stayed the same, 1078
length(which(dowJones$Adj.Close >= dowJones$Open))
```

```
[1] 1078
```

```
# Now show the number of rows where value decreased, 911
length(which(dowJones$Adj.Close < dowJones$Open))
```

```
[1] 911
```

```
# Create and populate label, the final class feature
dowJones$Label <- NA

dowJones$Label[which(dowJones$Adj.Close >= dowJones$Open)] <- 1
dowJones$Label[which(dowJones$Adj.Close < dowJones$Open)] <- 0
```

Quick check that label is populated correctly.

```
length(which(dowJones$Label == 1)) == length(which(dowJones$Adj.Close >= dowJones$Open))
```

```
[1] TRUE
```

```
length(which(dowJones$Label == 0)) == length(which(dowJones$Adj.Close < dowJones$Open))
```

```
[1] TRUE
```

Let us remove the date as it is not needed.

```
dowJones$Date <- NULL

# Last check of the date
sapply(dowJones, class)
```

```
       Open        High         Low      Volume   Adj.Close       News1
  "numeric"   "numeric"   "numeric"   "integer"   "numeric" "character"
      News2       News3       News4       News5       News6       News7
"character" "character" "character" "character" "character" "character"
      News8       News9      News10      News11      News12      News13
"character" "character" "character" "character" "character" "character"
     News14      News15      News16      News17      News18      News19
"character" "character" "character" "character" "character" "character"
     News20      News21      News22      News23      News24      News25
"character" "character" "character" "character" "character" "character"
      Label
  "numeric"
```

Let us convert Label and all News features to factors.

```
dowJones$Label <- as.factor(dowJones$Label)

for (i in 1:25)
{
  columnName = paste("dowJones$News",toString(i), sep="")
  factorColumn = paste(columnName, " <- as.factor(", columnName, ")", collapse="", sep="")
  eval(parse(text = factorColumn))
```

```
}

# Check that the conversion to factors worked
sapply(dowJones, class)
```

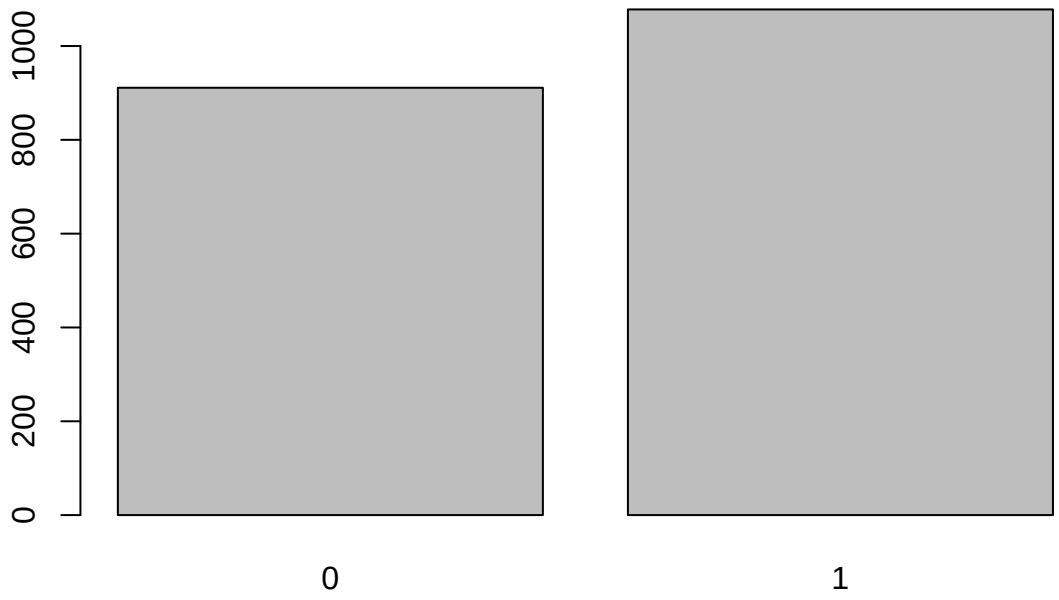
```
     Open      High       Low    Volume Adj.Close     News1     News2
"numeric" "numeric" "numeric" "integer" "numeric"  "factor"  "factor"
    News3     News4     News5     News6     News7     News8     News9
 "factor"  "factor"  "factor"  "factor"  "factor"  "factor"  "factor"
   News10    News11    News12    News13    News14    News15    News16
 "factor"  "factor"  "factor"  "factor"  "factor"  "factor"  "factor"
   News17    News18    News19    News20    News21    News22    News23
 "factor"  "factor"  "factor"  "factor"  "factor"  "factor"  "factor"
   News24    News25     Label
 "factor"  "factor"  "factor"
```

Data is now ready. It is clean, fully populated and we can commence the model building process. The objective is to predict Label as a consequence of the feature set that contains daily share totals and the top 25 news for the given date. Label can have two values, 1 or 0. If the closing price is greater or equal to the opening price, Label is set to 1, otherwise to 0.

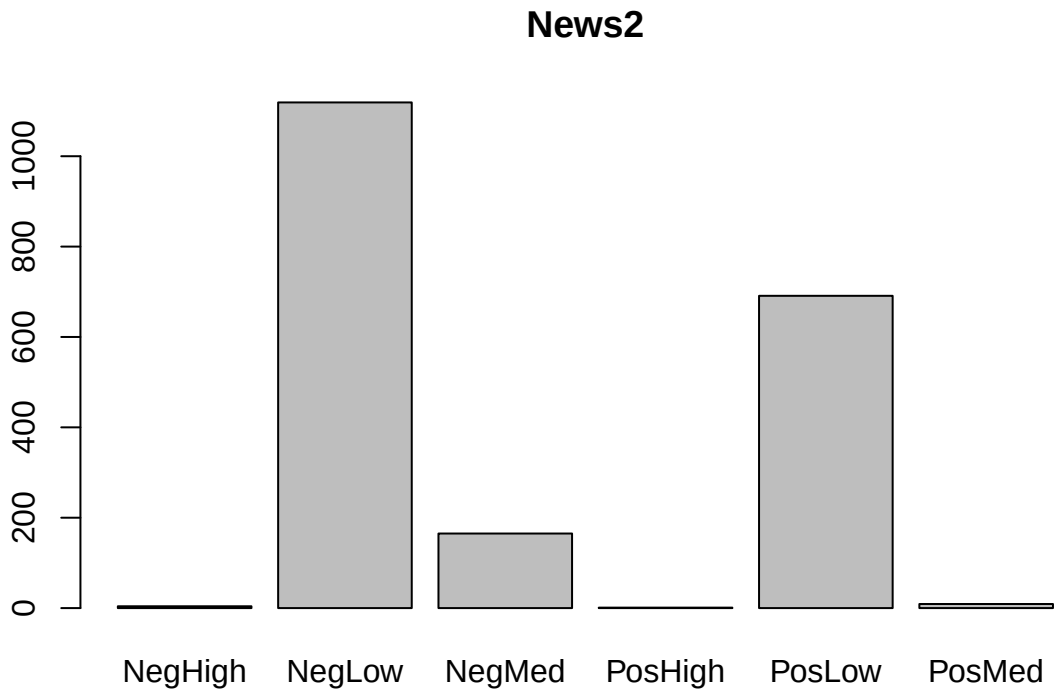
Build Classifier

We are now ready to build a classifier. But first, let us partition data into training and testing data sets, and then plot the label distribution.

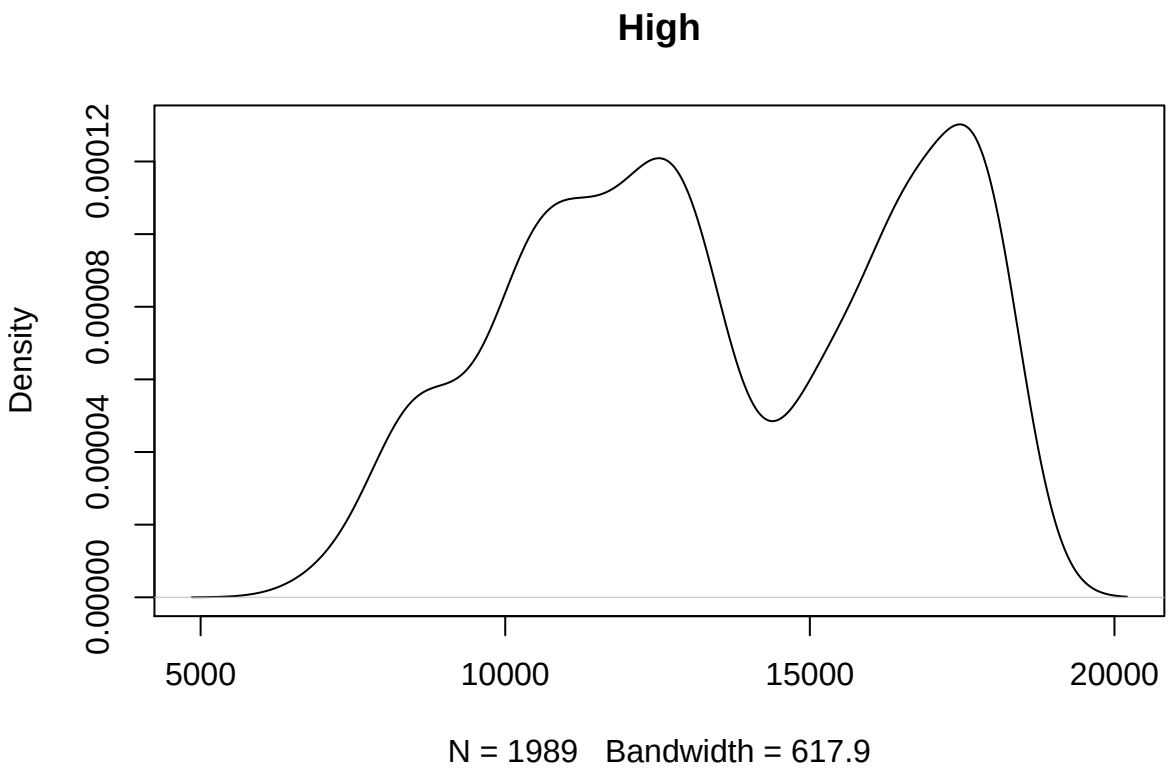
```
set.seed(7)
validationIndex <- createDataPartition(dowJones$Label, p=0.80, list=FALSE)

# select 20% of the data for validation
validationDataset <- dowJones[-validationIndex,]

# use the remaining 80% of data to training and testing the models
trainingDataset <- dowJones[validationIndex,]

barplot(summary(dowJones$Label))
```




```
plot(dowJones[,7], main=names(dowJones)[7])
```




```
plot(density(dowJones[,2]), main=names(dowJones)[2])
```



We can now evaluate algorithms, and we will use a few algorithms and then compare their results. 10-fold cross validation with 3 repeats will be used on the training data.

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

# Let us create a formula with the features that will be used for modelling
# First trial is with news features only
featuresToUse <- as.formula("Label~News1+News2+News3+News4+News5+News6+News7+News8+News9+News10+
                            News11+News12+News13+News14+News15+News16+News17+News18+News19+News20+
                            News21+News22+News23+News24+News25")
```

Now that we have features created as a formula, we can run our classifiers.

```
# LG Logistic Regression
set.seed(7)
fit.glm <- train(featuresToUse, data=trainingDataset, method="glm", metric=metric,
                 trControl=control, na.action=na.omit)

# GLMNET Regularized Logistic Regression
set.seed(7)
fit.glmnet <- train(featuresToUse, data=trainingDataset, method="glmnet", metric=metric,
                    trControl=control, na.action=na.omit)

# KNN k-Nearest Neighbours
set.seed(7)
fit.knn <- train(featuresToUse, data=trainingDataset, method="knn", metric=metric,
```

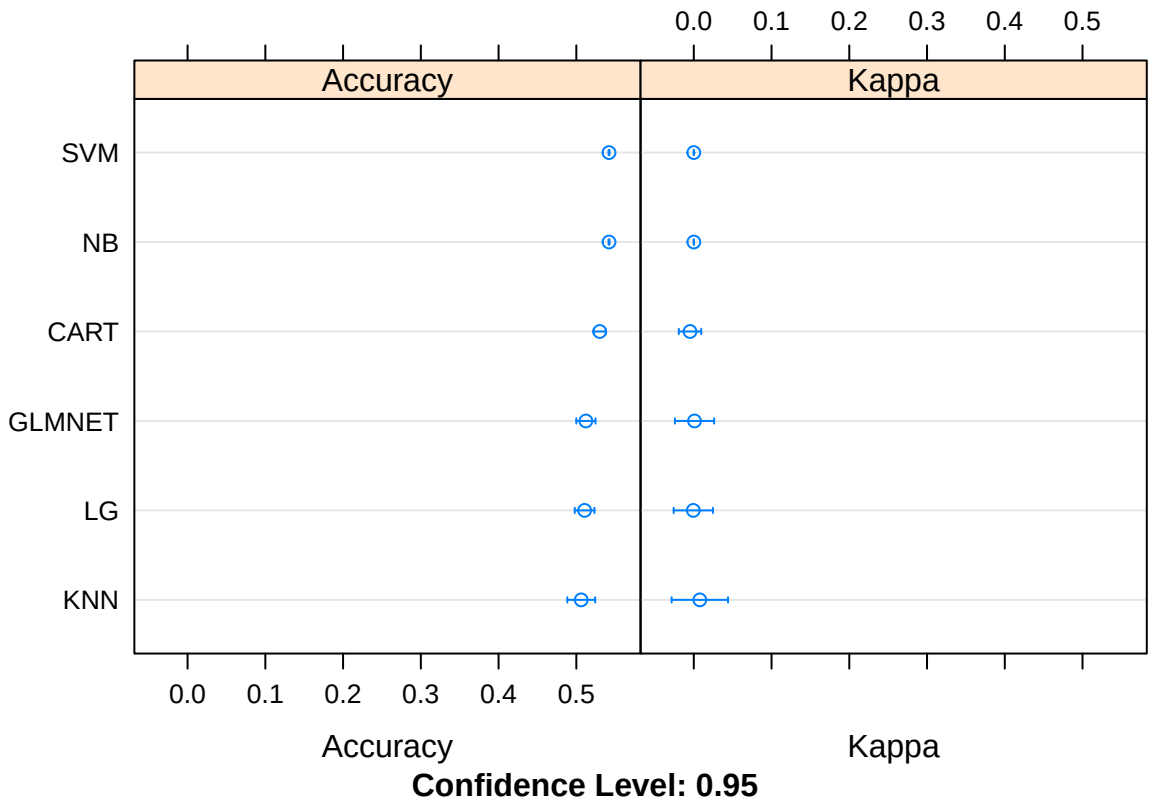
```
                trControl=control, na.action=na.omit)

# CART Classification and Regression Trees
set.seed(7)
fit.cart <- train(featuresToUse, data=trainingDataset, method="rpart", metric=metric,
                  trControl=control, na.action=na.omit)

# NB Naive Bayes
set.seed(7)
fit.nb <- train(featuresToUse, data=trainingDataset, method="nb", metric=metric,
                trControl=control, na.action=na.omit)

# SVM Support Vector Machines with Radial Basis Functions
set.seed(7)
fit.svm <- train(featuresToUse, data=trainingDataset, method="svmRadial", metric=metric,
                 trControl=control, na.action=na.omit)
```

Let us now compare the results.

```
resultsNews <- resamples(list(LG=fit.glm, GLMNET=fit.glmnet, KNN=fit.knn,
                              CART=fit.cart, NB=fit.nb, SVM=fit.svm))
summary(resultsNews)
```

```
Call:
summary.resamples(object = resultsNews)

Models: LG, GLMNET, KNN, CART, NB, SVM
Number of resamples: 30

Accuracy
         Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
LG     0.4438  0.4898 0.5126 0.5107  0.5289 0.5660    0
GLMNET 0.4403  0.4945 0.5110 0.5123  0.5275 0.5786    0
KNN    0.4188  0.4755 0.5110 0.5063  0.5393 0.5938    0
CART   0.4875  0.5157 0.5409 0.5301  0.5409 0.5849    0
NB     0.5409  0.5409 0.5409 0.5421  0.5438 0.5443    0
SVM    0.5409  0.5409 0.5409 0.5421  0.5438 0.5443    0

Kappa
          Min.  1st Qu.    Median       Mean 3rd Qu.     Max. NA's
LG     -0.1284 -0.04490  0.004970 -0.0006469 0.03501 0.116200    0
GLMNET -0.1400 -0.03186 -0.002263  0.0009320 0.02411 0.138200    0
KNN    -0.1728 -0.05744  0.017510  0.0077580 0.07122 0.182100    0
CART   -0.0741 -0.01253  0.000000 -0.0048330 0.00000 0.142900    0
NB      0.0000  0.00000  0.000000  0.0000745 0.00000 0.002235    0
SVM     0.0000  0.00000  0.000000  0.0000000 0.00000 0.000000    0
```

```
dotplot(resultsNews)
```



Now we can evaluate boosting and bagging classifiers.

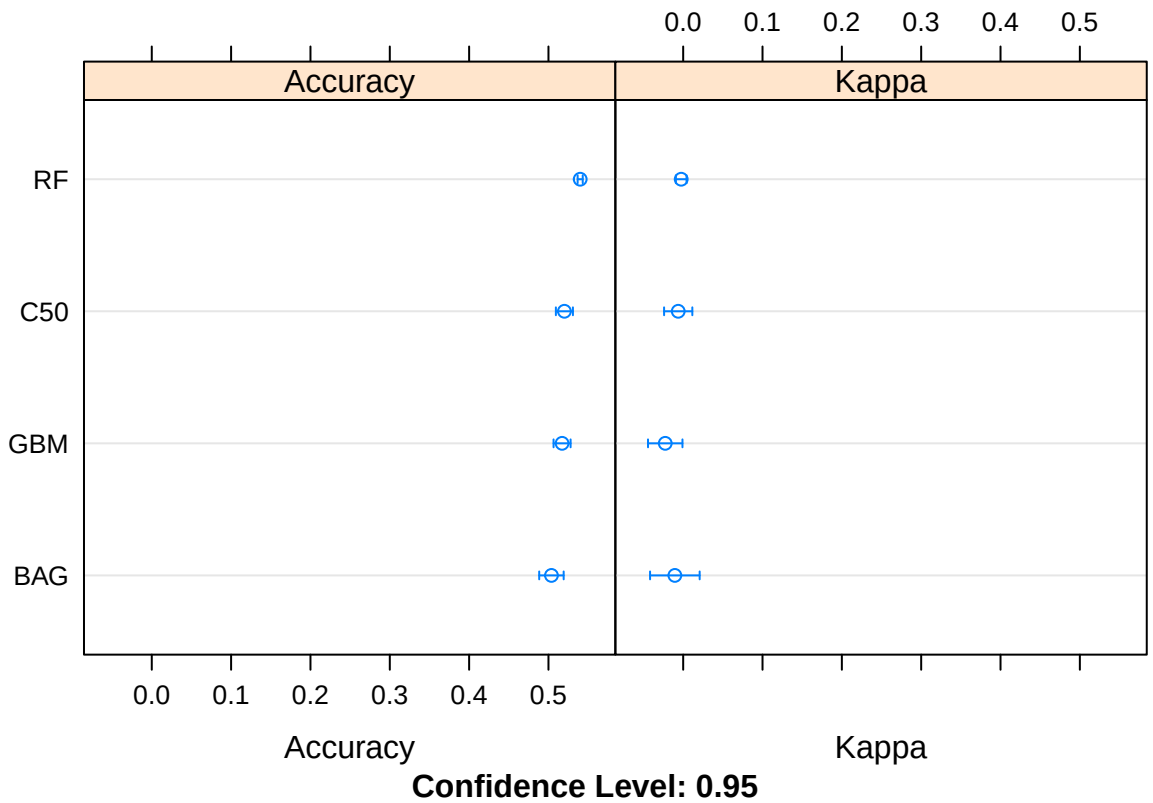
```
# Bagged CART
set.seed(7)
fit.treebag <- train(featuresToUse, data=trainingDataset, method="treebag", metric=metric,
                     trControl=control, na.action=na.omit)

# Random Forest
set.seed(7)
fit.rf <- train(featuresToUse, data=trainingDataset, method="rf", metric=metric,
                trControl=control, na.action=na.omit)

# Stochastic Gradient Boosting
set.seed(7)
fit.gbm <- train(featuresToUse, data=trainingDataset, method="gbm", metric=metric,
                 trControl=control, verbose=FALSE, na.action=na.omit)

# C5.0
set.seed(7)
fit.c50 <- train(featuresToUse, data=trainingDataset, method="C5.0", metric=metric,
                 trControl=control, na.action=na.omit)
```

Again, let us compare the results of the 10-fold cross validation.

```
ensembleResultsNews <- resamples(list(BAG=fit.treebag, RF=fit.rf, GBM=fit.gbm, C50=fit.c50))
summary(ensembleResultsNews)
```

```
Call:
summary.resamples(object = ensembleResultsNews)

Models: BAG, RF, GBM, C50
Number of resamples: 30

Accuracy
      Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
BAG 0.4312  0.4748 0.5000 0.5038  0.5244 0.5875    0
RF  0.5250  0.5324 0.5409 0.5400  0.5442 0.5570    0
GBM 0.4465  0.5031 0.5157 0.5172  0.5359 0.5723    0
C50 0.4465  0.5039 0.5235 0.5201  0.5430 0.5812    0

Kappa
        Min.  1st Qu.     Median      Mean  3rd Qu.    Max. NA's
BAG -0.15260 -0.07228 -0.0177900 -0.010460 0.030010 0.16310    0
RF  -0.03737 -0.02001  0.0000000 -0.002471 0.004177 0.03016    0
GBM -0.16760 -0.05561 -0.0216100 -0.022620 0.009354 0.08990    0
C50 -0.14280 -0.02501 -0.0009335 -0.006251 0.013040 0.10790    0
```

```
dotplot(ensembleResultsNews)
```



Top 3 best performing models are NB = 0.5420, SVM = 0.5420 and RF = 0.5403 Let us now predict against validation data using these 3 models.

```
# NB=fit.nb
set.seed(7)
```

```r
predictionsNB <- predict(fit.nb, newdata=validationDataset)
confusionMatrix(predictionsNB, validationDataset$Label, positive = "1")
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0   0   0
         1 182 215

               Accuracy : 0.5416
                 95% CI : (0.4911, 0.5914)
    No Information Rate : 0.5416
    P-Value [Acc > NIR] : 0.5206

                  Kappa : 0
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 1.0000
            Specificity : 0.0000
         Pos Pred Value : 0.5416
         Neg Pred Value :    NaN
             Prevalence : 0.5416
         Detection Rate : 0.5416
   Detection Prevalence : 1.0000
      Balanced Accuracy : 0.5000

       'Positive' Class : 1
```

```r
# SVM=fit.svm
set.seed(7)
predictionsSVM <- predict(fit.svm, newdata=validationDataset)
confusionMatrix(predictionsSVM, validationDataset$Label, positive = "1")
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0   0   0
         1 182 215

               Accuracy : 0.5416
                 95% CI : (0.4911, 0.5914)
    No Information Rate : 0.5416
    P-Value [Acc > NIR] : 0.5206

                  Kappa : 0
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 1.0000
            Specificity : 0.0000
         Pos Pred Value : 0.5416
         Neg Pred Value :    NaN
```

```
             Prevalence : 0.5416
         Detection Rate : 0.5416
   Detection Prevalence : 1.0000
      Balanced Accuracy : 0.5000

       'Positive' Class : 1
```

```
# RF=fit.rf
set.seed(7)
predictionsRF <- predict(fit.rf, newdata=validationDataset)
confusionMatrix(predictionsRF, validationDataset$Label, positive = "1")
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0   0   1
         1 182 214

               Accuracy : 0.539
                 95% CI : (0.4886, 0.5889)
    No Information Rate : 0.5416
    P-Value [Acc > NIR] : 0.5606

                  Kappa : -0.005
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.9953
            Specificity : 0.0000
         Pos Pred Value : 0.5404
         Neg Pred Value : 0.0000
             Prevalence : 0.5416
         Detection Rate : 0.5390
   Detection Prevalence : 0.9975
      Balanced Accuracy : 0.4977

       'Positive' Class : 1
```

Results of the predictions are very low, as expected, but above 50% rate, so the models can outdo random guessing, just. Let us now try to tune the SVM to see if we can increase the final accuracy.

```
grid <- expand.grid(sigma = c(0,0.01, 0.02, 0.025, 0.03, 0.04, 0.05, 0.06, 0.07,0.08, 0.09,
                              0.1, 0.25, 0.5, 0.75,0.9),
                              C = c(0,0.01, 0.05, 0.1, 0.25, 0.5, 0.75,1, 1.5, 2,5))
fit.svmTuned <- train(featuresToUse, data=trainingDataset, method="svmRadial", metric=metric,
                 trControl=control, na.action=na.omit, tuneGrid = grid)

fit.svmTuned$finalModel

# Let us now use the tuned SVM to predict
set.seed(7)
predictionsSVMTuned <- predict(fit.svmTuned, newdata=validationDataset)
confusionMatrix(predictionsSVMTuned, validationDataset$Label, positive = "1")
```

Accuracy has actually gone down, so the SVM tuning did not bring on any improvements whatsoever. The use of the algorithms in this part of the exercise has been inspired by the examples from "An Introduction to Statistical Learning" by G. James [3] and "MACHINE LEARNING MASTERY WITH R, Binary Classification Machine Learning Case Study Project" by J. Brownlee [6] and "MASTER MACHINE LEARNING ALGORITHMS" by J.Brownlee [7].

This experimental work has been leading us to the final exercise that will attempt to predict the movement of the stock market closing value in respect of the opening value on the same day. We also need to be mindful that this model has to work for streaming classification. So we shall use dowJones to craft a new dataset dowJonesNext. It will have the following features:

NegHigh, NegMed, NegLow, PosLow, PosMed, PosHigh, DayMinus3Label, DayMinus3CORatio, DayMinus2Label, DayMinus2CORatio,DayMinus1Label, DayMinus1CORatio, Label

The first 6 features NegHigh to PosHigh will be the sum aggregates of the corresponding feature values for each observation/record. DayMinues3Label will be the Label value from the (today - 3 days) observation, and DayMinues3CORatio will be Adj.Close/Open from the (today - 3 days) observation etc... The reason for this is that we want to widen the feature scope to try to capture as much model behaviour as possible. The features from the previous 3 days consisting of the label and the ratio between the closing and opening stock market value serve as an addition to the news features that already proved to be insufficient for any serious modelling when being used in isolation.

Let us now create the new dataset and populate it. We shall call it dowJonesNext to represen the next step in our exploration.

```
dowJonesNext <- data.frame(NegHigh=integer(), NegMed=integer(), NegLow=integer(),
                           PosLow=integer(), PosMed=integer(), PosHigh=integer(),
                           DayMinus1Label = character(), DayMinus1CORatio = numeric(),
                           DayMinus2Label = character(), DayMinus2CORatio = numeric(),
                           DayMinus3Label = character(), DayMinus3CORatio = numeric(),
                           Label = character(), stringsAsFactors = FALSE)

newsAggr <- c('NegHigh', 'NegMed', 'NegLow', 'PosLow', 'PosMed', 'PosHigh')
rowsCount <- nrow(dowJones)
for (i in 1:rowsCount)
{
  for (j in 1:6)
  {
    dowJonesNext[i,j] <- length(which(dowJones[i,6:30] == newsAggr[j]))
  }

  if(i <= rowsCount - 3)
  {
    dowJonesNext$DayMinus1Label[i] <- as.character(levels(dowJones$Label[i+1])[dowJones$Label[i+1]])
    dowJonesNext$DayMinus1CORatio[i] <- dowJones$Adj.Close[i+1] / dowJones$Open[i+1]

    dowJonesNext$DayMinus2Label[i] <- as.character(levels(dowJones$Label[i+2])[dowJones$Label[i+2]])
    dowJonesNext$DayMinus2CORatio[i] <- dowJones$Adj.Close[i+2] / dowJones$Open[i+2]

    dowJonesNext$DayMinus3Label[i] <- as.character(levels(dowJones$Label[i+3])[dowJones$Label[i+3]])
    dowJonesNext$DayMinus3CORatio[i] <- dowJones$Adj.Close[i+3] / dowJones$Open[i+3]
  }
}

dowJonesNext$Label <- as.character(levels(dowJones$Label)[dowJones$Label])

# Let us remove the last 3 records as they will not have values for the previous 3 days
dowJonesNext <- dowJonesNext[1:(rowsCount-3),]

# Data is there, so let us convert two label columns into factors
dowJonesNext$DayMinus1Label <- as.factor(dowJonesNext$DayMinus1Label)
dowJonesNext$DayMinus2Label <- as.factor(dowJonesNext$DayMinus2Label)
```

```
dowJonesNext$DayMinus3Label <- as.factor(dowJonesNext$DayMinus3Label)
dowJonesNext$Label <- as.factor(dowJonesNext$Label)
```

Let us do a quick check that aggregations are right 25 news columns x 1986 observations = 49650

```
sum(rowSums(dowJonesNext[1:1986, 1:6])) == 49650
```

```
[1] TRUE
```

And yet again, it is time now to partition the dataset into the training and testing data sets. For training, we will continue using the 10-fold cross validation control.

```
set.seed(7)
validationIndex <- createDataPartition(dowJonesNext$Label, p=0.80, list=FALSE)

# select 20% of the data for validation
validationDataset <- dowJonesNext[-validationIndex,]

# use the remaining 80% of data to training and testing the models
trainingDataset <- dowJonesNext[validationIndex,]

# LG Logistic Regression
set.seed(7)
fitNext.glm <- train(Label~., data=trainingDataset, method="glm", metric=metric,
                     trControl=control, na.action=na.omit)

# GLMNET Regularized Logistic Regression
set.seed(7)
fitNext.glmnet <- train(Label~., data=trainingDataset, method="glmnet", metric=metric,
                        trControl=control, na.action=na.omit)

# KNN k-Nearest Neighbours
set.seed(7)
fitNext.knn <- train(Label~., data=trainingDataset, method="knn", metric=metric,
                     trControl=control, na.action=na.omit)

# CART Classification and Regression Trees
set.seed(7)
fitNext.cart <- train(Label~., data=trainingDataset, method="rpart", metric=metric,
                      trControl=control, na.action=na.omit)

# NB Naive Bayes
set.seed(7)
fitNext.nb <- train(Label~., data=trainingDataset, method="nb", metric=metric,
                    trControl=control, na.action=na.omit)

# SVM Support Vector Machines with Radial Basis Functions
set.seed(7)
fitNext.svm <- train(Label~., data=trainingDataset, method="svmRadial", metric=metric,
                     trControl=control, na.action=na.omit)

# Bagged CART
set.seed(7)
fitNext.treebag <- train(Label~., data=trainingDataset, method="treebag", metric=metric,
                         trControl=control, na.action=na.omit)
```

```
# Random Forest
set.seed(7)
fitNext.rf <- train(Label~., data=trainingDataset, method="rf", metric=metric,
                    trControl=control, na.action=na.omit)

# Stochastic Gradient Boosting
set.seed(7)
fitNext.gbm <- train(Label~., data=trainingDataset, method="gbm", metric=metric,
                     trControl=control, verbose=FALSE, na.action=na.omit)

# C5.0
set.seed(7)
fitNext.c50 <- train(Label~., data=trainingDataset, method="C5.0", metric=metric,
                     trControl=control, na.action=na.omit)
```

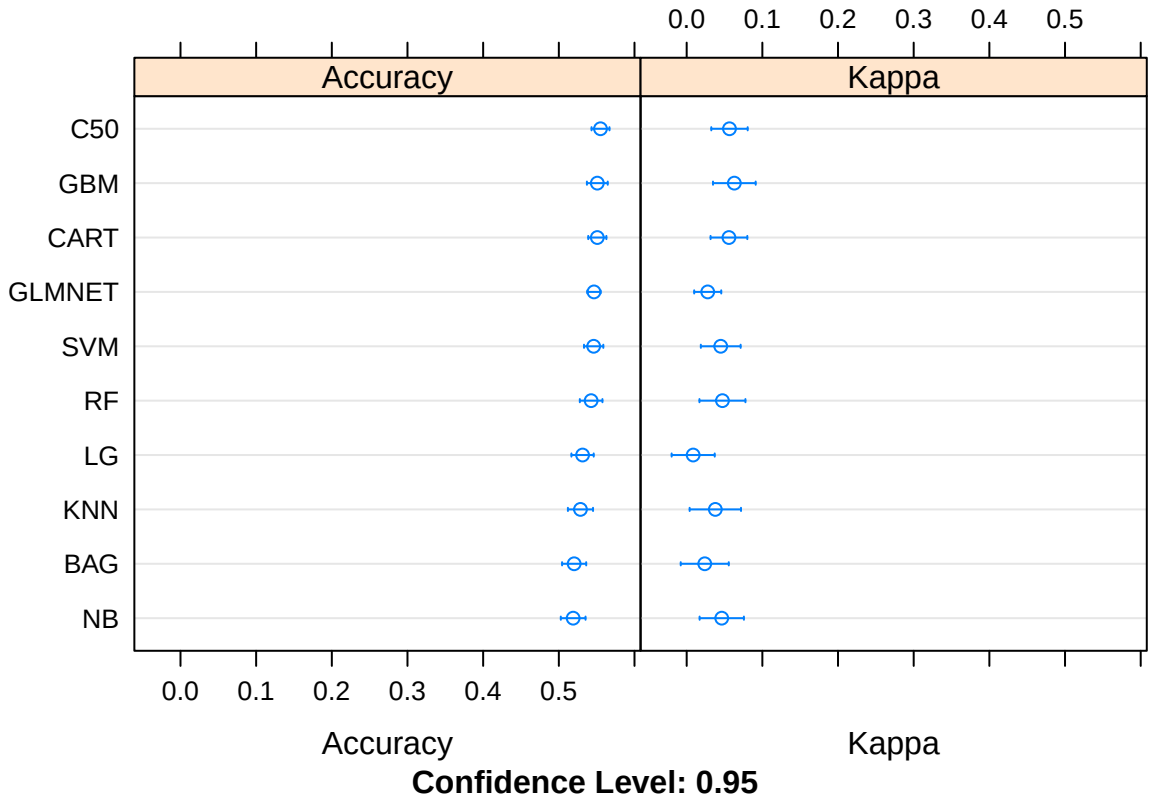
Let us have a peek at the results of the training.

```
resultsNext <- resamples(list(LG=fitNext.glm, GLMNET=fitNext.glmnet, KNN=fitNext.knn,
                              CART=fitNext.cart, NB=fitNext.nb, SVM=fitNext.svm,
                              BAG=fitNext.treebag, RF=fitNext.rf, GBM=fitNext.gbm,
                              C50=fitNext.c50))
summary(resultsNext)
```

```
Call:
summary.resamples(object = resultsNext)

Models: LG, GLMNET, KNN, CART, NB, SVM, BAG, RF, GBM, C50
Number of resamples: 30

Accuracy
         Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
LG     0.3962  0.5133 0.5316 0.5314  0.5535 0.5912    0
GLMNET 0.5063  0.5321 0.5472 0.5465  0.5660 0.5912    0
KNN    0.4430  0.5008 0.5220 0.5286  0.5645 0.6289    0
CART   0.4969  0.5252 0.5503 0.5509  0.5723 0.6289    0
NB     0.4151  0.4937 0.5268 0.5190  0.5512 0.5975    0
SVM    0.4654  0.5346 0.5423 0.5460  0.5645 0.6164    0
BAG    0.4403  0.4969 0.5094 0.5202  0.5535 0.6289    0
RF     0.4654  0.5190 0.5377 0.5427  0.5708 0.6203    0
GBM    0.4780  0.5251 0.5472 0.5509  0.5723 0.6226    0
C50    0.4717  0.5324 0.5503 0.5551  0.5827 0.6038    0

Kappa
           Min.   1st Qu.   Median     Mean 3rd Qu.   Max. NA's
LG     -0.24400 -0.028900 0.002898 0.008568 0.04974 0.1329    0
GLMNET -0.05316 -0.002774 0.028920 0.027790 0.06836 0.1214    0
KNN    -0.13040 -0.026950 0.028540 0.037840 0.10760 0.2427    0
CART   -0.05986  0.004771 0.053960 0.055850 0.10270 0.2197    0
NB     -0.12300 -0.003947 0.058110 0.046350 0.09400 0.1801    0
SVM    -0.11930  0.018720 0.039460 0.044990 0.08246 0.1950    0
BAG    -0.14720 -0.026610 0.006131 0.023870 0.08814 0.2430    0
RF     -0.11450 -0.012610 0.031390 0.047240 0.10970 0.2020    0
GBM    -0.08359  0.004979 0.052940 0.062960 0.11370 0.2091    0
```

```
C50     -0.09547  0.008512 0.061020 0.056540 0.11100 0.1577     0
```

```
dotplot(resultsNext)
```



The best accuracy is obtained by C50 = 0.5551, GBM = 0.5509, CART = 0.5509

Let us do predictions with these three classifiers now, and also try SVM and RF.

```
set.seed(7)
predictionsNextC50 <- predict(fitNext.c50, newdata=validationDataset)
confusionMatrix(predictionsNextC50, validationDataset$Label, positive = "1")
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0  39  39
         1 143 176

               Accuracy : 0.5416
                 95% CI : (0.4911, 0.5914)
    No Information Rate : 0.5416
    P-Value [Acc > NIR] : 0.5206

                  Kappa : 0.0344
 Mcnemar's Test P-Value : 2.261e-14

            Sensitivity : 0.8186
```

```
            Specificity : 0.2143
         Pos Pred Value : 0.5517
         Neg Pred Value : 0.5000
             Prevalence : 0.5416
         Detection Rate : 0.4433
   Detection Prevalence : 0.8035
      Balanced Accuracy : 0.5164

       'Positive' Class : 1
```

```
set.seed(7)
predictionsNextGBM <- predict(fitNext.gbm, newdata=validationDataset)
confusionMatrix(predictionsNextGBM, validationDataset$Label, positive = "1")
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0  46  59
         1 136 156

               Accuracy : 0.5088
                 95% CI : (0.4585, 0.559)
    No Information Rate : 0.5416
    P-Value [Acc > NIR] : 0.9129

                  Kappa : -0.0224
 Mcnemar's Test P-Value : 5.255e-08

            Sensitivity : 0.7256
            Specificity : 0.2527
         Pos Pred Value : 0.5342
         Neg Pred Value : 0.4381
             Prevalence : 0.5416
         Detection Rate : 0.3929
   Detection Prevalence : 0.7355
      Balanced Accuracy : 0.4892

       'Positive' Class : 1
```

```
set.seed(7)
predictionsNextCART <- predict(fitNext.cart, newdata=validationDataset)
confusionMatrix(predictionsNextCART, validationDataset$Label, positive = "1")
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0  52  63
         1 130 152

               Accuracy : 0.5139
                 95% CI : (0.4635, 0.564)
```

```
    No Information Rate : 0.5416
    P-Value [Acc > NIR] : 0.8766

                  Kappa : -0.0075
 Mcnemar's Test P-Value : 2.026e-06

            Sensitivity : 0.7070
            Specificity : 0.2857
         Pos Pred Value : 0.5390
         Neg Pred Value : 0.4522
             Prevalence : 0.5416
         Detection Rate : 0.3829
   Detection Prevalence : 0.7103
      Balanced Accuracy : 0.4963

       'Positive' Class : 1
```

```r
set.seed(7)
predictionsNextSVM <- predict(fitNext.svm, newdata=validationDataset)
confusionMatrix(predictionsNextSVM, validationDataset$Label, positive = "1")
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0  45  52
         1 137 163

               Accuracy : 0.5239
                 95% CI : (0.4735, 0.574)
    No Information Rate : 0.5416
    P-Value [Acc > NIR] : 0.7752

                  Kappa : 0.0056
 Mcnemar's Test P-Value : 9.957e-10

            Sensitivity : 0.7581
            Specificity : 0.2473
         Pos Pred Value : 0.5433
         Neg Pred Value : 0.4639
             Prevalence : 0.5416
         Detection Rate : 0.4106
   Detection Prevalence : 0.7557
      Balanced Accuracy : 0.5027

       'Positive' Class : 1
```

```r
set.seed(7)
predictionsNextRF <- predict(fitNext.rf, newdata=validationDataset)
confusionMatrix(predictionsNextRF, validationDataset$Label, positive = "1")
```

```
Confusion Matrix and Statistics
```

```
          Reference
Prediction   0   1
         0  43  69
         1 139 146

               Accuracy : 0.4761
                 95% CI : (0.426, 0.5265)
    No Information Rate : 0.5416
    P-Value [Acc > NIR] : 0.9961

                  Kappa : -0.0872
 Mcnemar's Test P-Value : 1.716e-06

            Sensitivity : 0.6791
            Specificity : 0.2363
         Pos Pred Value : 0.5123
         Neg Pred Value : 0.3839
             Prevalence : 0.5416
         Detection Rate : 0.3678
   Detection Prevalence : 0.7179
      Balanced Accuracy : 0.4577

       'Positive' Class : 1
```

C50 produces the best prediction accuracy rate of 0.5416

Let us now introduce a feed-forward neural network and evaluate the model. After tuning the model with rang, size and decay parameters, the following model was chosen.

```
set.seed(7)
fitNext.ann = nnet(Label~., data=trainingDataset,rang = 2, size=70, maxit=50000,
                   decay=0.00001, na.action=na.omit)

predictionsNextAnn <- predict(fitNext.ann, newdata=validationDataset, type="class")
predictionsNextAnn <- as.factor(predictionsNextAnn)
confusionMatrix(predictionsNextAnn, validationDs$Label, positive = "1")
```

Confusion Matrix and Statistics

```
      Reference
```

Prediction 0 1 0 93 98 1 89 117

```
           Accuracy : 0.529
             95% CI : (0.4785, 0.579)
No Information Rate : 0.5416
P-Value [Acc > NIR] : 0.7105

              Kappa : 0.055
```

Mcnemar's Test P-Value : 0.5585

```
        Sensitivity : 0.5442
        Specificity : 0.5110
     Pos Pred Value : 0.5680
     Neg Pred Value : 0.4869
         Prevalence : 0.5416
```

```
     Detection Rate : 0.2947
```

Detection Prevalence : 0.5189

Any simmering hope of achieving a slightly better result with the use of a neural network with a large number of neurons in the hidden layer quickly disappears.

Our neural network only produces accuracy of 0.529, which falls in line with the results we have obtained so far.

And now, the last throw of the dice. Let us try to center and scale the dataset, and then throw it into yet another neural network R package called 'neuralnet'. This package likes the data to be in the range 0-1. As far as the number of neurons is concerned, it should be between the input layer size and the output layer size, usually 2/3 of the input size [9]. Variuos combinations of the size of the hidden layers were attempted anyway, and the code below represents one of them.

```
preprocessParams <- preProcess(dowJonesNext[,1:13], method=c("range"))
transformed <- predict(preprocessParams, dowJonesNext[,1:13])
summary(transformed)

library(neuralnet)
transformedNN <- dummyVars(" ~ .", data = transformed)
testFrame <- data.frame(predict(transformedNN, newdata = transformed))

features <- as.formula("Label.1~NegHigh+NegMed+NegLow+PosLow+PosMed+PosHigh+
                        DayMinus1Label.0+DayMinus1Label.1+DayMinus1CORatio+
                        DayMinus2Label.0+DayMinus2Label.1+DayMinus2CORatio+
                        DayMinus3Label.0+DayMinus3Label.1+DayMinus3CORatio")

nn <- neuralnet(features,data=testFrame,hidden=c(6,6),linear.output=T)
```

Neural network did not converge, therefore no weights were created rendering prediction impossible. This outcome is not entirely unpredicted, as so far we have only managed to extract about 54% accuracy rate from the various models being explored.

The more detailed analysis and conclusion will follow up after we complete the stream classifier.

Build Stream Classifier

Now that we have the classical supervised classifier, we shall build the stream classifier utilising the RMOA library in R.

Legt us first initialise options for OCBoost ensemble boosting algorithm which is one of the built-in algorithms in RMOA package.

```
ctrl <- MOAoptions(model = "OCBoost", randomSeed = 123456789, ensembleSize = 25, smoothingParameter = 0
```

Now we shall iterate over the whole stream. This model is being designed to predict the closing state of the stock market on the given day. That means that the requirement is that every chunk will have to do the model training that consists of the model trained on the previous days' data plus the current date.

Once trained, it will have to do the prediction for the current date, and so on.

The following is the possible limitation of the model. It is becoming obvious that after a number of days the model might become too big and the computational strains on the system resources will make it necessary to reset the model and train it with the next chunk.

Then follow it up with the iterations as presented here. Given the small amount of data in our data set, we shall not include this requirement at this point in time.

We will record predictions and plot after the stream is finished.

Now we need to define the variables to control the iterations. A little experiment will be conducted with the size of the data chunks for each repetition. We shall train 3 models with different chunk sizes and compare them.

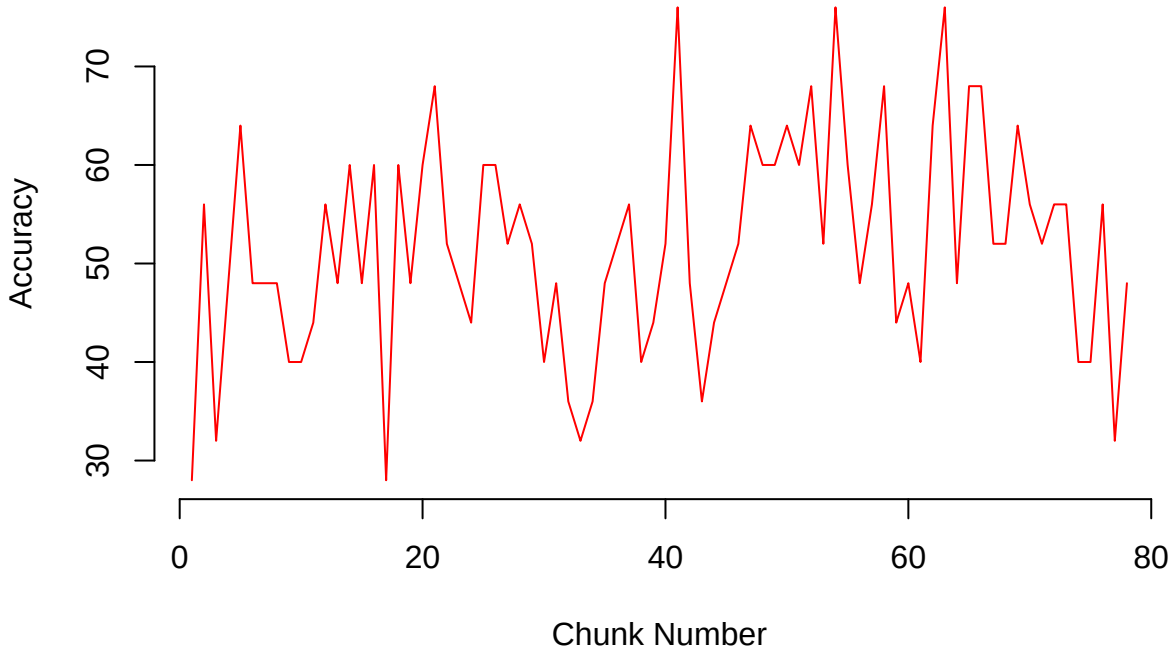
```
chunks <- c(25,50,100)
for (c in 1:3)
{
  # Create an empty OCBoost model
  dowModel <- OCBoost(control = ctrl)

  # Create a stream to data frame
  dowStream <- datastream_dataframe(data=as.data.table(dowJonesNext))

  chunk <- chunks[c]
  iterations <- (nrow(dowStream$data)/chunk)-1
  iterations <- floor(iterations)

  # This is the first sample of 100 records taken from the data stream
  dowSample <- dowStream$get_points(dowStream, n = chunk, outofpoints = c("stop", "warn", "ignore"))

  # Now that we have our first 100 records ready in the stream, let us train the model on these first 1
  dowClassifier <- trainMOA(model = dowModel,
                            formula = Label~NegHigh+NegMed+NegLow+PosLow+PosMed+PosHigh+
                              DayMinus1Label+DayMinus1CORatio+
                              DayMinus2Label+DayMinus2CORatio+
                              DayMinus3Label+DayMinus3CORatio,
                            data = datastream_dataframe(dowSample))

  accuracies <- c()
  for (i in 1:iterations)
  {
    # Next sample
    dowSample <- dowStream$get_points(dfStream, n = chunk, outofpoints = c("stop", "warn", "ignore"))
```

```
    # Update the trained model with the new chunks
    dowClassifier <- trainMOA(model = dowClassifier$model,
                              formula = Label~NegHigh+NegMed+NegLow+PosLow+PosMed+PosHigh+
                                DayMinus1Label+DayMinus1CORatio+
                                DayMinus2Label+DayMinus2CORatio+
                                DayMinus3Label+DayMinus3CORatio,
                              data = datastream_dataframe(dowSample),
                              reset = FALSE,trace = FALSE)

    dowPredictions <- predict(dowClassifier, dowSample)

    # calculate accuracy
    accuracies[i] <- sum(dowPredictions==dowSample$Label)/nrow(dowSample)*100

    #cat("chunk: ",i,"\n")
  }

  # Save the accuracy into the list
  if(c==1)
  {
    accuraciesList <- list(accuracies)
  }
  else
  {
    accuraciesList <- c(accuraciesList, list(accuracies))
  }

  # Reset the stream for the next chunk size
  dowStream$reset()
}
```

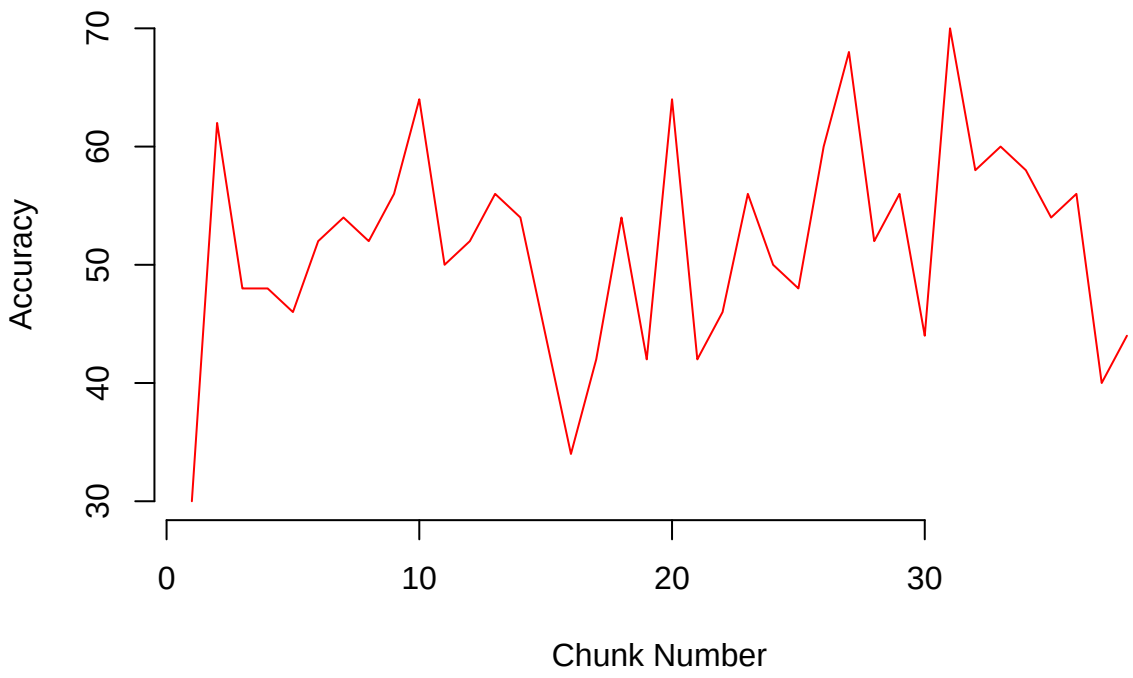
Now is the time to check the accuracies, derive the mean and visualise the accuracies.

```
for (c in 1:3)
{
  print(chunks[c])
  accuracyVector <- accuraciesList[[c]]
  print(summary(accuracyVector))
  plot(accuracyVector,type='l',col='red', xlab="Chunk Number",ylab="Accuracy",frame=FALSE)
}
```

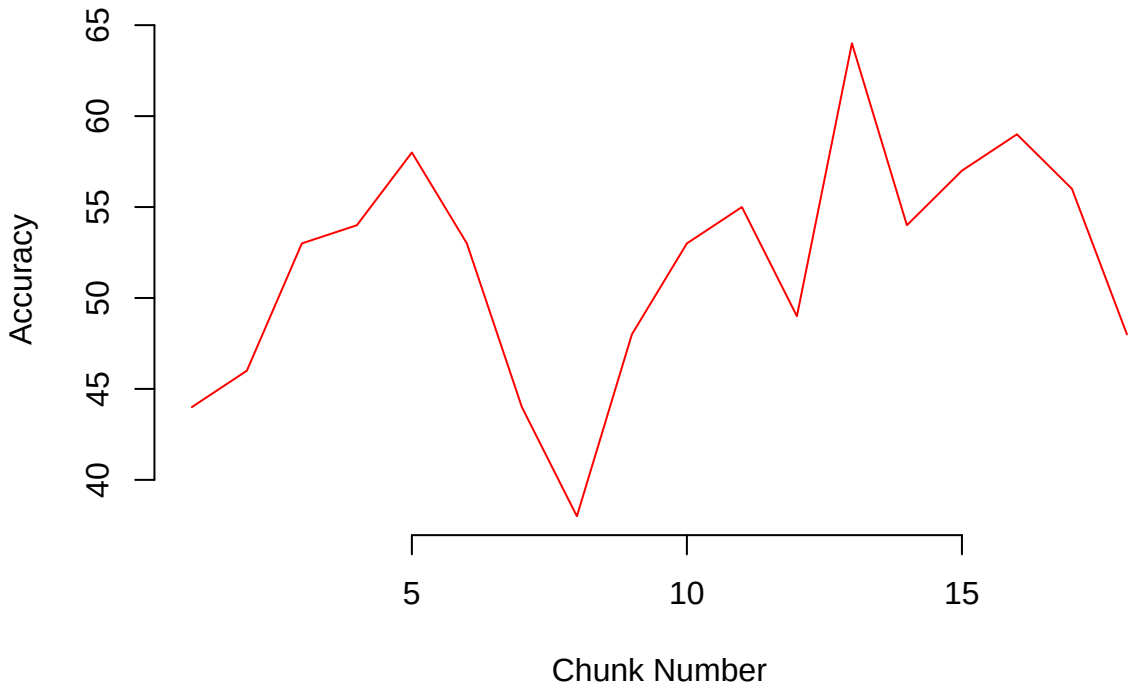
```
[1] 25
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  28.00   45.00   52.00   51.79   60.00   76.00
```




```
[1] 50
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  30.00   46.00   52.00   51.74   56.00   70.00
```




```
[1] 100
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  38.00   48.00   53.00   51.83   55.75   64.00
```



Conclusion

We can see that the mean of 3 models is just about the same. It is also just over 50%, and that was expected, given the previous classical classification models. But what we can see here is that the best short bursts of predictions come from the model with the smallest chunk size of 25. This does not mean that we have the winning model, not at all. But it might be suggesting the direction to take. The downside is that it also has the smallest prediction rate of 28%.

In practice, to be able to build a model with a higher accuracy rate, we would require much better domain knowledge. That is in particular important as it would give us the knowledge of which features are important in stock market flactuations. But that is only the beginning. The second step would be the actual data science and machine learning,that would involve testing of the kind described in this document, plus the possibility of involving deep learning with the large neural networks.

As this example has illustrated, the data set with better features would yield better data streaming classifications results. In reality, the limitation of the model becoming too big would require the resetting of it with the latest data and starting over.

This experiment with the news data was never going to produce fantastic results so that it can be applied in practice, but it has demonstrated the possible first steps. Changing the data structures from the horizontal news features into aggregated news features was an attempt to see if that would bring improvments.

The news sentiment analysis could have been replaced by n-grams text mining as an alternative, but I suspect the results would be around 50% or just slightly above. The main reason for not being able to get more than that is just the nature of the news data. It is unpredictable and therefore very difficult to fit the model around it.

Text Classification

This is the third part of the coursework that focuses on the text analysis and classification. Let us first load the data from the spreadsheet containing the labelled leave and remain twitter data.

```
df <- read.csv("leaveRemainTweets_CW.csv")

# There is no need to have the user.id column as it serves no purpose
df$user.id = NULL
```

Show data frame dimensions (rows and columns) and the names of the columns.

```
nrow(df)
```

```
[1] 2283
```

```
ncol(df)
```

```
[1] 2
```

```
names(df)
```

```
[1] "text"  "label"
```

Divide the data frame into 2 frames, one where the label is set to Leave, and another for Remain.

```
dfLeave <- subset(df, label=="Leave", select=text:label)
dfRemain <-subset(df, label=="Remain", select=text:label)
```

Create constants to be used throughout the script. The change to the value assigned to the constant will propagate to wherever it is being used, so only one change is required, rather than multiple chnages.

```
cnstLr <- "Leave and remain"
cnstLea <- "Leave"
cnstRem <- "Remain"
```

A number of custom functions are written to demonstrate code re-usability. These functions will be called multiple times with different parameters and for different purposes.

This is a custom function to generate a text corpus from a text vector.

```
buildTextCorpus <- function(textAsVector)
{
  # Build the text corpus
  textCorpus <- Corpus(VectorSource(textAsVector))

  # Please note the use of lazy initialisation where possible. It is a computing standard practice
  # of accessing resources only when needed. It may contribute towards the increase of the overall perf

  # Turn the characters into lower case characters
  textCorpus <- tm_map(textCorpus, content_transformer(tolower))

  # Remove punctuation
  textCorpus <- tm_map(textCorpus, removePunctuation,lazy = TRUE)

  # Remove numbers
  textCorpus <- tm_map(textCorpus, removeNumbers,lazy = TRUE)

  # Remove URLs
  removeURL <- function(x)
```

```
  {
    gsub("http[[:alnum:]]*", "", x)
  }

  textCorpus <- tm_map(textCorpus, content_transformer(removeURL))

  # Add extra stop words: 'rt' and 'brexit'
  stopWords <- c(stopwords("english"), "rt", "brexit")

  # Remove stopwords from corpus
  textCorpus <- tm_map(textCorpus, removeWords, stopWords)

  # Remove white spaces
  textCorpus <- tm_map(textCorpus, stripWhitespace)

  # Remove leading and trailing white spaces
  trimLeadingAndTrailing <- function (x)
  {
    gsub("^\\s+|\\s+$", "", x)
  }

  textCorpus <- tm_map(textCorpus, content_transformer(trimLeadingAndTrailing))

  # Stem words
  textCorpus <- tm_map(textCorpus, stemDocument)

  # Remove non UTF8 characters
  removeNonUtf8 <- function (x)
  {
    gsub("[^\x20-\x7E]", "", x)
  }

  textCorpus <- tm_map(textCorpus, content_transformer(removeNonUtf8))

  # Return the clean text corpus to the function caller
  return(textCorpus)
}
```

This function creates a data frame with terms and frequency from the term document matrix with DESC order.

```
getWordFrequencyFromTdm <- function(tdm)
{
  tFreq <- rowSums(as.matrix(tdm))
  tFreq <- subset(tFreq, tFreq >= 15)

  # Store results in a data.frame
  dfFreqTerms <- data.frame(Term = names(tFreq), freq = tFreq)

  # Sort the data frame by most frequent words
  dfFreqTerms <- dfFreqTerms[order(-dfFreqTerms$freq),]

  return(dfFreqTerms)
}
```

This function converts data frame of terms and frequency into a table for displaying purpose. The table will have a client-driven number of rows. It produces a nicely laid out table with 2 columns. It also takes a list of terms to be excluded from displaying, if necessary.

```
getTableFromDataFrame <- function (dfFreqTerms, rowsCount, excludeTerms = NULL)
{
  tfMatrix <- matrix(, nrow = rowsCount, ncol = 2)
  colnames(tfMatrix) <- c("Term", "Frequency")
  rownames(tfMatrix) <- rep("", nrow(tfMatrix)) # Row names are not required

  for (i in 1:rowsCount)
  {
    if(is.null(excludeTerms))
    {
      tfMatrix[i, 1] <- as.character(dfFreqTerms$Term[i])
      tfMatrix[i, 2] <- as.character(dfFreqTerms$freq[i])
    }
  }

  return(tfMatrix)
}
```

Custom function to display the bar chart of the most frequent words. Client can determine how many most frequent words are to be displayed by passing in a parameter rowsCount.

```
displayPlot <- function(dfFreqTerms, rowsCount, title)
{
  wordsPlot <- ggplot(dfFreqTerms[1:rowsCount,], aes(x = Term, y = freq))
  wordsPlot <- wordsPlot + geom_bar(stat = "identity")
  wordsPlot <- wordsPlot + xlab( paste(title, "Terms", sep=" ")) + ylab("Frequency") + coord_flip()
  wordsPlot
}
```

This custom function generates a word cloud from the given term document matrix.

```
createWordCloud <- function(tdm)
{
  m <- as.matrix(tdm)
  v <- sort(rowSums(m), decreasing = TRUE)
  d <- data.frame(word = names(v), freq = v)
  d$word <- gsub("~", " ", d$word)

  wordcloud(words = d$word, freq = d$freq, min.freq = 10,
            max.words=2000, random.order=FALSE, rot.per=0.2,
            colors=brewer.pal(8, "Dark2"))
}
```

This function uses the kable function from knitr package to create presentable tables. In this case, the table is populated with the terms frequencies.

```
printFrequency <- function(dfTermsFreq, rowsCount, title)
{
  tfMatrix <- getTableFromDataFrame(dfTermsFreq, rowsCount)
  kable(title)
  #print(tfMatrix, row.names = FALSE, quote=FALSE)
  kable(tfMatrix)
}
```

Custom function to generate words associations diagram. Takes in a term-document matrix and a diagram title to be displayed as part of the diagram.

```
visualiseAssociations <- function(tdm, title)
{
  # lowfreq is set to 80 occurences as it offers the best choice of words to provide
  # a meaningful set of associations
  fTerms <- findFreqTerms(tdm, lowfreq = 80)

  # corThreshold has been reduced to 0.075 to offer the diagram
  # with the best insight into word associations
  plot(tdm, term = fTerms, corThreshold = 0.075, weighting = T, main=title)
}
```

Custom function to generates a text cluster dendrogram of the associated words. Takes in a term-document matrix and a diagram title to be displayed as part of the diagram.

```
visualiseTextClusters <- function(tdm, title)
{
  denseTdm <- removeSparseTerms(tdm, sparse = 0.97) # remove sparse terms
  denseTdmMatrix <- as.matrix(denseTdm)

  # compute distance between rows of the matrix
  distMatrix <- dist(scale(denseTdmMatrix))

  # cluster using hclust() function
  fit <- hclust(distMatrix, method = "ward.D")

  # plot the results
  plot(fit)

  # add red rectangles to the plot
  rect.hclust(fit, k = 9)
}
```

This function takes in a reference to a text corpus and a n-number of rows, and then displays the n-number of rows from the given text corpus. It also makes use of knitr package for presentable displays.

```
displayNRowsFromCorpus <-function(n, corpus)
{
  tcMatrix <- matrix(, nrow = n, ncol = 2)
  colnames(tcMatrix) <- c("Row", "Text corpus")
  rownames(tcMatrix) <- rep("", nrow(tcMatrix)) # Row names are not required

  for (i in 1:n)
  {
    tcMatrix[i, 1] <- as.character(i)
    tcMatrix[i, 2] <- as.character(corpus[[i]])
  }

  kable(tcMatrix)
}
```

Now that we have defined the functions for this part of the functionality, let us start with building the text corpus and analysing the content. This will show us the quality of the text data in after processing and clean up actions take place.

I will provide seperate answers for each data frame; Leave, Remain and Leave/Remain mixed together. This is done easily through the use of custom functions.

```
textCorpus <- buildTextCorpus(df$text)
textCorpusLeave <- buildTextCorpus(dfLeave$text)
textCorpusRemain <- buildTextCorpus(dfRemain$text)
```

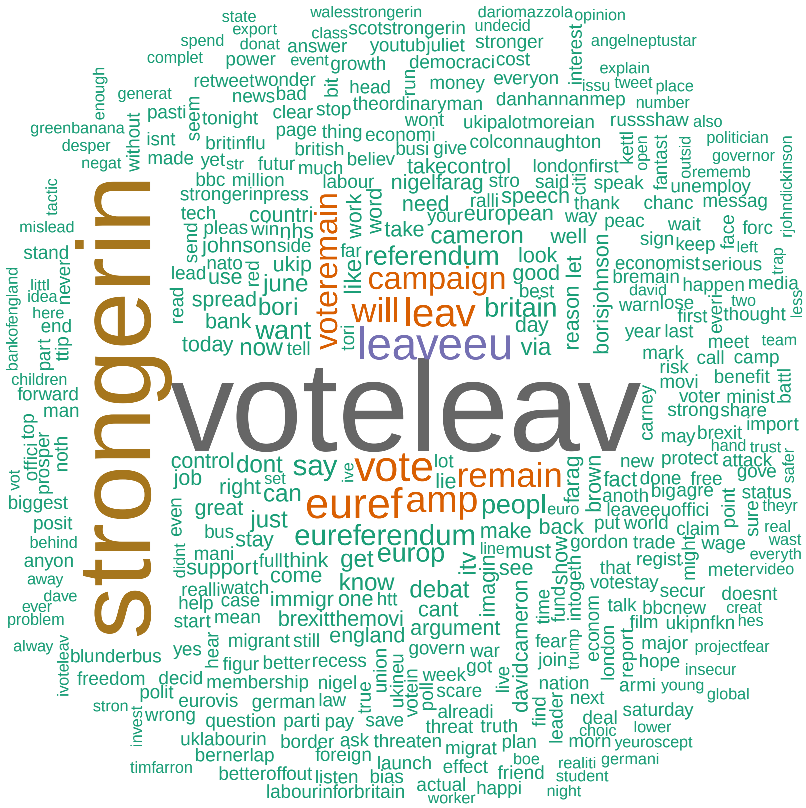
Let us have a look at the first 20 rows from the main text corpus to check out the data.

```
displayNRowsFromCorpus(20, textCorpus)
```

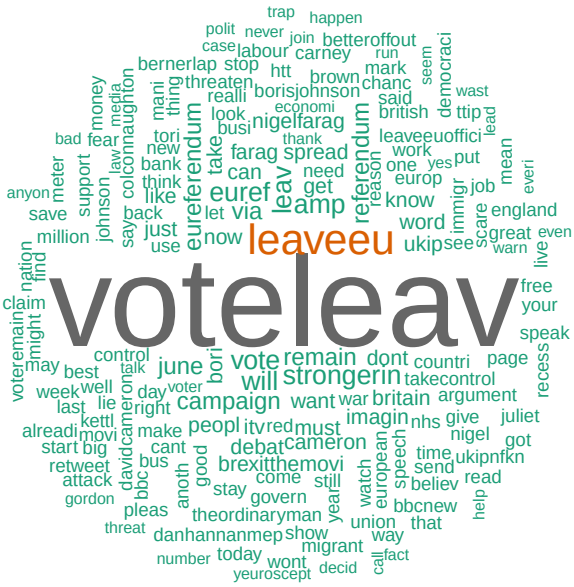
Row	Text corpus
1	kitchi strongerin dont trust unelect beaurocrat children futurebrexit
2	stewartpboro check new voteleav facebook page launch today
3	richarddlewi set referendum strongerin
4	lucill biggest poll yet give point lead borisjohnson voteleav borisjohnson
5	strongerinpress bank england governor mark carney clear leav eu lead mean lower growth plummet pound stronge
6	strongerin book bori johnson say eu help deliv period peac amp prosper strongerin
7	ccblsee leaveeu want uk like albania albania want albanianmfa eualbania almissioneu
8	voteleav retweet five question strongerin courtesi borisjohnson voteleav takecontrol
9	olliewick anyth else countri right decid want voteleav greatbritain
10	nicholastyron yesterday leaveeu post video facebook page might wish sit first
11	lmj londonfirst ralli remain celebr tech london amp europ strongerin
12	hockingsmartin bank warn may increas unemploy much leav uk taxpay billion
13	danielrendal thought sheepl tweet might get retweet voteleav type wonder racist eu meant realis tha
14	tcoflondon argument stay eu weak piss voteleav
15	latticeworkwlth trade anticip market britain vote leav via ft brexitrisk
16	solschleav tale two campaign one gloom pessim passion posit voteleav
17	lordgabalfa spread word voteleav via voteleav
18	bigbigben daili heil unbias eu referendum section strongerin
19	wdjstraw voteleav use word like hyster consid man mark carney know theyr lose argument st
20	andnowimagin imagin efc came said lie havent sack him now imagin voteleav june

Now we shall proceed to build the term document matrix for each of the 3 groups (all data, leave only and remain only). Once we have the term document matrices available, we shall build 3 word clouds.

```
tdm <- TermDocumentMatrix(textCorpus)
tdmLeave <- TermDocumentMatrix(textCorpusLeave)
tdmRemain <- TermDocumentMatrix(textCorpusRemain)

createWordCloud(tdm)
```



```
createWordCloud(tdmLeave)
```




```
createWordCloud(tdmRemain)
```

We are now ready to find the most frequent word in the collection of tweets.

```
dfTermsFreq = getWordFrequencyFromTdm(tdm)
dfTermsFreqLeave = getWordFrequencyFromTdm(tdmLeave)
dfTermsFreqRemain = getWordFrequencyFromTdm(tdmRemain)

printFrequency(dfTermsFreq, 1, cnstLr)
```

Term	Frequency
voteleav	987

```
printFrequency(dfTermsFreqLeave, 1, cnstLea)
```

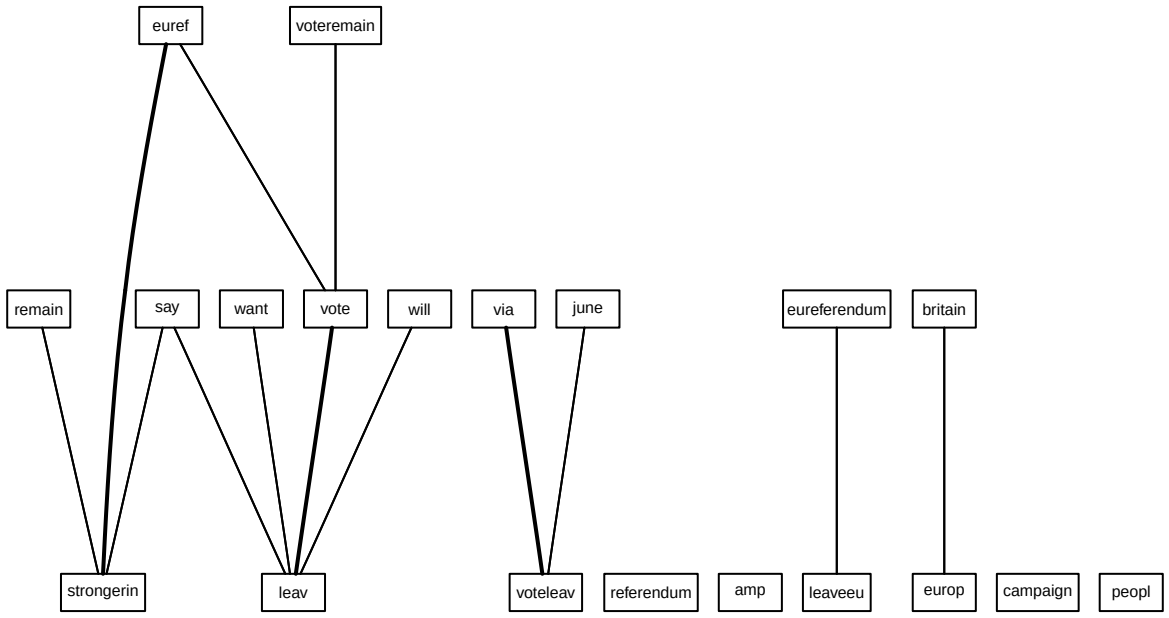
Term	Frequency
voteleav	971

```
printFrequency(dfTermsFreqRemain, 1, cnstRem)
```

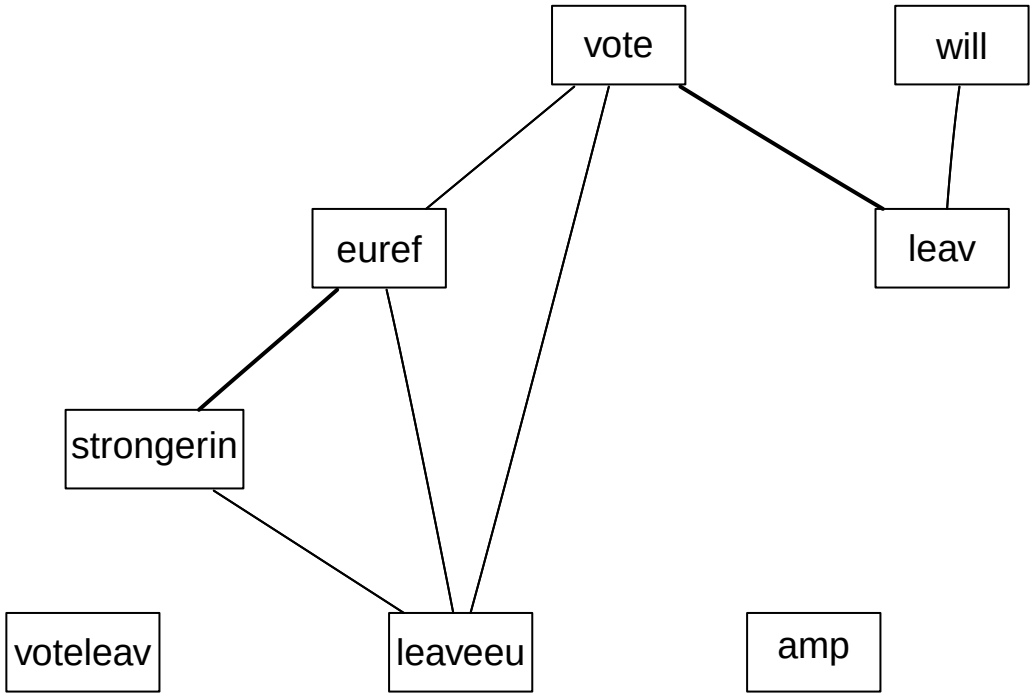
Term	Frequency
strongerin	681

The following code will create words associations visualisations and text clusters.

```
visualiseAssociations(tdm, cnstLr)
```

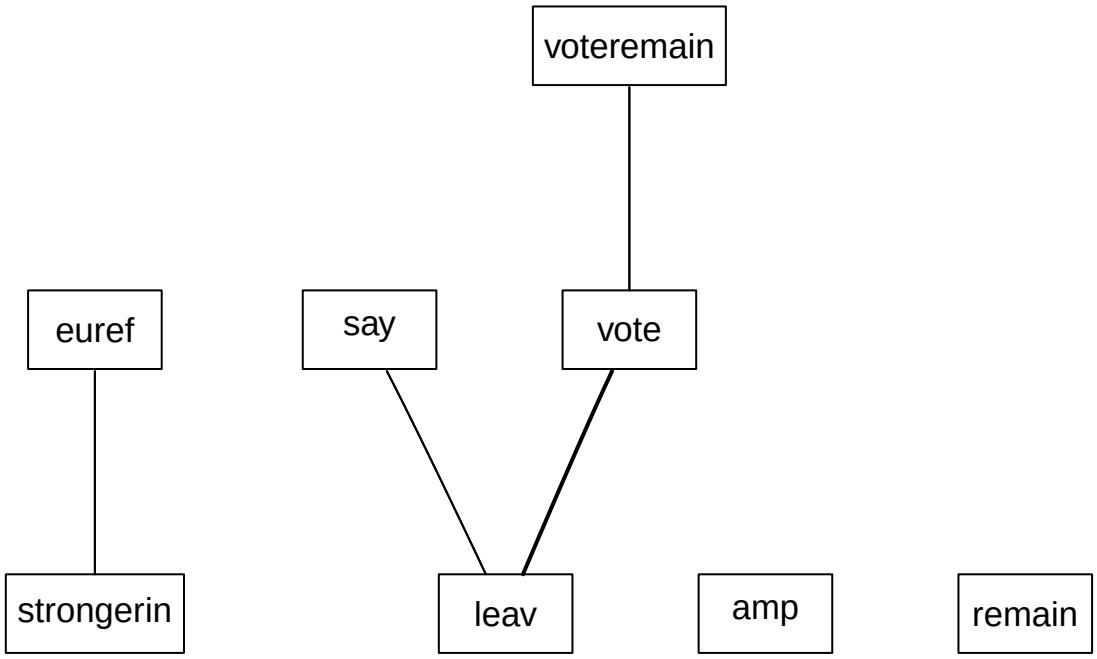



```
visualiseAssociations(tdmLeave, cnstLea)
```

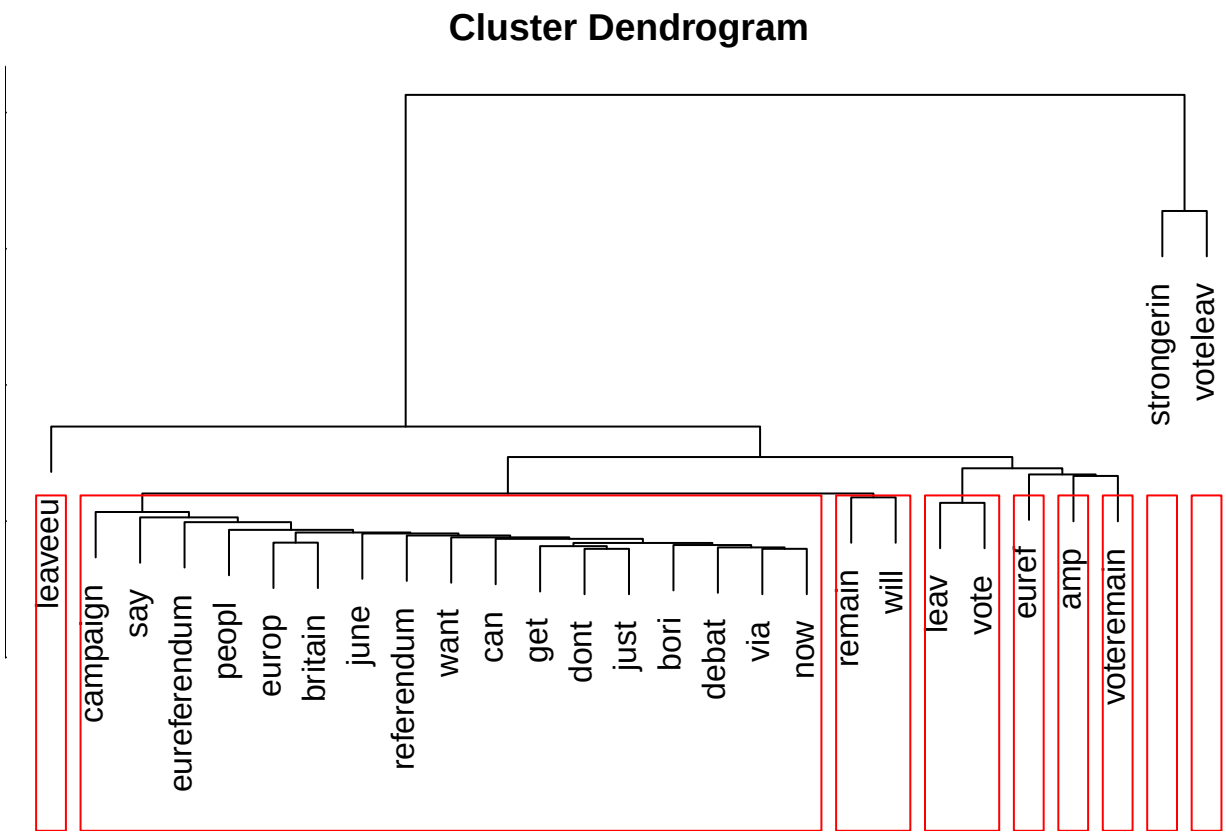



```
visualiseAssociations(tdmRemain, cnstRem)
```

Remain

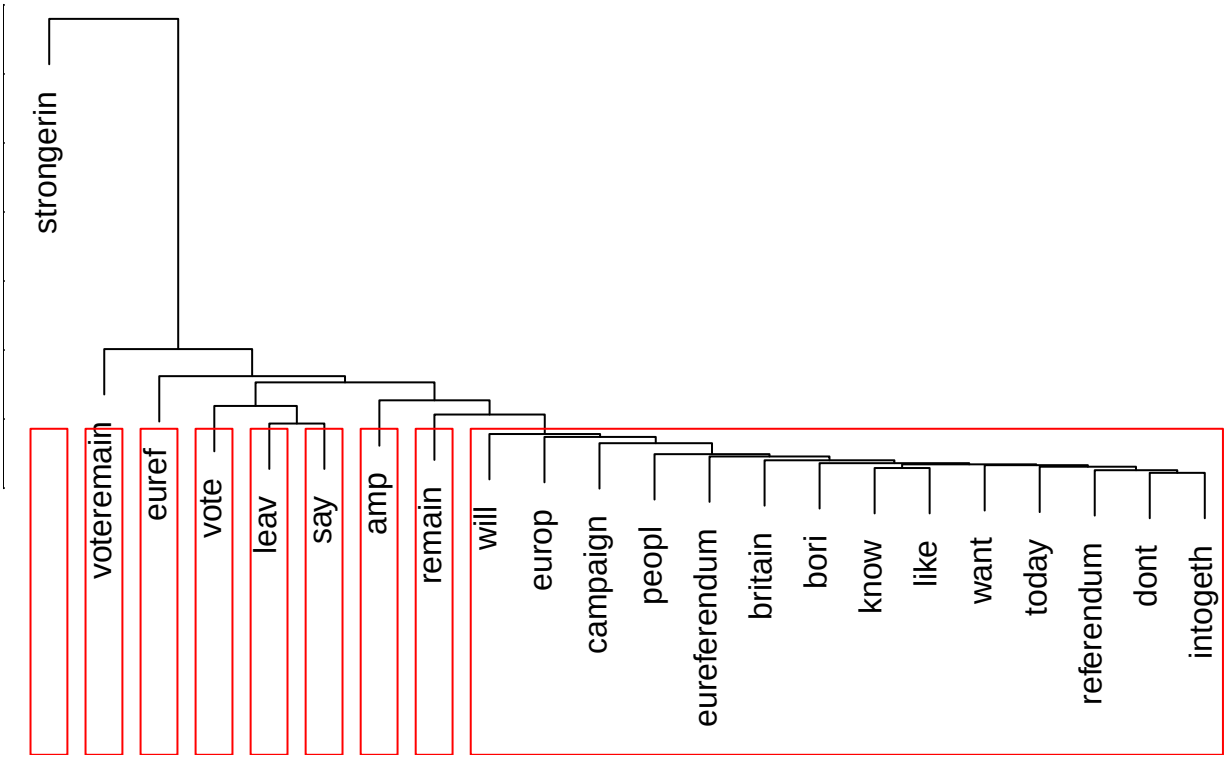



```
visualiseTextClusters(tdm, cnstLr)
```




```
visualiseTextClusters(tdmLeave, cnstLea)
```


```
visualiseTextClusters(tdmRemain, cnstRem)
```



Here we shall display the table with top 20 most frequent words.The number 20 is chosen to provide a wider understanding of the model.The frequent words table will be accompanied with 3 plots, which will be histograms of the most frequent words.

```
printFrequency(dfTermsFreq, 20, cnstLr)
```

Term	Frequency
voteleav	987
strongerin	776
leaveeu	256
vote	239
euref	238
leav	200
amp	186
remain	168
will	164
voteremain	133
campaign	130
eureferendum	116
say	113
want	100
peopl	97
referendum	95
europ	89
britain	83
via	82

Term	Frequency
june	82

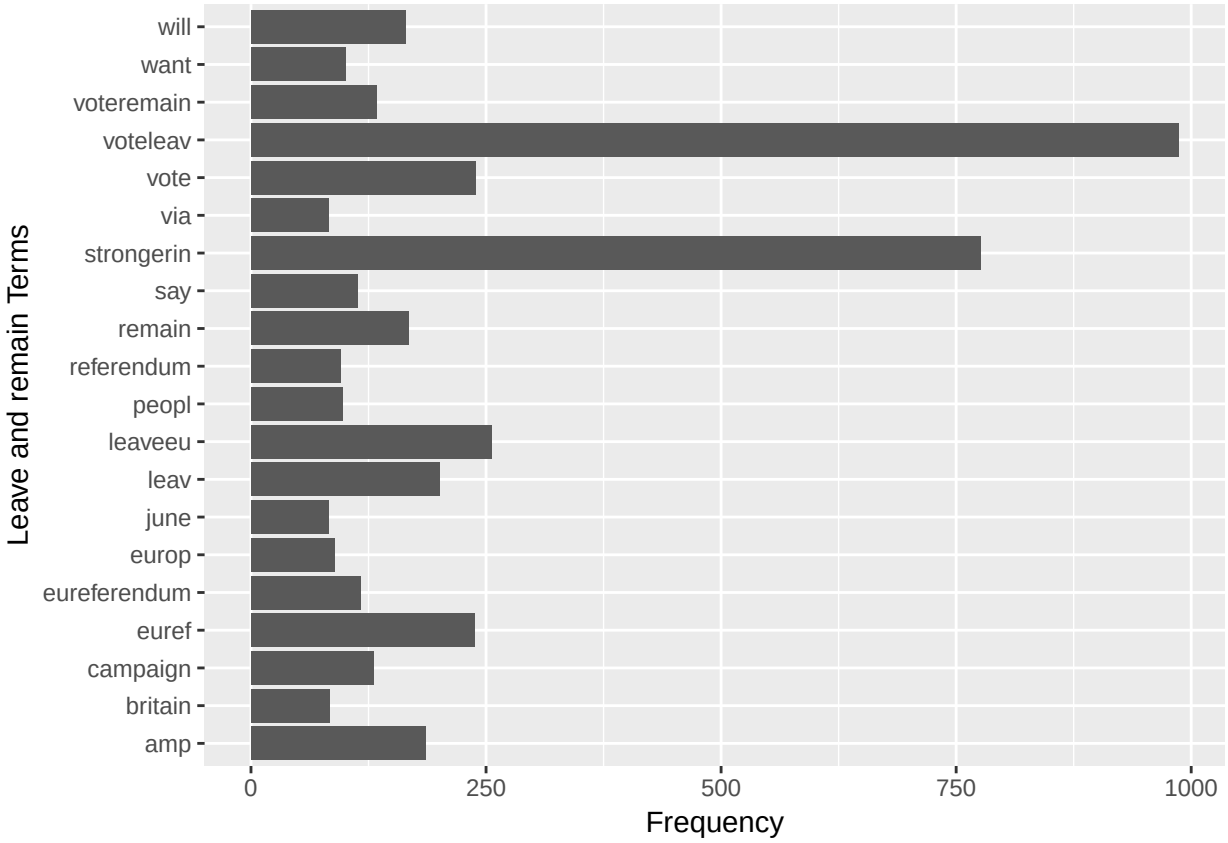
```
printFrequency(dfTermsFreqLeave, 20, cnstLea)
```

Term	Frequency
voteleav	971
leaveeu	234
vote	100
will	99
strongerin	95
amp	89
leav	88
euref	84
campaign	76
remain	72
eureferendum	72
june	67
referendum	60
via	58
want	57
peopl	54
now	52
debat	52
itv	52
get	51

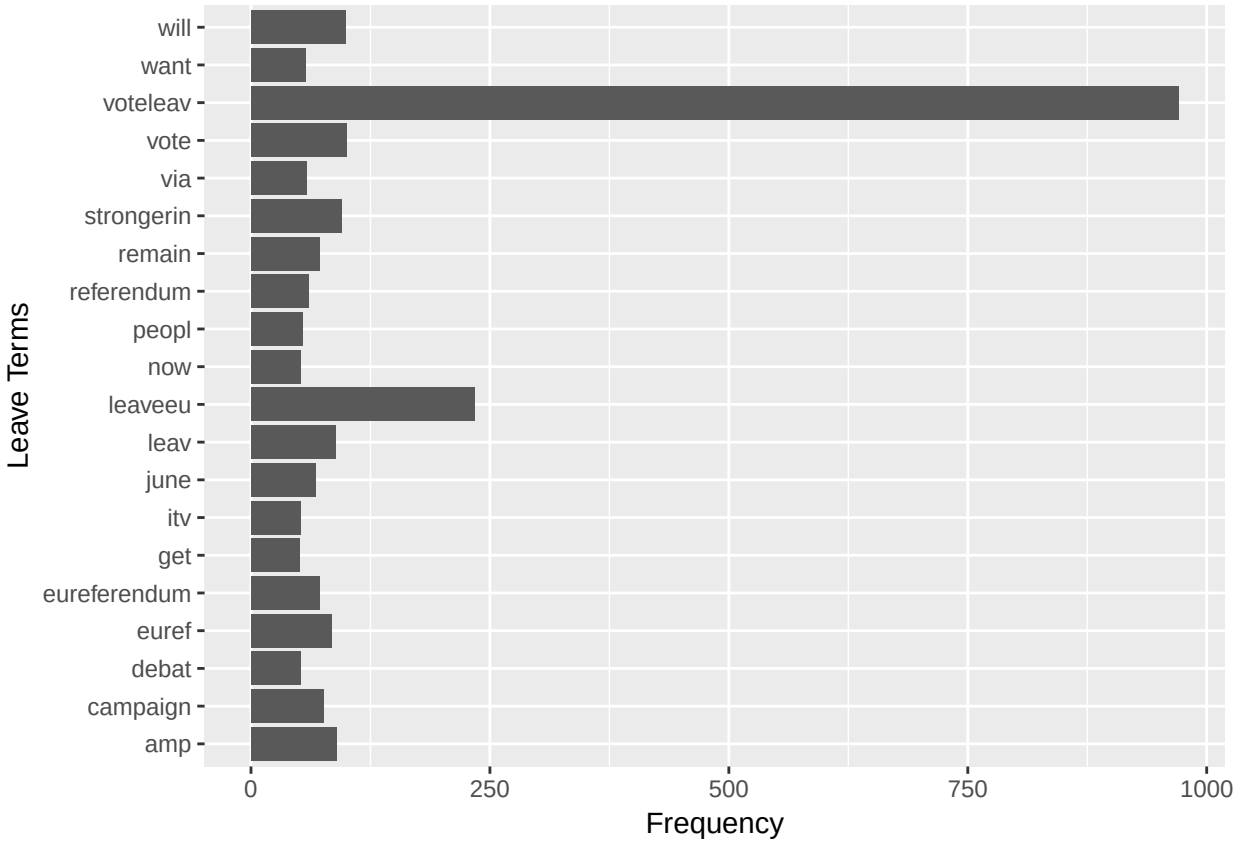
```
printFrequency(dfTermsFreqRemain, 20, cnstRem)
```

Term	Frequency
strongerin	681
euref	154
vote	139
voteremain	120
leav	112
amp	97
remain	96
say	82
will	65
europ	58
campaign	54
eureferendum	44
want	43
peopl	43
britain	41
bori	40
referendum	35
know	34
like	34
intogeth	33

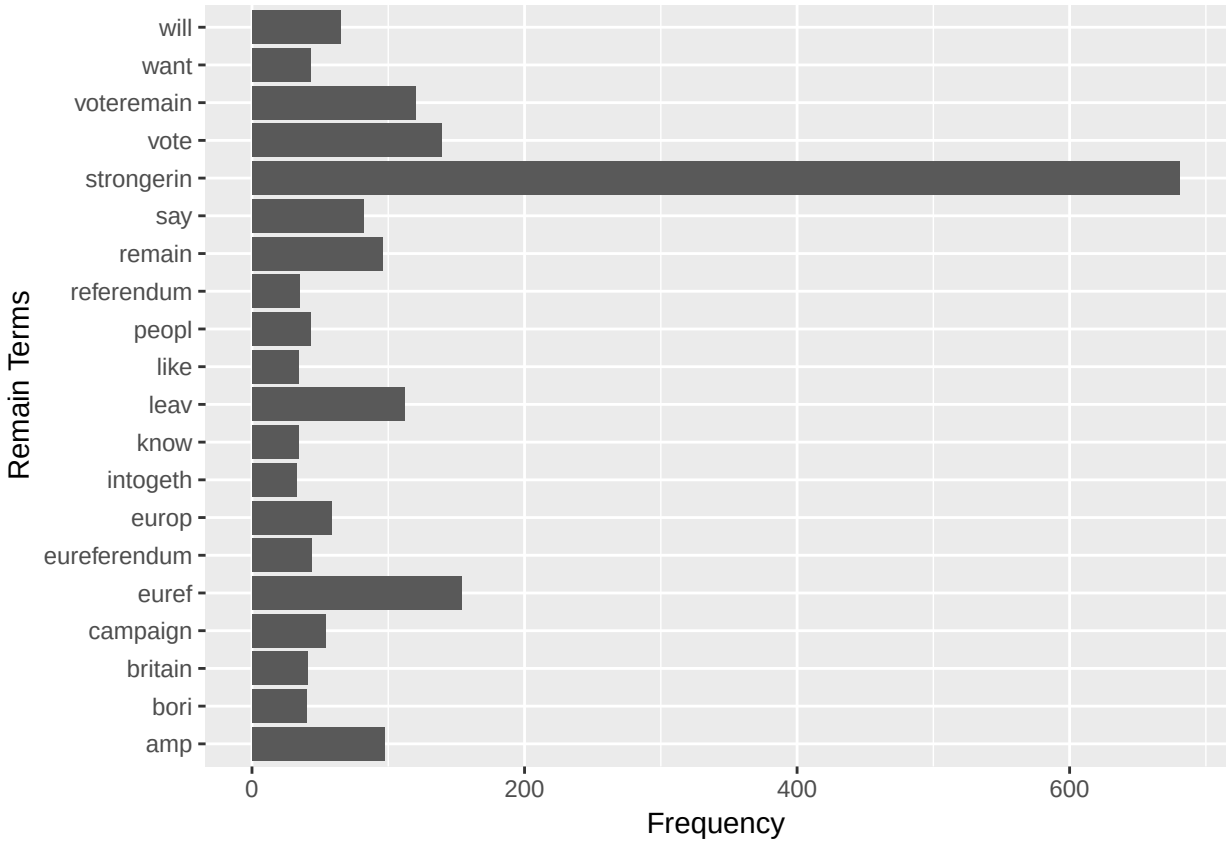
```
displayPlot(dfTermsFreq, 20, cnstLr)
```




```
displayPlot(dfTermsFreqLeave, 20, cnstLea)
```




```
displayPlot(dfTermsFreqRemain, 20, cnstRem)
```


will
want
voteremain
vote
strongerin
say
remain
referendum
peopl
like
leav
know
intogeth
europ
eureferendum
euref
campaign
britain
bori
amp
Remain Terms
0
200
400
600
Frequency

Naive Bayes approach to text classification

Now we shall engage Naive Bayes algorithm to help us with the text classification challenge where we have to build a predictive model based on the leave and remain tweets. As the ongoing trend already suggests, we shall start with building all the funtions that will be utilised to arrive to the final model.

The Bayes classifier only works on categoricial data. This custom function will convert numbers into categorical values.

```
convert_counts <- function(x)
{
  x <- ifelse(x > 0,1,0)
  x <- factor(x,levels = c(0,1),labels = c("No","Yes"))
  return(x)
}
```

This function takes in the terms reduction parameter and runs Naive Bayes classifier. It saves predictions for each lowFreq value passed in to the function and returns them to the caller.

```
runNaiveBayesClassifier <- function (lowFreqVector, laplaceSmooting = 0)
{
  predictions <- vector("list", length(lowFreqVector))

  for(i in 1:length(lowFreqVector))
  {
    # Reduction of the features will be performed by removing terms
    # that appear less than n times across the documents. Let us experiment to find the best n value.
    freqTerms <- findFreqTerms(dtm, lowFreqVector[i])

    # Check the length of the freqTerms/T above. Now we update our training and testing sets
    trainDtmRed <- trainDtm[,freqTerms]
    testDtmRed <- testDtm[,freqTerms]

    # Categorise values for Naive Bayes algorithm
    trainDtmRedCat <- apply(trainDtmRed, MARGIN = 2, FUN=convert_counts)
    testDtmRedCat <- apply(testDtmRed, MARGIN = 2, FUN = convert_counts)

    # Run Naive Bayes classification algorithm
    set.seed(7)
    bayesModel <- naiveBayes(trainDtmRedCat, trainLabels, laplace = laplaceSmooting)

    # Let us evaluate the classifier
    prediction <- predict(bayesModel,testDtmRedCat)
    predictions[[i]] <- prediction

  }

  return (predictions)
}
```

Custom function to create a table from the given dimensions and column names.

```
createTable <- function(nrow, ncol, columnNames)
{
  accMatrix <- matrix(, nrow = nrow, ncol = ncol)
  colnames(accMatrix) <- columnNames
```

```
  rownames(accMatrix) <- rep("", nrow(accMatrix)) # Row names are not required
  return(accMatrix)
}
```

Custom function to display the table indicating the features reduction and the corresponding accuracy.

```
printAccuracyTable <- function(predictions)
{
  accTable = createTable(length(predictions), 2, c("Feature reduction", "Accuracy"))
  for (i in 1:length(predictions))
  {
    prediction <- predictions[[i]]

    # Calculate the accuracy metrics from the confusion matrix
    acc <- sum(diag(as.matrix(table(prediction,testLabels))))/length(prediction)

    accTable[i, 1] <- as.character(lowFreqVector[i])
    accTable[i, 2] <- as.character(acc)
  }

  kable(accTable)

}
```

Custom function to display confusion matrix and CrossTable evaluation metrics.

```
printConfusionMatrix <- function(predictions)
{
  for (i in 1:length(predictions))
  {
    prediction <- predictions[[i]]

    # Confusion matrix
    print(table(prediction,testLabels))

    # Library gmodels provides a function CrossTable for alternative evaluation
    CrossTable(prediction,testLabels,prop.chisq = FALSE, prop.t = FALSE, dnn = c('predicted','actual'))
  }
}
```

Now that we have all the functions defined, let us find out what the class/label distribution is.

```
table(df$label)
```

```
 Leave Remain
  1254   1029
```

The DocumentTermMatrix() method will now be used to basically pivot the TermMatrixDocument already created.

```
dtm <- DocumentTermMatrix(textCorpus)
```

Data will now be split into training and testing data sets with the ratio of 80% v 20%, 80% being training data and 20% validation data. Once we have the data split, we shall show the dimensions of the training and validation data sets.

```
partitionIndex <- round(nrow(dtm) * 0.8)
trainDtm <- dtm[1:partitionIndex,]
trainLabels <- df[1:partitionIndex,]$label

nextPartitionIndex <- partitionIndex + 1
testDtm <- dtm[nextPartitionIndex:nrow(dtm),]
testLabels <- df [nextPartitionIndex:nrow(dtm),]$label

dim(trainDtm)
```

```
[1] 1826 5908
```

```
dim(testDtm)
```

```
[1]  457 5908
```

Now that our data is ready, we can run Naive Bayes classifier with different feature reduction sizes to find the optimum feature set.

```
lowFreqVector = c(5,7,10,15)
predictions <- runNaiveBayesClassifier(lowFreqVector)
```

Let us have a look at the accuracy table and the confusion matrix.

```
printAccuracyTable(predictions)
```

Feature reduction	Accuracy
5	0.892778993435449
7	0.897155361050328
10	0.905908096280088
15	0.899343544857768

We can see that the reduction of the words that occur less than 10 times provides the best accuracy rate of 0.905908096280088. The following set of 4 confusion matrices for each reduction shows us how well the classifier has performed for each of the labels.

```
printConfusionMatrix(predictions)
```

```
          testLabels
prediction Leave Remain
    Leave    236     24
    Remain    25    172


   Cell Contents
|-------------------------|
|                       N |
|           N / Row Total |
|           N / Col Total |
|-------------------------|


Total Observations in Table:  457


             | actual
```

```
   predicted |     Leave |    Remain | Row Total |
-------------|-----------|-----------|-----------|
       Leave |       236 |        24 |       260 |
             |     0.908 |     0.092 |     0.569 |
             |     0.904 |     0.122 |           |
-------------|-----------|-----------|-----------|
      Remain |        25 |       172 |       197 |
             |     0.127 |     0.873 |     0.431 |
             |     0.096 |     0.878 |           |
-------------|-----------|-----------|-----------|
Column Total |       261 |       196 |       457 |
             |     0.571 |     0.429 |           |
-------------|-----------|-----------|-----------|
```

```
          testLabels
prediction Leave Remain
    Leave    236     22
    Remain    25    174
```

```
   Cell Contents
|-------------------------|
|                       N |
|           N / Row Total |
|           N / Col Total |
|-------------------------|
```

```
Total Observations in Table:  457
```

```
             | actual
   predicted |     Leave |    Remain | Row Total |
-------------|-----------|-----------|-----------|
       Leave |       236 |        22 |       258 |
             |     0.915 |     0.085 |     0.565 |
             |     0.904 |     0.112 |           |
-------------|-----------|-----------|-----------|
      Remain |        25 |       174 |       199 |
             |     0.126 |     0.874 |     0.435 |
             |     0.096 |     0.888 |           |
-------------|-----------|-----------|-----------|
Column Total |       261 |       196 |       457 |
             |     0.571 |     0.429 |           |
-------------|-----------|-----------|-----------|
```

```
          testLabels
prediction Leave Remain
    Leave    239     21
    Remain    22    175
```

```
   Cell Contents
|-------------------------|
|                       N |
|           N / Row Total |
|           N / Col Total |
|-------------------------|


Total Observations in Table:  457


             | actual
   predicted |     Leave |    Remain | Row Total |
-------------|-----------|-----------|-----------|
       Leave |       239 |        21 |       260 |
             |     0.919 |     0.081 |     0.569 |
             |     0.916 |     0.107 |           |
-------------|-----------|-----------|-----------|
      Remain |        22 |       175 |       197 |
             |     0.112 |     0.888 |     0.431 |
             |     0.084 |     0.893 |           |
-------------|-----------|-----------|-----------|
Column Total |       261 |       196 |       457 |
             |     0.571 |     0.429 |           |
-------------|-----------|-----------|-----------|


          testLabels
prediction Leave Remain
    Leave    236     21
    Remain    25    175


   Cell Contents
|-------------------------|
|                       N |
|           N / Row Total |
|           N / Col Total |
|-------------------------|


Total Observations in Table:  457


             | actual
   predicted |     Leave |    Remain | Row Total |
-------------|-----------|-----------|-----------|
       Leave |       236 |        21 |       257 |
             |     0.918 |     0.082 |     0.562 |
             |     0.904 |     0.107 |           |
-------------|-----------|-----------|-----------|
      Remain |        25 |       175 |       200 |
             |     0.125 |     0.875 |     0.438 |
             |     0.096 |     0.893 |           |
```

```
-------------|-----------|-----------|-----------|
Column Total |       261 |       196 |       457 |
             |     0.571 |     0.429 |           |
-------------|-----------|-----------|-----------|
```

Confusion matrix for the feature reduction of 10 illustrates the best accuracy rates for leave and remain.

It is obvious that the classifiers have performed slightly better when predicting the leave tweets. That could be due to the slightly skewed class distribution working in leave favour. But the initial accuracy rate of 0.905 is encouraging and represents a good start.

Let us now introduce a Laplacian smoothing parameter to see the effects on the predictions. We shall use the highest scoring feature reduction size of 10 minimum counts per word across the documents, and a smoothing parameter set to 0.5

```
lowFreqVector = c(10)
predictions <- runNaiveBayesClassifier(lowFreqVector, 0.5)
printAccuracyTable(predictions)
```

Feature reduction	Accuracy
10	0.910284463894967

```
printConfusionMatrix(predictions)
```

```
          testLabels
prediction Leave Remain
    Leave    241     21
    Remain    20    175


   Cell Contents
|-------------------------|
|                       N |
|           N / Row Total |
|           N / Col Total |
|-------------------------|


Total Observations in Table:  457


             | actual
   predicted |     Leave |    Remain | Row Total |
-------------|-----------|-----------|-----------|
       Leave |       241 |        21 |       262 |
             |     0.920 |     0.080 |     0.573 |
             |     0.923 |     0.107 |           |
-------------|-----------|-----------|-----------|
      Remain |        20 |       175 |       195 |
             |     0.103 |     0.897 |     0.427 |
             |     0.077 |     0.893 |           |
-------------|-----------|-----------|-----------|
Column Total |       261 |       196 |       457 |
```

```
             |     0.571 |     0.429 |           |
-------------|-----------|-----------|-----------|
```

Laplacian smoothing parameter [8] set to 0.5 increases accuracy by roughly 1% to 0.910284463894967. It shows a very slight improvement when comparing the confusion matrix to the confusion matrix produced with the reduction factor of 10.

It is interesting to see that the improvement, however small, applies to both labels.

RTextTools approach to text classification

Our aim is to try to increase the accuracy of the Naive Bayes classifer. For that purpose, we shall utilise the RTextTools package in R, that contains implementation of ensemble classification algorithms through the use of svm, slda, boosting, bagging, random forests, glmnet, decision trees, neural networks and maximum entropy.

As usual, we shall create all the functions to be used throughout the process of building the model. Also, using the prefix rtt for each variable will make it stand out from the other models created so far.

This function creates a data frame containing the algorithms and the mean of their balanced F1 score for each label.

```
getAlgorithmsPerformance <- function()
{
  # Create precision recall summary from prediction results
  pr <- create_precisionRecallSummary(rttContainer, rttResults, b_value = 1)

  # Extract the mean of the F1 balanced scoress from the precision recall summary
  scores <- c(0,0,0,0,0,0,0)
  startPos <- 3
  for (i in 1:7)
  {
    scores[startPos / 3] <- (pr[(startPos * 2) - 1] + pr[(startPos * 2)])/2
    startPos <- startPos + 3
  }

  # Create a final data frame
  scoresDf <- data.frame(Algorithm = c("SVM","SLDA","LOGITBOOST","BAGGING","FORESTS","TREE", "MAXENTROP
                         MeanF1Score = scores)

  return(scoresDf)
}
```

This function creates a confusion matrix for the given algorithms.

```
getConfusionMatrix <- function(result)
{
  # Convert the 1 and 2 factors into Leave and Remain factors
  resultFactored = factor(result, labels = c("Leave", "Remain"))
  return (confusionMatrix(resultFactored, testLabels))
}
```

Let us now create the document term matrix and container. The container receives the partition index so that it can work out what data to use for training and validation.

```
rttDtm <- create_matrix(df$text, language="english", removeNumbers=TRUE,
                          removePunctuation=TRUE, stripWhitespace=TRUE, toLower=TRUE,
                          removeStopwords=TRUE, stemWords=TRUE, removeSparseTerms=.998)

# Configure the training and testing data
rttContainer <- create_container(rttDtm, as.numeric(factor(df$label)), trainSize=1:partitionIndex,
                                 testSize=nextPartitionIndex:nrow(df), virgin=FALSE)
```

Now we cant train the model with SVM, BOOSTING, MAXENT, RF, TREE, SLDA and BAGGING algorithms.

```
set.seed(7)
rttModels <- train_models(rttContainer, algorithms=c("SVM", "BOOSTING", "MAXENT",
                                                     "RF", "TREE", "SLDA", "BAGGING"))
rttResults <- classify_models(rttContainer, rttModels)
rttAnalytics <- create_analytics(rttContainer, rttResults, b=1)
```

Get the algorithms' performance balanced F score and put them into a data frame. Then sort the data frame in descending order to see the best performing algorithms occupying the top positions.

```
rttFScores <- getAlgorithmsPerformance()
rttFScores <- rttFScores[with(rttFScores, order(-MeanF1Score)), ]
kable(rttFScores)
```

	Algorithm	MeanF1Score
5	FORESTS	0.930
1	SVM	0.920
3	LOGITBOOST	0.915
2	SLDA	0.900
4	BAGGING	0.895
7	MAXENTROPY	0.890
6	TREE	0.880

We can see that the two best performing algorithms are Random forests and Support vector machine.

The following code will now display confusion matrix for RF and SVM.

```
for (i in 1:2)
{
  algName <- as.character(rttFScores[i,1])
  parts <-  c('rttResults$', algName, '_LABEL')
  modelLabel <- paste(parts, collapse = '')

  # Use eval function to pass in a dynamic parameter converted from the text value in modelLabel
  #https://www.r-bloggers.com/converting-a-string-to-a-variable-name-on-the-fly-and-vice-versa-in-r/
  cat(algName, "", sep = " ")
  print(getConfusionMatrix(eval(parse(text = modelLabel))))
}
```

```
FORESTS Confusion Matrix and Statistics

          Reference
Prediction Leave Remain
    Leave    240      9
    Remain    21    187

               Accuracy : 0.9344
                 95% CI : (0.9076, 0.9553)
    No Information Rate : 0.5711
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.867
 Mcnemar's Test P-Value : 0.04461

            Sensitivity : 0.9195
```

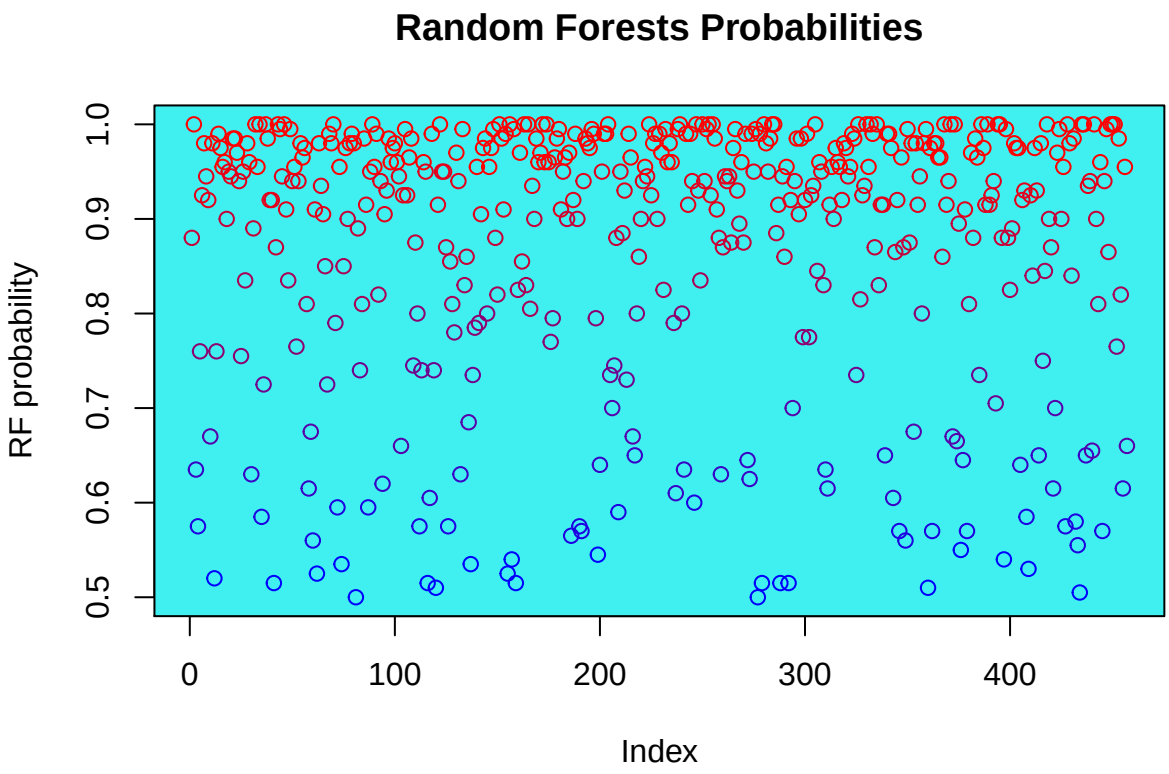
```
            Specificity : 0.9541
         Pos Pred Value : 0.9639
         Neg Pred Value : 0.8990
             Prevalence : 0.5711
         Detection Rate : 0.5252
   Detection Prevalence : 0.5449
      Balanced Accuracy : 0.9368

       'Positive' Class : Leave

SVM Confusion Matrix and Statistics

          Reference
Prediction Leave Remain
    Leave    235     11
    Remain    26    185

               Accuracy : 0.919
                 95% CI : (0.8901, 0.9424)
    No Information Rate : 0.5711
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.8363
 Mcnemar's Test P-Value : 0.02136

            Sensitivity : 0.9004
            Specificity : 0.9439
         Pos Pred Value : 0.9553
         Neg Pred Value : 0.8768
             Prevalence : 0.5711
         Detection Rate : 0.5142
   Detection Prevalence : 0.5383
      Balanced Accuracy : 0.9221

       'Positive' Class : Leave
```

Random forests shows only 30 mis-classified instances out of 457 with the overall accuracy of 0.9344 and Kappa value sitting at 0.867, which represents a very good to excellent ratio for Kappay value. That means that model is a good fit and that accuracy is down to that, rather than the pure random chance.

We can also note that leave prediction rate is very high at 0.9639. This could be due to the slight class imbalance where leave outnumbers remain, but the accuracy rate is very close between the two. Let us show a few digrams before we complete this with the final conclusion.

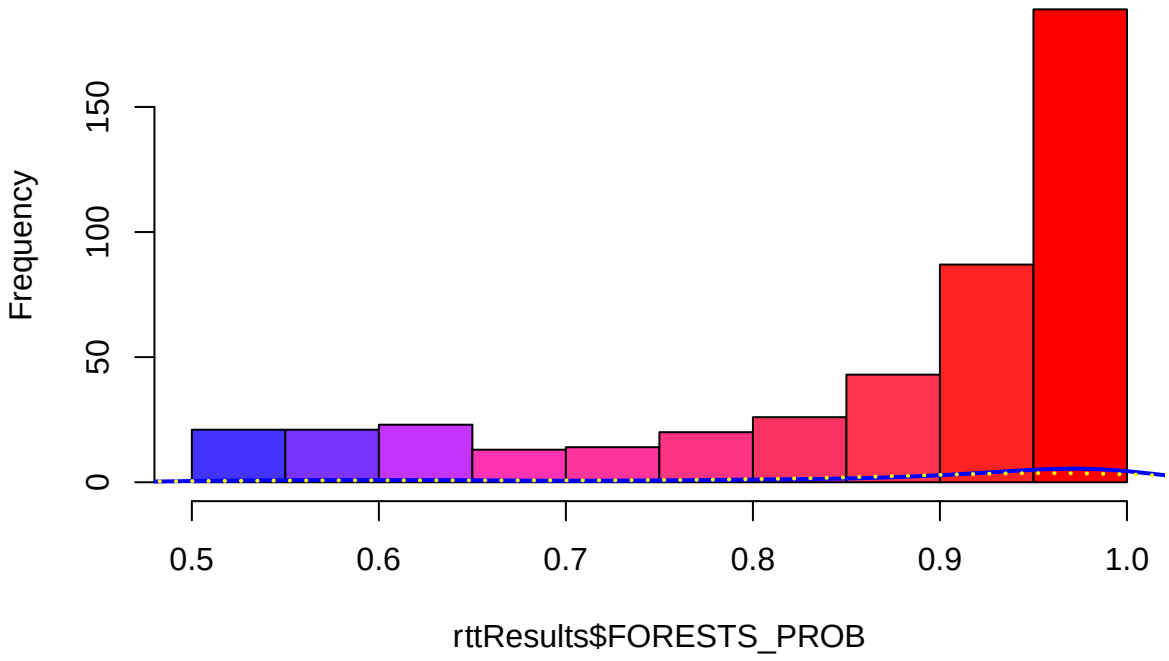
This is a plot of Random Forests probabilities for each classified instance. As the probability value gets closer to 100%, the colour transitions closer to red.

```
rbPal <- colorRampPalette(c('blue','red'))
colours <- rbPal(10)[as.numeric(cut(rttResults$FORESTS_PROB,breaks = 10))]
plot(rttResults$FORESTS_PROB, main="Random Forests Probabilities", ylab="RF probability", col=colours, 
rect(par("usr")[1],par("usr")[3],par("usr")[2],par("usr")[4],col = "#40F0F0")
points(rttResults$FORESTS_PROB, main="Random Forests Probabilities", ylab="RF probability", col=colours
```



Show the histogram of probabilities.

```
hist(rttResults$FORESTS_PROB, main=names(rttResults$FORESTS_PROB),
     col = c("#4333FF", "#7B33FF", "#C533FF", "#FF33B1", "#FF339C", "#FF3383", "#FF3361", "#FF334F","#F
lines(density(rttResults$FORESTS_PROB), col="blue", lwd=2) # Add a density estimate with defaults
lines(density(rttResults$FORESTS_PROB, adjust=2), lty="dotted", col="yellow", lwd=2)
```



Show the number of probabilities where P(prediction)>=0.75 and a percentage of it.

```
highProb <- length(rttResults$FORESTS_PROB[rttResults$FORESTS_PROB >= 0.75])
print(highProb)
```

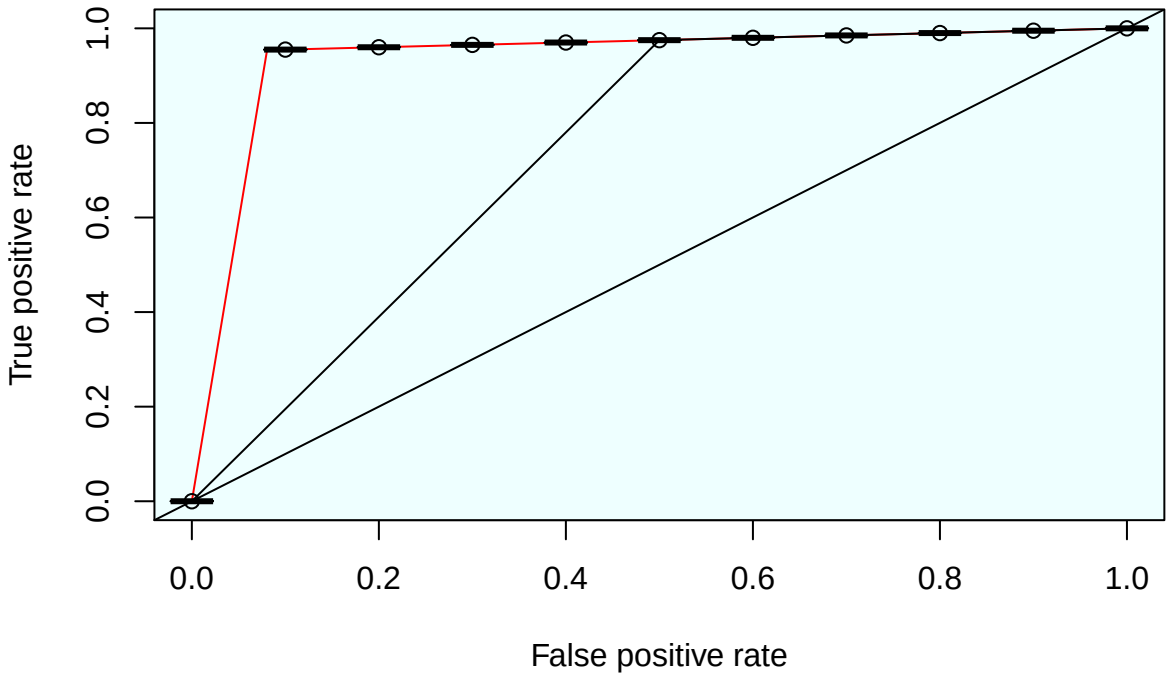
```
[1] 366
```

```
print(highProb / length(rttResults$FORESTS_PROB))
```

```
[1] 0.8008753
```

Now plot a ROC curve for Random forests algorithm.

```
testLabelsFac <- factor(testLabels, labels = c("1", "2"))
testLabelsFac <- as.numeric(levels(testLabelsFac))[testLabelsFac]
rfLabelsFac <-as.numeric(levels(rttResults$FORESTS_LABEL))[rttResults$FORESTS_LABEL]

rfPrediction <- prediction(rfLabelsFac, testLabelsFac)
plot(performance(rfPrediction,"tpr","fpr"))
rect(par("usr")[1],par("usr")[3],par("usr")[2],par("usr")[4],col = "#EEFFFF")
plot(performance(rfPrediction,"tpr","fpr"), col="red",  add=TRUE)
plot(performance(rfPrediction,"tpr","fpr"), avg="vertical", spread.estimate="boxplot", add=TRUE)
abline(a=0, b= 1)
```



Calculate and show the area under curve.

```
rfAuc <- performance(rfPrediction, measure = "auc")
rfAuc <- rfAuc@y.values[[1]]
rfAuc
```

```
[1] 0.9368109
```

The area under the ROC curve is set to 0.9368109 and is close to 1, which may be interpreted as an indication that the algorithm performs very well indeed. The curve going up quickly without shifting to the horizontal right represents a high accuracy of true positives with a few false positives.

Is it possible to further improve the accuracy - well, we can try by experimenting with the ntree parameter in Random forests algorithm. We will try with the following values for ntree: 100, 150, 500 and 1000.

```
ntreeExp = c(100,150,500,1000)
for (i in 1:4)
{
  set.seed(7)
  rttModelsExp <- train_models(rttContainer, algorithms=c("RF"), ntree = ntreeExp[i])
  rttResultsExp <- classify_models(rttContainer, rttModelsExp)
  cat("ntree = ", ntreeExp[i], sep = "")
  cat(" ", "", sep = "")
  print(getConfusionMatrix(rttResultsExp$FORESTS_LABEL))
}
```

Conclusion and analysis

The best result is achieved by ntree=100 but is no improvement to what we already have, so the final accuracy rate of 0.9344 achieved by Random forest algorithm is an improvement to 0.91 achieved by Naive Bayes.

The obtained accuracy rate is good given the fairly small amount of data. It is almost senseless to even discuss whether the model suffers from the potential overfitting, given the small training and testing data. In the real life scenario, we would need a larger text corpus to extract more features into the model, and then test it with a larger test data to put it under a proper test.

Only then we would be in the position to discuss the real results, bias and variance, and if variance is present, then that might indicate a potential overfitting for some test data and the bias/underfitting in other test areas.

This in a way indicates that we might need the ultimate document terms matrix with all words from the specific language in order to get a fully trained model. This sounds computationally expensive and would require feature (terms) reduction at a larger scale to make it doable.

One of possible ways of doing it would be to utilise Deep Learning methods and libraries, such as word embeddings and Python libraries, such as Gensim, Keras. These also work with Google's Word2Vec embedding and Stanford's GloVe embedding. A word embedding is a class of approaches for representing words and documents using a dense vector representation. It is an improvement over more the traditional bag-of-word model encoding schemas where large sparse vectors are used to represent each word. In an embedding, the position of a word within thee vector space is learned from text and is based on the words that surround the word when it is used. [5]

References

[1] C. Dunis, P. Middleton, A. Karathanasopoulos, K. Theofilatos, "Artificial Intelligence in Financial Markets (Cutting-Edge Applications for Risk Management, Portfolio Optimisation and Economics)", pp. 3, pp. 179-186

[2] E. Siegel, Predictive Analytics, pp.148-149.

[3] G. James, D. Witten, T. Hastie, R. Tibshirani "An Introduction to Statistical Learning", pp. 154 - 164.

[4] Andrew NG, "Machine Learning Yearning", pp. 11

[5] J. Brownlee, "Deep Learning for NLP", pp. 151-152

[6] J. Brownlee, "MACHINE LEARNING MASTERY WITH R, Binary Classification Machine Learning Case Study Project", pp. 190-221

[7] J. Brownlee, "MASTER MACHINE LEARNING ALGORITHMS", pp. 77-82

[8] https://www.quora.com/What-is-Laplacian-smoothing-and-why-do-we-need-it-in-a-Naive-Bayes-classifier

[9] https://datascienceplus.com/fitting-neural-network-in-r/

[x] https://journal.r-project.org/archive/2013/RJ-2013-001/RJ-2013-001.pdf

[x] https://stats.stackexchange.com/questions/132777/what-does-auc-stand-for-and-what-is-it

Appendix

The main R script code used in this coursework can be found at https://github.com/amiromicevic/CMM536AdvancedDataScience

Machine Learning Yearning by Andrew NG, the first draft version can be found at https://github.com/amiromicevic/CMM536AdvancedDataScience/blob/master/Ng_MLY02.pdf

Acknowledgements

I would like to express gratitude to Professor Andrew Ng at Stanford University who has provided the first window to machine learning through his Coursera course. That built great foundations before starting my MSc studies here at RGU. The course was a fantastic experience and made me want to learn more.

I would also like to express my gratitude to Professor Eyad Elyan at RGU for providing us students with first-class lectures in the subject of advanced data science, and maintaining our interest in data science. I enjoyed every minute of it.

And last, but not the least, gratitude goes to my wife and young family for being supportive and understanding during the long hours away studying at RGU library.