Coursework

Advanced Data Science (CMM536)

Amir Omicevic, 1615285@rgu.ac.uk

21 April 2018

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.

LOAD AND SET UP DATA FRAMES

```
# 1. LOAD AND SET UP DATA FRAMES

# Load all libraries required to run the script
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)
library(Rgraphviz)

# Set working directory and load the file
setwd('C:\\Users\\Amir\\Documents\\Coursework\\CMM536AdvancedDataScience')
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"
```

```
# 2. CUSTOM FUNCTIONS

# A number of custom functions are written to demonstrate code re-usability

# 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)
}


# 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)
}


# Convert data frame of terms and frequency into a table for displaying purpose.
# The table will have a client-driven number of rows.
# This function produces a nicely laid out table with 2 columns,
# as opposed to three unnecessary columns in the frequency terms data frame.
# It also takes a list of terms to be excluded from displaying
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
}

# 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"))
}

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

# Generates the words associations 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)
}

# Generates a text cluster dendrogram of the associated words
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)
}

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)
}
```

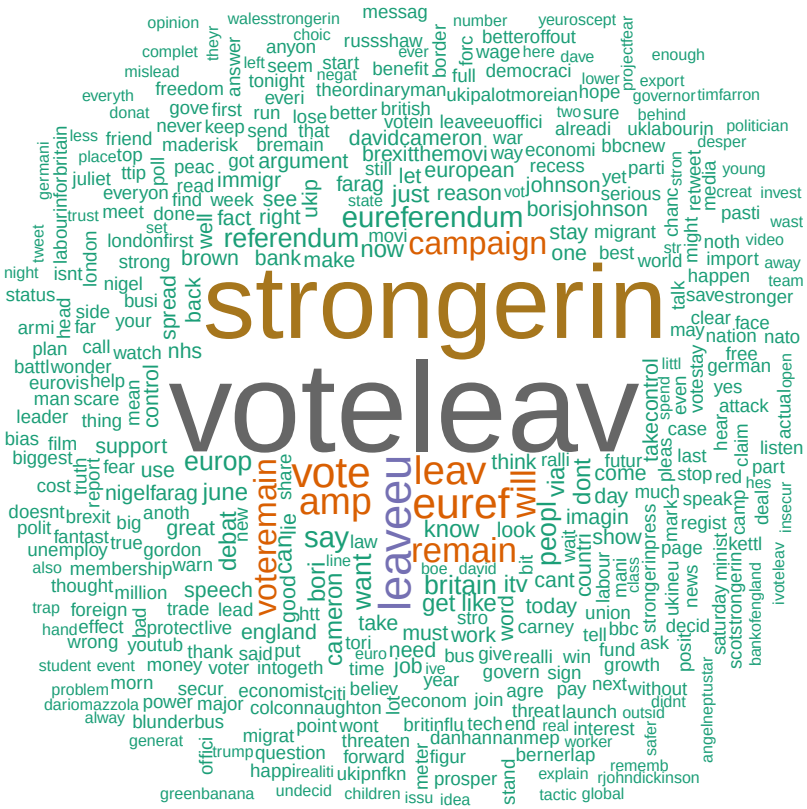
```
# This level of precision might not be needed, but I will provide seperate answers
# for each data frame: Leave, Remain and Leave/Remain mixed together
# This is done easily through the use of custom functions

# 3. TEXT PRE PROCESSING

# Build the text corpus by calling our custom function for each data frame
textCorpus <- buildTextCorpus(df$text)
textCorpusLeave <- buildTextCorpus(dfLeave$text)
textCorpusRemain <- buildTextCorpus(dfRemain$text)
```

```
# Now we shall print out the first 20 rows from the main text corpus to check out
# how well the cleaning process has performed, and if there is anything else
# we might want to do in respect of the text cleaning

displayNRowsFromCorpus(20, textCorpus)
```

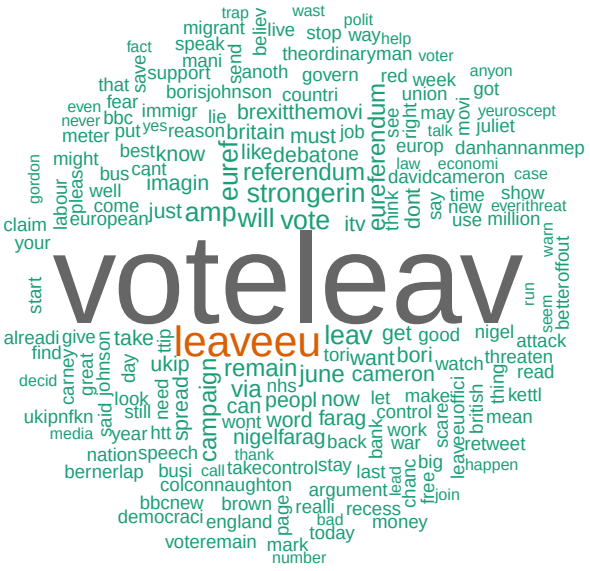
Row	Text corpus
1	kitchi strongerin dont trust unelect beaurocrat children futurebrexit
2	stewartpboro check new voteleav facebook page launch today
3	richarddlewi set referendum strongerin
4	lucill biggest poll yet give point lead borisjohnson voteleav borisjohnson
5	strongerinpress bank england governor 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

Row	Text corpus
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

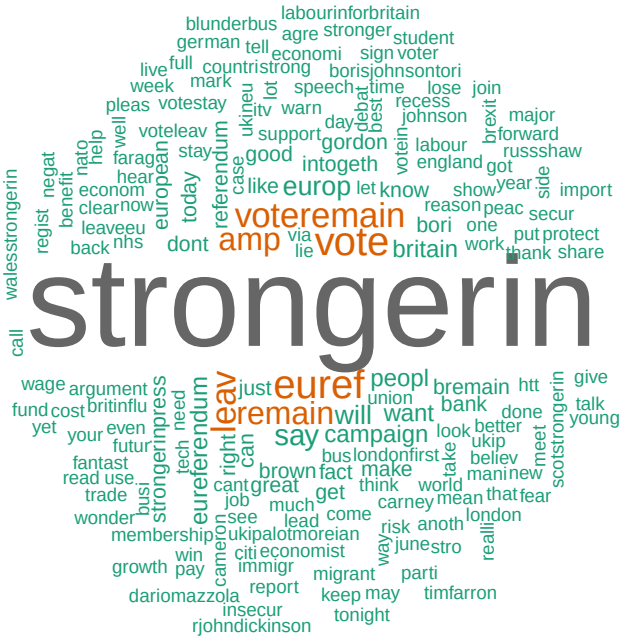
```
# 4. CREATE WORD CLOUDS

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

createWordCloud(tdm)
```




```
createWordCloud(tdmLeave)
```




```
createWordCloud(tdmRemain)
```




```
# 5. GET WORD FREQUENCIES AND ASSOCIATIONS

# 5.1 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

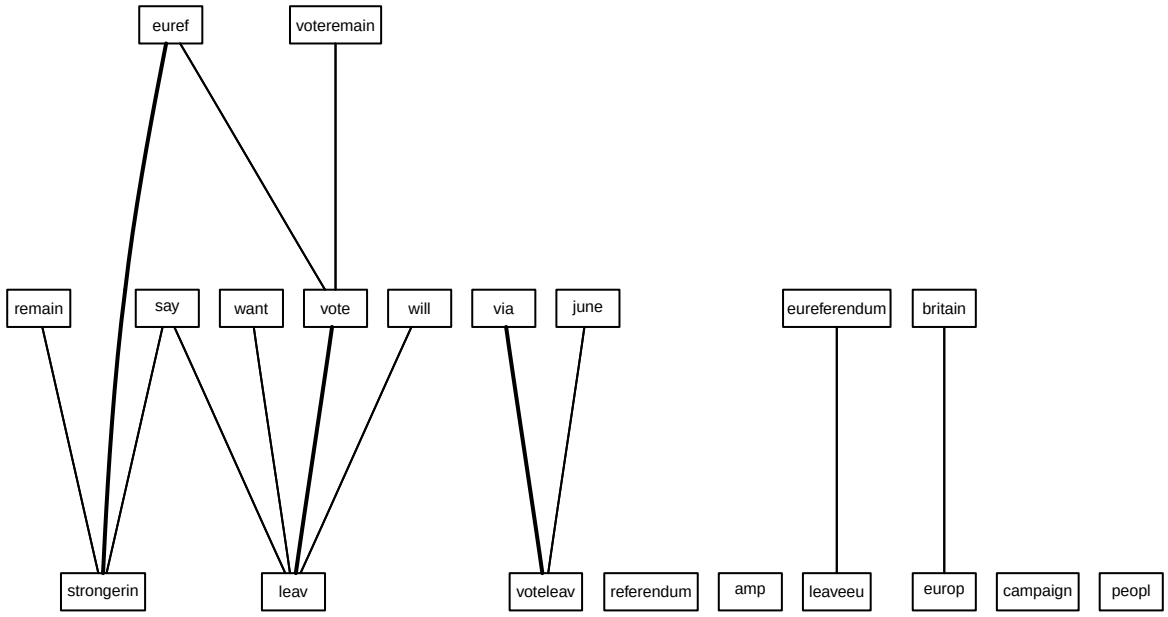
```
printFrequency(dfTermsFreqLeave, 1, cnstLea)
```

Term	Frequency
voteleav	971

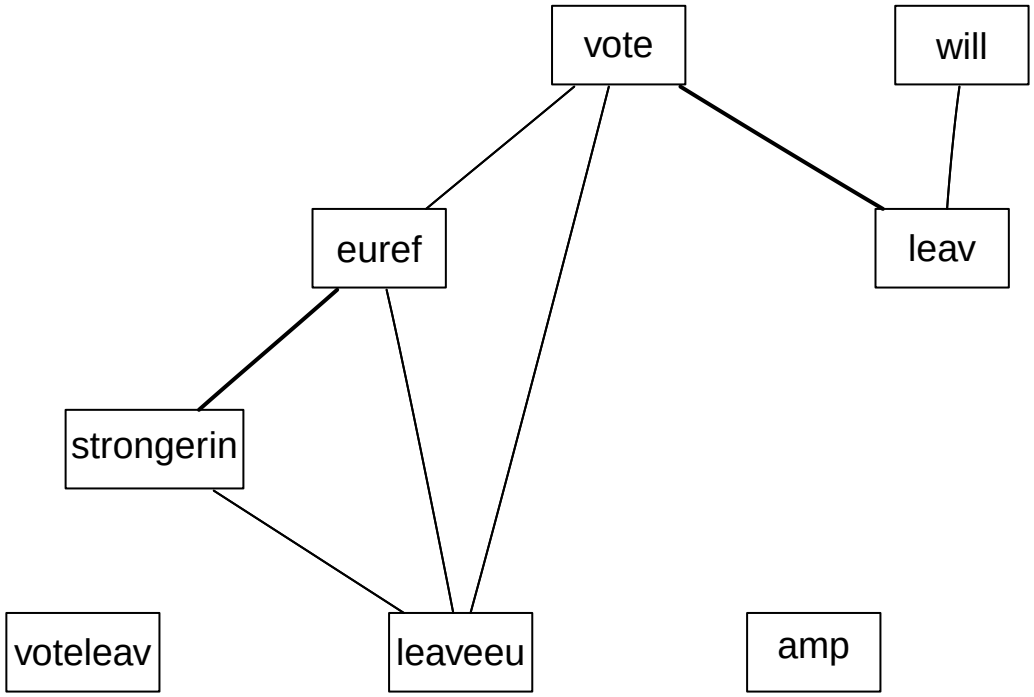
```
printFrequency(dfTermsFreqRemain, 1, cnstRem)
```

Term	Frequency
strongerin	681

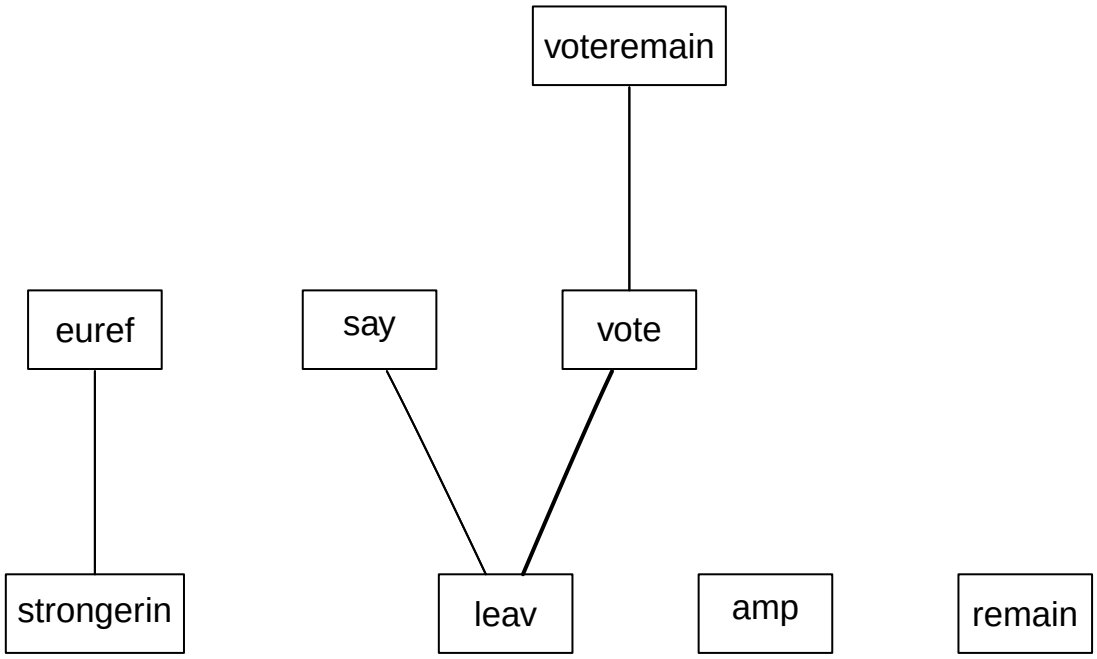
```
# 5.2 Words association, identify the that words appear together often
visualiseAssociations(tdm, cnstLr)
```



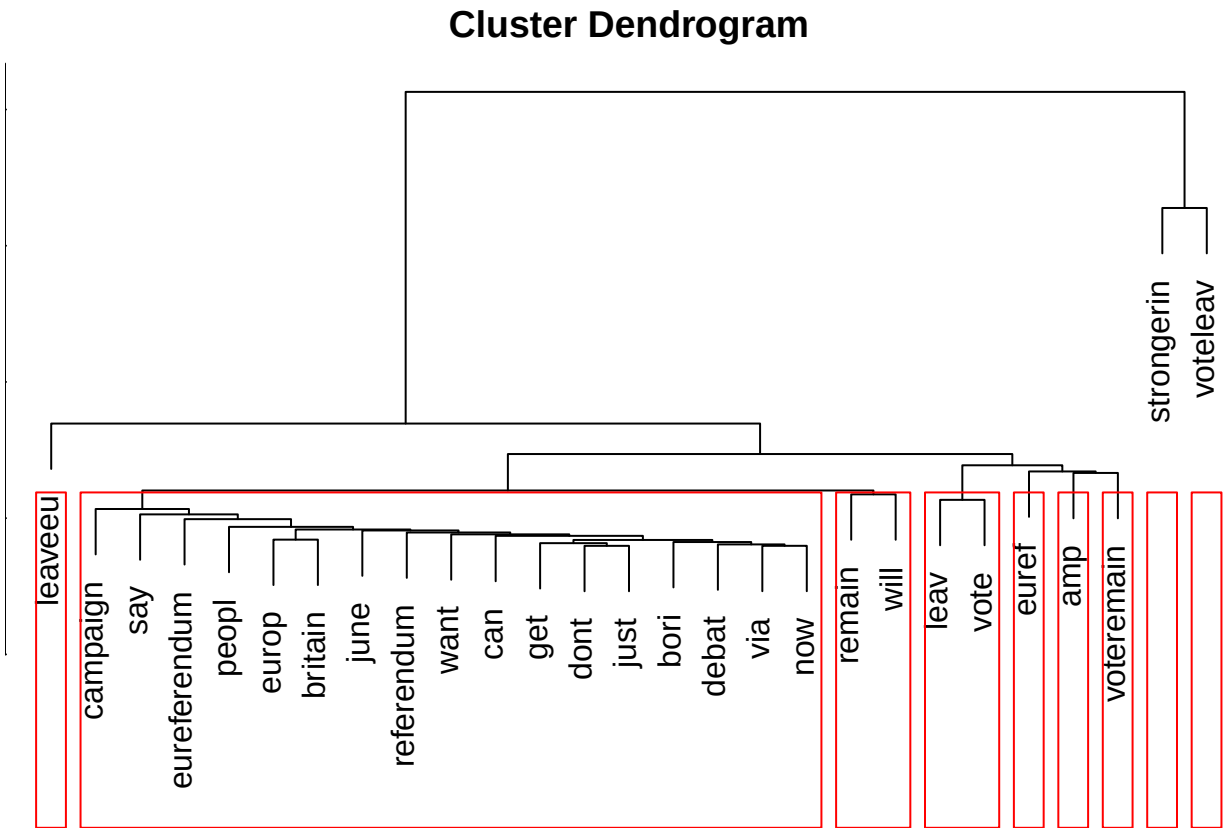

```
visualiseAssociations(tdmLeave, cnstLea)
```




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

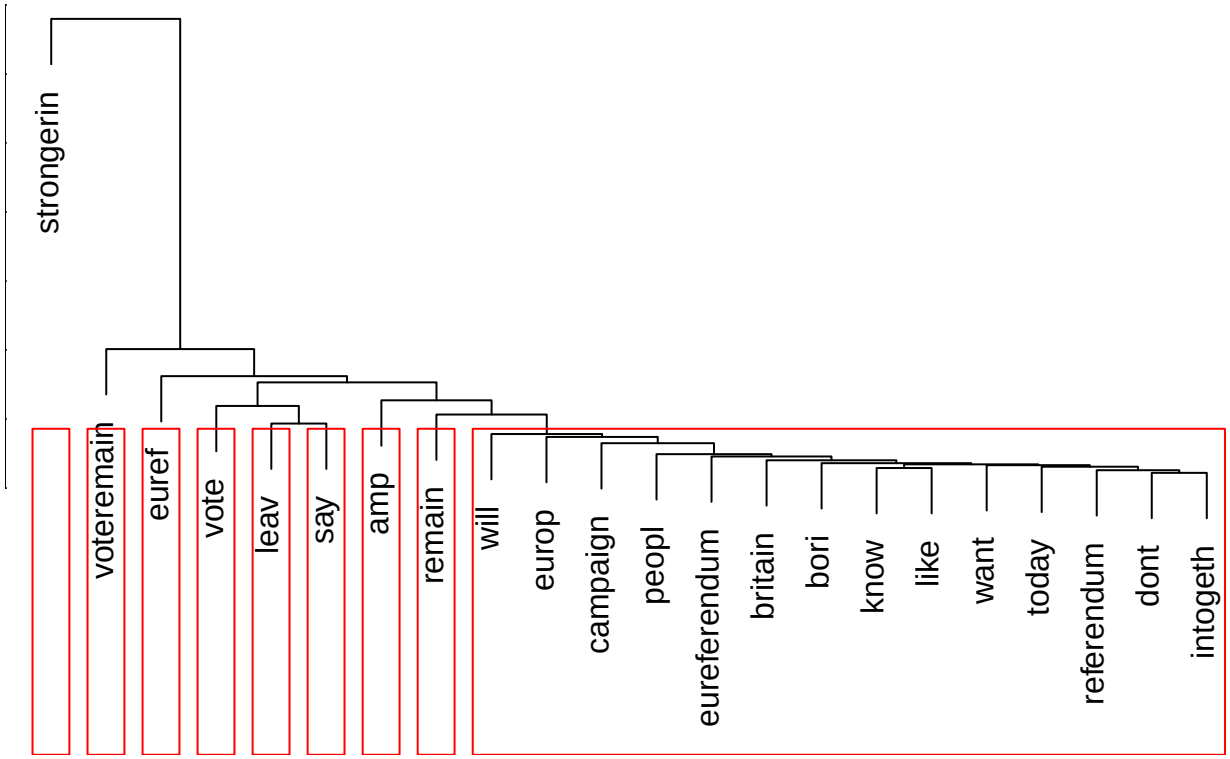



```
# Show the text clusters
visualiseTextClusters(tdm, cnstLr)
```




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


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




```
# 5.3 The most frequent words that appear in the tweets

# Display the table with top 20 most frequent words
# The number 20 is chosen to provide a wider understanding of the model
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

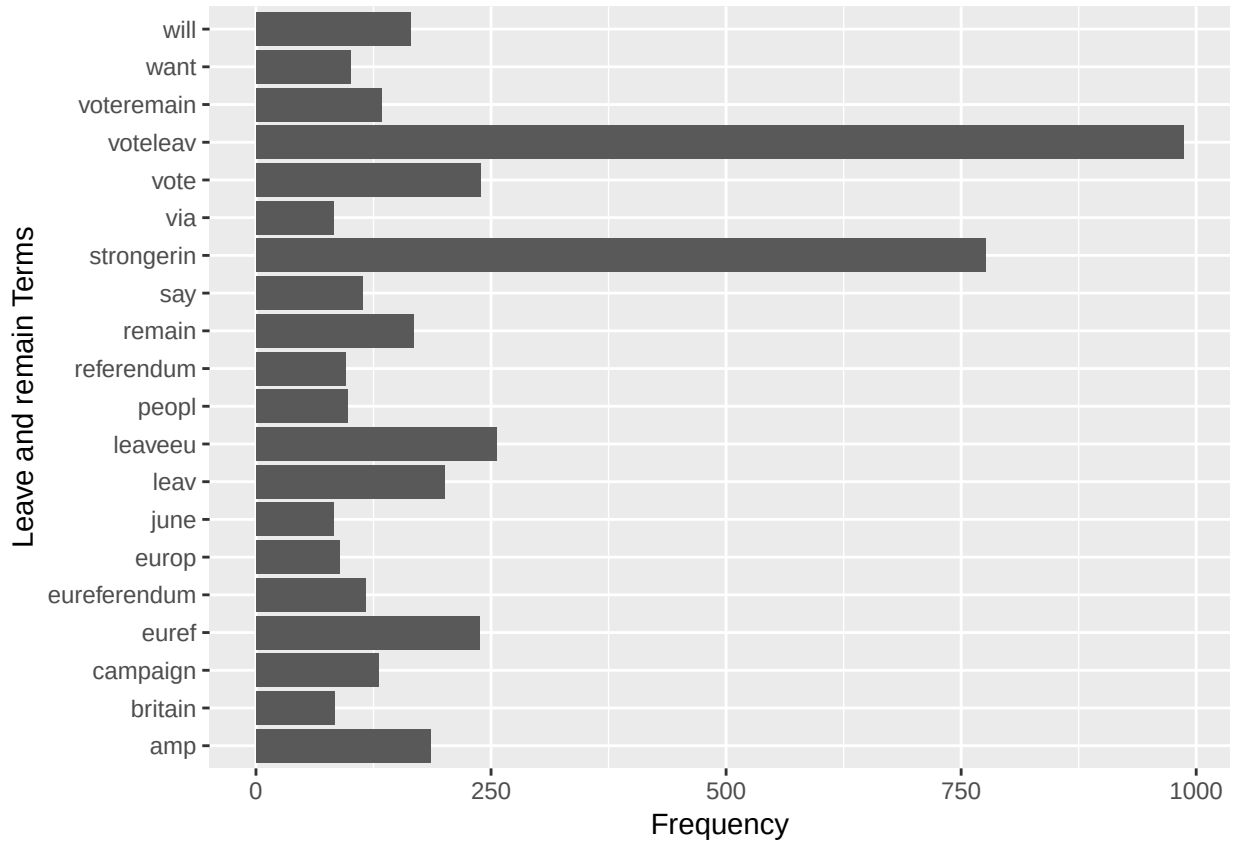
```
printFrequency(dfTermsFreqLeave, 20, cnstLea)
```

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

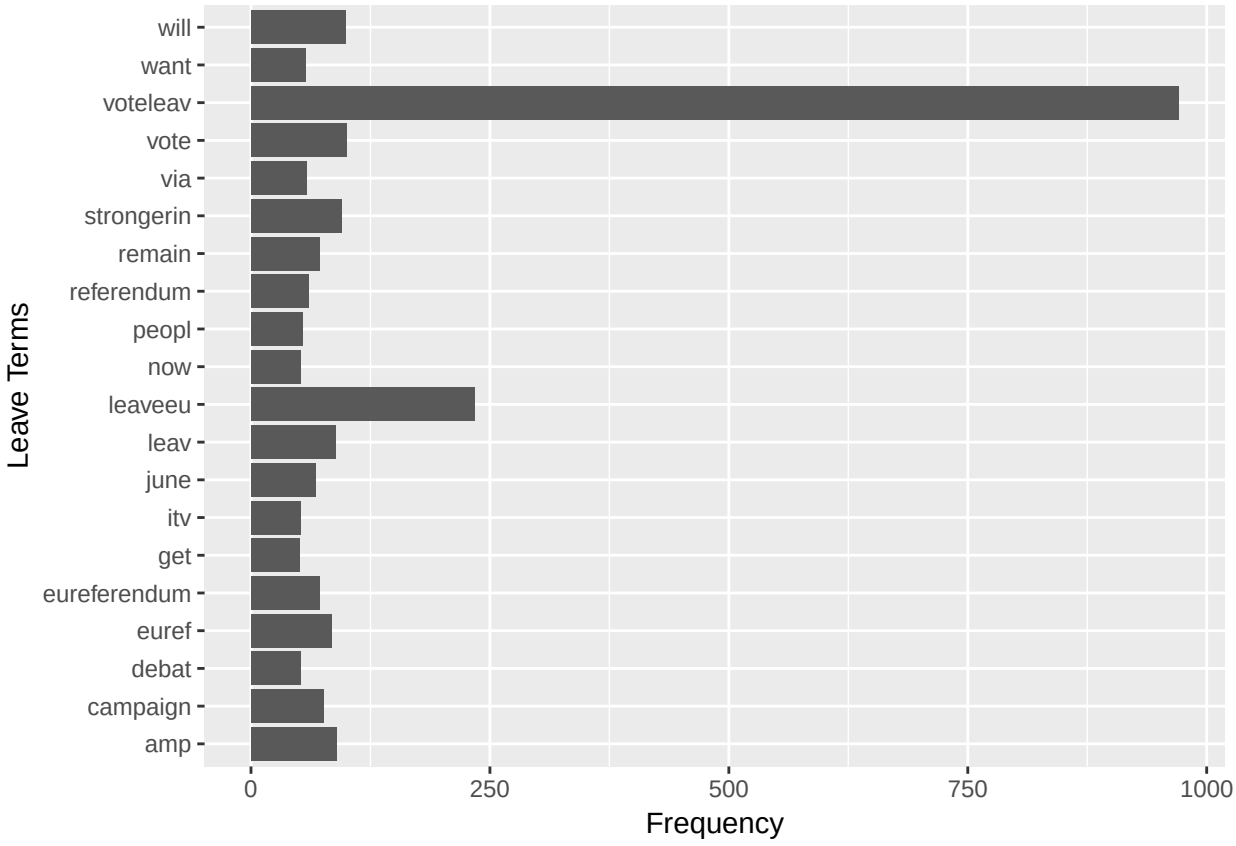
```
printFrequency(dfTermsFreqRemain, 20, cnstRem)
```

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

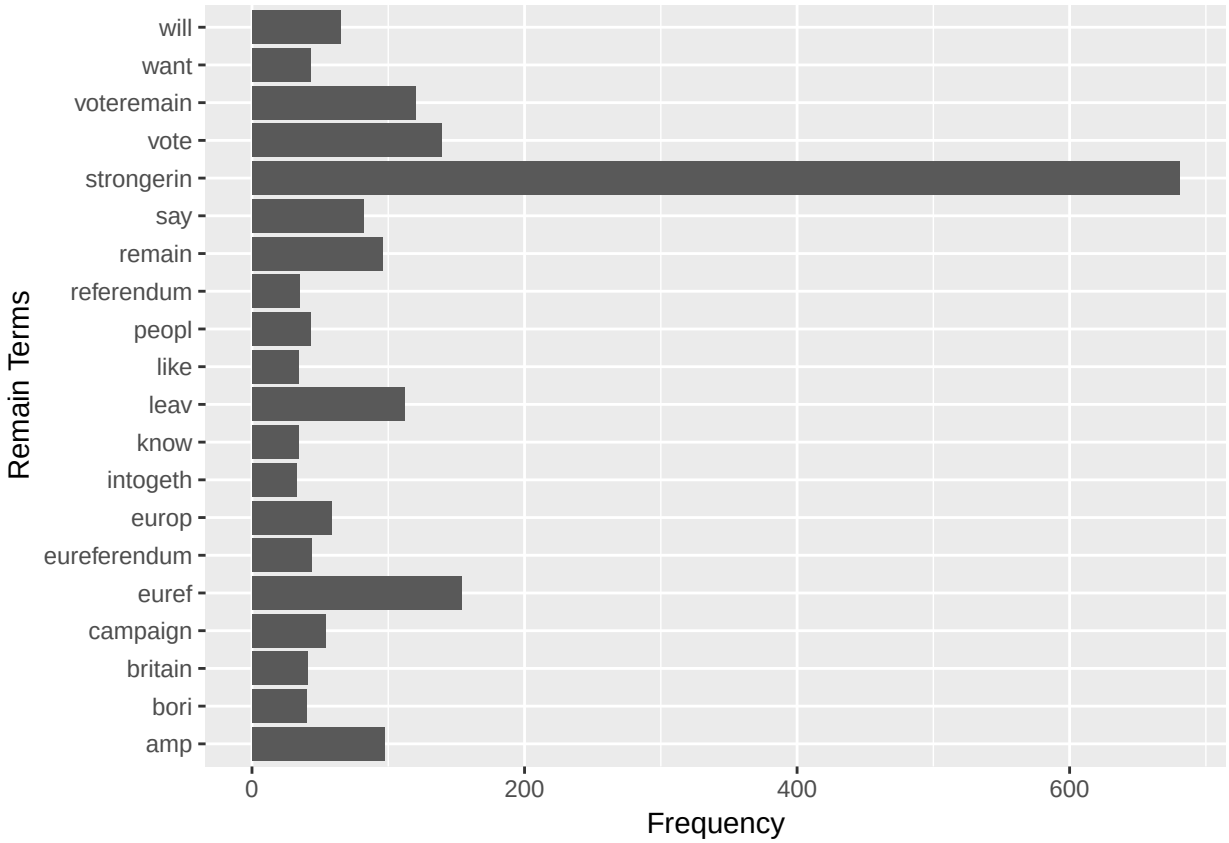
```
# Display the diagram with the top 20 most frequent words
displayPlot(dfTermsFreq, 20, cnstLr)
```




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




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


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

TEXT CLASSIFICATION

Naive Bayes approach

```
# 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
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)
}
```

```
# 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)

}
```

```
# 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'))
  }
}
```

```
# First, let us find out the class/label distribution
table(df$label)
```

```
 Leave Remain
  1254   1029
```

```
# The DocumentTermMatrix() 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%
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
```

```
# Run Naive Bayes classifier with different feature reduction sizes
# to find the optimum feature set
lowFreqVector = c(5,7,10,15)
predictions <- runNaiveBayesClassifier(lowFreqVector)
printAccuracyTable(predictions)
```

Feature reduction	Accuracy
5	0.892778993435449
7	0.897155361050328
10	0.905908096280088
15	0.899343544857768

```
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 |           |
-------------|-----------|-----------|-----------|
```

```
# 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 docu
# and a smoothing parameter set to 0.5
# REF: https://www.quora.com/What-is-Laplacian-smoothing-and-why-do-we-need-it-in-a-Naive-Bayes-classif
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 set to 0.5 increases accuracy by roughly 1% to 0.910284463894967
```

RTextTools approach

REF: https://journal.r-project.org/archive/2013/RJ-2013-001/RJ-2013-001.pdf

Using the prefix rtt for each variable to make it stand out from the other models

```
# This function creates a data frame containing the algorithms by their performance
# determined by 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))
}
```

```
# Create the document term matrix
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)

# Train the model with SVM, BOOSTING, MAXENT, RF, TREE, SLDA, BAGGING
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
rttFScores <- getAlgorithmsPerformance()

# Sort the data frame in the descending order and print
rttFScores <- rttFScores[with(rttFScores, order(-MeanF1Score)), ]
kable(rttFScores)
```

	Algorithm	MeanF1Score
5	FORESTS	0.930
1	SVM	0.920
3	LOGITBOOST	0.915

	Algorithm	MeanF1Score
2	SLDA	0.900
4	BAGGING	0.895
7	MAXENTROPY	0.890
6	TREE	0.880

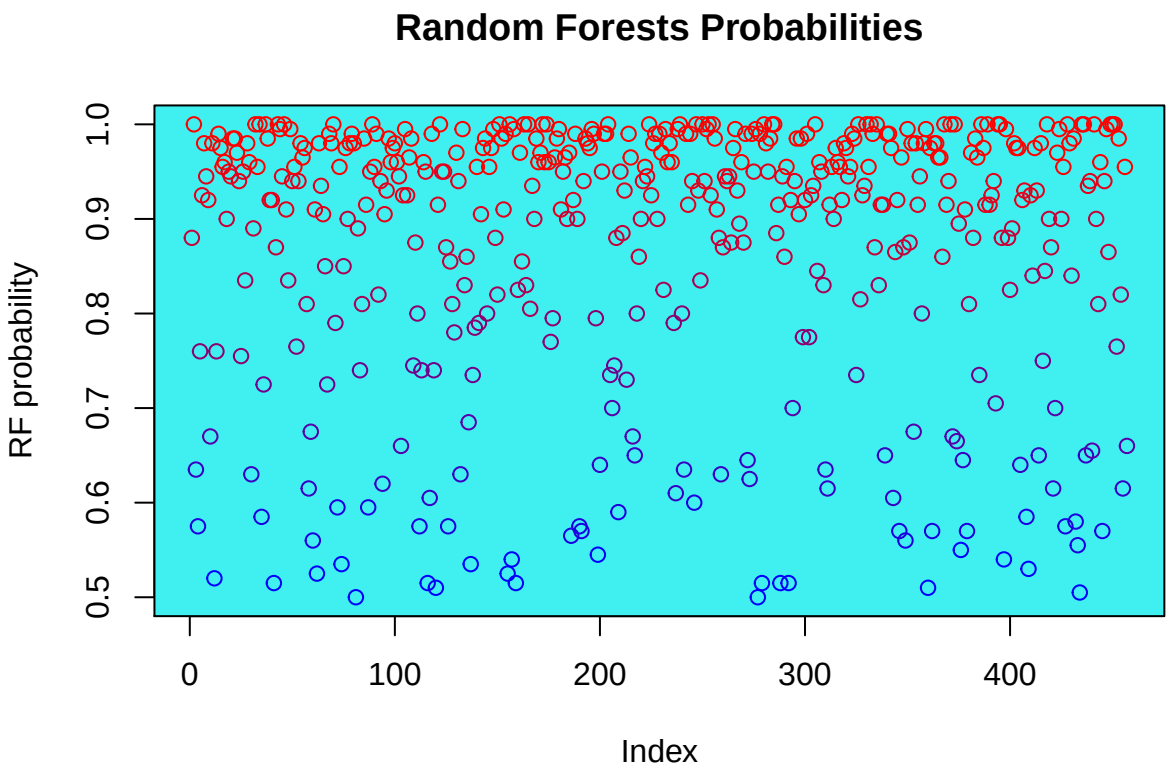
```
# We can see that the two best performing algorithms are Random forests and Support vector machine
# The following will now display confusion matrix for RF and SVM
for (i in 1:2)
{
  algName <- as.character(rttFScores[i,1])
  parts <-  c('rttResults$', algName, '_LABEL')
  modelLabel <- paste(parts, 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
```

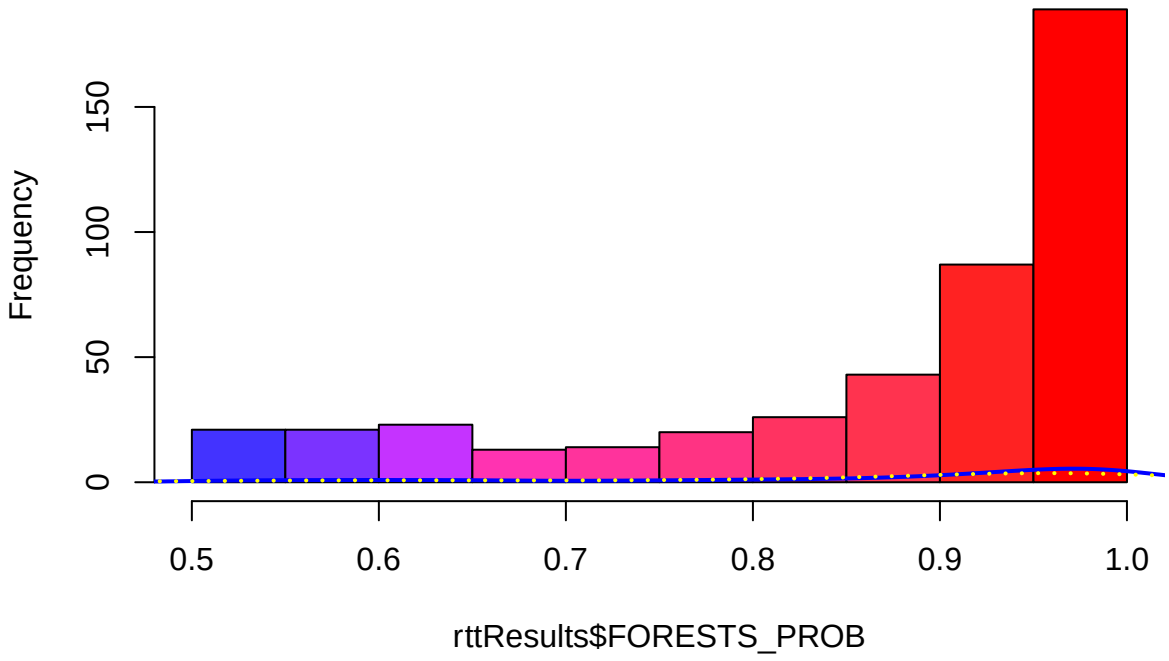
```
               Accuracy : 0.919
                 95% CI : (0.8901, 0.9424)
    No Information Rate : 0.5711
    P-Value [Acc > NIR] : < 2e-16

                  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
```

```r
# Random forests shows only 30 mis-classified instances out of 457 with the following metrics:
# an overall accuracy of 0.9344, Kappa value of a very high 0.867
# Leave prediction rate is very high at 0.9639
```

```r
# Some interesting metrics
# Plot RF 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","#FF
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)
```



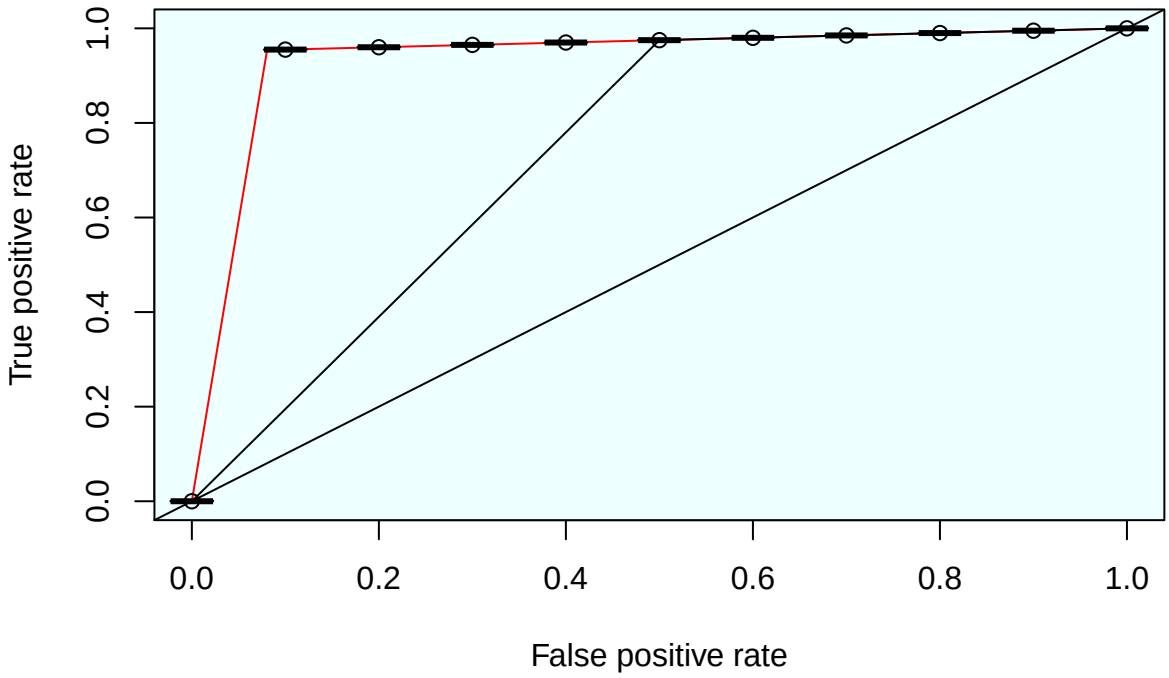

```
# Show the number of probabilities where P(prediction)>=0.75 and a percentage of it
highProb <- length(rttResults$FORESTS_PROB[rttResults$FORESTS_PROB >= 0.75])
print(highProb)
```

```
[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.

There is more discussion at https://stats.stackexchange.com/questions/132777/what-does-auc-stand-for-and-what-is-it

```
# Let us experiment with the ntree parameter in Random forests algorithm
# to see if we can further improve the performance. 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))
}

# The best result is achieved by ntree=100 but is no improvement to what we already have
```

POTENTIAL ADDITIONAL WORK: try the neural network

POTENTIAL ADDITIONAL WORK: try words correlation a bit more, maybe show a diagram

POTENTIAL ADDITIONAL WORK: try to show a diagram to see if the RF algorithm overfits

CONCLUSION GOES HERE

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.