Coursework
Advanced Data Science (CMM536)

Andrew Tait, *1504693@rgu.ac.uk*

April 27, 2018

---

1 Research

The paper that was chosen for this work is [1]. The authors provided a full review on different streaming algorithms and method that handle data mining. Below is a review of this paper that includes problem statement, related work and methods applied.

1.1 Problem Statement

The process of data mining is to discover otherwise hidden information and patterns from already existing data. What makes this complicated when it comes to data streams is that unlike a traditional relation database or data warehouse there isn't a pre-defined structure. This paper outlines an algorithm that could possibly overcomes these difficulties.

1.2 Relevant Work

There is numerous related works and research in the field of data streaming. For instance the ACM KDDD international conference held in 2010, outlined the problem of finding the top-k frequent items in a data stream with flexible sliding widows. This would be only to mine the top-k frequent items, not reporting on all frequent items.

1.3 Methods

The paper outlines the issues that the new data streaming algorithm may face. This included memory constraint, data prepossessing choice of data structure, identifying data distributions and target concepts and dimensional reduction. These are import issues to cover as they will reduce the efficiency and overall accuracy of the data streaming algorithm.

1.4 Results

The algorithm produces a decision tree which it outlines the nodes that are active/live as black, killed nodes as red and green nodes on the first level to indicate that they are not closed. It produces an more effective decision tree it only looks for the top-k frequent items in the decision and discards the rest.

1.5 Conclusion

As the subject matter is dealing with an unlimited amount of information, whether this will be from social media streams or general information retrieval. There is still problem to overcome with clustering, classification and topic detection in terms of data streaming. The proposed algorithm provides a starting point by creating a decision tree of the top-k frequent items.

2 Data Streams

2.1 Dataset Choice

The dataset that has been chosen for this part of the coursework is the 'adult' dataset. This is available on the UCI repository. It has been proved to a good evaluation of classification methods
https://archive.ics.uci.edu/ml/datasets/adult

2.2 Data Exploration

The main purpose of the adult dataset is to find out which characteristics of the us population affect if their income is either ¡=50kor >=50k
To start off I will clear the RStudio environment and import the required libraries.

```
#Clean RStudio Environment
rm(list = ls())

#Import librarys
library(caret)
library(partykit)
library(mlbench)
library(RWeka)
library(C50)
library(datasets)
library(rpart)
library(ggplot2)
library(data.table)
library(stream)
library(mlbench)
library(doParallel)
library(streamMOA)
library(e1071)
library(RMOA)
library(ROCR)
library(tm)
library(wordcloud)
library(wordcloud,quietly=TRUE)
library(RColorBrewer,quietly=TRUE)
```

Then set the working directory to the coursework folder

```
#Set working directory
setwd("~/CMM536 Advanced Data Science/Coursework/CMM536_Coursework")
```

In order to the import the adult dataset, the feature names first need to be defined.

```
#Feature names
adultNames <- c("age", "workclass", "fblwgt",
                "education", "education-num",
                "martial-status",
                "occupation",
                "relationship",
                "race",
                "sex",
                "captial-gain",
                "captain-loss",
                "hours-per-week",
                "native-country",
                "class")
```

The adult dataset is then imported from adult.data file:

```
#Import datasets
adult <- read.table("data/adult.data" ,header = FALSE, sep = ",",
                         strip.white = TRUE, col.names = adultNames,
                         na.strings = "?", stringsAsFactors = TRUE)
```

Now the adult dataset is imported, it's time for some basic exploration
Number of rows (instances) in the dataset

```
nrow(adult)
```

```
## [1] 32561
```

The number of columns (features)

```
ncol(adult)
```

```
## [1] 15
```

Summary of the adult dataset:

```
#inspect dataset
str(adult)
```

```
## 'data.frame': 32561 obs. of  15 variables:
##  $ age           : int  39 50 38 53 28 37 49 52 31 42 ...
##  $ workclass     : Factor w/ 8 levels "Federal-gov",..: 7 6 4 4 4 4 4 6 4 4 ...
##  $ fblwgt        : int  77516 83311 215646 234721 338409 284582 160187 209642 45781 159449 ...
##  $ education     : Factor w/ 16 levels "10th","11th",..: 10 10 12 2 10 13 7 12 13 10 ...
##  $ education.num : int  13 13 9 7 13 14 5 9 14 13 ...
##  $ martial.status: Factor w/ 7 levels "Divorced","Married-AF-spouse",..: 5 3 1 3 3 3 4 3 5 3 ...
##  $ occupation    : Factor w/ 14 levels "Adm-clerical",..: 1 4 6 6 10 4 8 4 10 4 ...
##  $ relationship  : Factor w/ 6 levels "Husband","Not-in-family",..: 2 1 2 1 6 6 2 1 2 1 ...
##  $ race          : Factor w/ 5 levels "Amer-Indian-Eskimo",..: 5 5 5 3 3 5 3 5 5 5 ...
##  $ sex           : Factor w/ 2 levels "Female","Male": 2 2 2 2 1 1 1 2 1 2 ...
```

```
##  $ captial.gain  : int  2174 0 0 0 0 0 0 0 14084 5178 ...
##  $ captain.loss  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ hours.per.week: int  40 13 40 40 40 40 16 45 50 40 ...
##  $ native.country: Factor w/ 41 levels "Cambodia","Canada",..: 39 39 39 39 5 39 23 39 39 39 ...
##  $ class         : Factor w/ 2 levels "<=50K",">50K": 1 1 1 1 1 1 1 2 2 2 ...
```

Now that some basic data exploration is covered, next to inspect the dataset a bit further. Starting with the class distribution in the adult dataset, see (Figure 1)

```
#Class Distribution
barplot(table(adult$class))
```

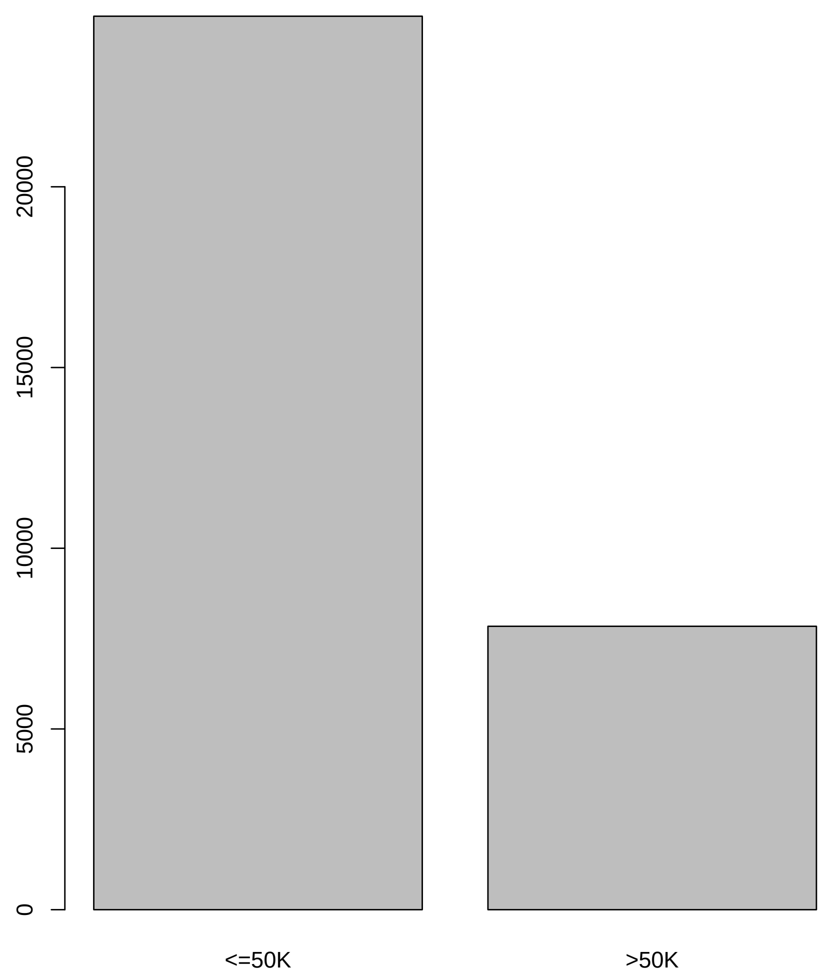

Figure 1: Barplot of Class Distribution

As the barplot clearly shows, the adult dataset is highly unbalanced with ¡=50k being the majority class by a large margin!

2.3 Build Classifier

2.3.1 Pre-Processing

Before the adult dataset can classified, some pre-processing is required first.
The dataset is checked for missing values ('?')

```
#Check for missing values ('?')
table(complete.cases (adult))
```

```
##
## FALSE  TRUE
##  2399 30162
```

As shown above there are missing values in the workclass,occupation and class columns. so these are removed.

```
cleanadult = adult[!is.na(adult$workclass)& !is.na(adult$occupation) & !is.na(adult$class),]
```

The flwgt feature is also removed as it's not required.

```
#Remove flwgt feature
cleanadult$fblwgt = NULL
```

Lets inspect the cleanadult dataframe before going further:

```
str(cleanadult)
```

```
## 'data.frame': 30718 obs. of  14 variables:
##  $ age           : int  39 50 38 53 28 37 49 52 31 42 ...
##  $ workclass     : Factor w/ 8 levels "Federal-gov",..: 7 6 4 4 4 4 4 6 4 4 ...
##  $ education     : Factor w/ 16 levels "10th","11th",..: 10 10 12 2 10 13 7 12 13 10 ...
##  $ education.num : int  13 13 9 7 13 14 5 9 14 13 ...
##  $ martial.status: Factor w/ 7 levels "Divorced","Married-AF-spouse",..: 5 3 1 3 3 3 4 3 5 3 ...
##  $ occupation    : Factor w/ 14 levels "Adm-clerical",..: 1 4 6 6 10 4 8 4 10 4 ...
##  $ relationship  : Factor w/ 6 levels "Husband","Not-in-family",..: 2 1 2 1 6 6 2 1 2 1 ...
##  $ race          : Factor w/ 5 levels "Amer-Indian-Eskimo",..: 5 5 5 3 3 5 3 5 5 5 ...
##  $ sex           : Factor w/ 2 levels "Female","Male": 2 2 2 2 1 1 1 2 1 2 ...
##  $ captial.gain  : int  2174 0 0 0 0 0 0 0 14084 5178 ...
##  $ captain.loss  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ hours.per.week: int  40 13 40 40 40 40 16 45 50 40 ...
##  $ native.country: Factor w/ 41 levels "Cambodia","Canada",..: 39 39 39 39 5 39 23 39 39 39 ...
##  $ class         : Factor w/ 2 levels "<=50K",">50K": 1 1 1 1 1 1 1 2 2 2 ...
```

As the adult dataset is a mixture of factors, integers and characters, I decided that transforming the dataset into binary. This will be create features based on every possible value in the dataset.
The cleanadult dataframe is first copied and the class is removed (it's not being transformed into binary)

```
#Copy dataset
noClass <-cleanadult
#Remove class as it is not being transformed to binary
noClass$class <- NULL
```

Then the noClass dataframe is transformed into binary

```
binaryVars <- caret::dummyVars(~ ., data = noClass)
newAdult <- predict(binaryVars, newdata = noClass)
```

The class feature is then added to the binarised dataset

```
#add class to binarised dataset
binAdult <-cbind(newAdult, cleanadult[14])
```

Any rows with NA values after being binary transformed.

```
#remove any rows with NA values
row.has.na <- apply(binAdult, 1, function(x){any(is.na(x))})
sum(row.has.na)
binAdult <- binAdult[!row.has.na,]
```

Number of NA rows removed.

```
## [1] 556
```

2.3.2 Classification

When it came to dividing the mushroom dataset into training and testing subsets, I decided to go with 80 percent training and 20 percent testing split as a starting point/baseline. Since the class distribution is unbalanced, I thought this split would cover the majority of cases.

```
#split 80% training and 20% testing datasets
inTrain <- createDataPartition(y=binAdult$class, p=0.8, list=FALSE)

#Assign indexes to split the binAdult dataset into training and testing
training <- binAdult[inTrain,]
testing <- binAdult[inTrain,]
```

For the initial classifier I decided to go with the kNN Classifier as it has proven to be a good baseline in previous labs and exercises in R. Before the classification begins, parallel processing is enabled to speed up this process.

```
#Setup Parallel processing to speed up classification modelling
cl <- makeCluster(detectCores(), type='PSOCK')
registerDoParallel(cl)
```

The train control is set to repeated-cross-validation with 10 folds and 3 repeats

```
ctrl <- trainControl(method = "repeatedcv",
                     repeats = 3,
                     number = 10,
                     verboseIter=TRUE)
```

Next the seed is set to 1, in order to make the model reproducible and the kNN model is set up with the train control from above and k value set to 3.

```
# ensure reproducibility of results by setting the seed to a known value
set.seed(1)
#use knn
mod21.knn<- train(class~., data=training,
                  method="knn", tuneGrid=expand.grid(.k=3),trControl=ctrl)
```

```
## Aggregating results
## Fitting final model on full training set
```

Once the knn Model is complete, it's time to analyse the results, first with a print of the kNNModel as shown below.

```r
print(mod21.knn)
```

```
## k-Nearest Neighbors
##
## 24131 samples
##   104 predictor
##     2 classes: '<=50K', '>50K'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 21717, 21718, 21719, 21718, 21718, 21718, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.8361309  0.5541515
##
## Tuning parameter 'k' was held constant at a value of 3
```

On first inspection, the knnModel produces an accuracy of 83.61% and kappa value of 0.55. Further evaluation can be found in the next section.

2.3.3 Evaluation

To evaluate the module, a confusion matrix is produced by predicting the against the testing (20 percent of the total dataset) subset.

```
#Evaluation
predictkNN <- predict(mod21.knn,testing)
confusionMatrix(predictkNN, testing$class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction <=50K  >50K
##      <=50K 17026  1477
##      >50K   1098  4530
##
##                Accuracy : 0.8933
##                  95% CI : (0.8893, 0.8972)
##     No Information Rate : 0.7511
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7085
##  Mcnemar's Test P-Value : 9.399e-14
##
##             Sensitivity : 0.9394
##             Specificity : 0.7541
##          Pos Pred Value : 0.9202
##          Neg Pred Value : 0.8049
##              Prevalence : 0.7511
##          Detection Rate : 0.7056
##    Detection Prevalence : 0.7668
##       Balanced Accuracy : 0.8468
##
##        'Positive' Class : <=50K
##
```

As the confusion matrix shows that the kNNmodel produced an accuracy of 89.33% when predicted against the training subset.
It correctly classified 17026 rows as ¡=50k and 4530 rows as ¿=50k. However 1098 rows were incorrectly classified as ¡=50k and 1477 rows were incorrectly classified as ¿50k.
Overall it's a good start and could be improved upon further, with more computing resources and time to hand.

2.4 Build Stream Classifier

For the data streaming classifier, I used the binary adult dataset as used in the kNN model.
First the binary adult dataset is copied to a dataframe.

```
#Pre-processing
#copy the binary adult dataset
df <- binAdult
```

Next the control settings for the classification model are configured.

```
ctrl <- MOAoptions(model = "OCBoost", randomSeed = 123456789, ensembleSize = 25,
                   smoothingParameter = 0.5)
mymodel <- OCBoost(control = ctrl)
mymodel
```

```
## OCBoost modelling options:
## MOA model name: Online Coordinate boosting for two classes evolving data streams.
##   - baseLearner: trees.HoeffdingTree   (Classifier to train.)
##   - ensembleSize: 25   (The number of models to boost.)
##   - smoothingParameter: 0.5   (Smoothing parameter.)
##   - randomSeed: 123456789   (Seed for random behaviour of the classifier.)
```

After setting the classification model, it's time to create the data stream and set some variables to control the iteration over the stream (see for loop)

```
#Create datastream from the dataframe.
dfStream <-datastream_dataframe(data=as.data.table(df))
#Set variables for stream iteration
chunk <- 100
turns <- (nrow(dfStream$data)/chunk)-1
turns <- floor(turns)
position <- chunk
```

Next a sample of the dataset is created (first 100 rows)

```
#first sample (train)
sample <- dfStream$get_points(dfStream, n =chunk,
                              outofpoints = c("stop", "warn", "ignore"))
```

For verification that the sample and original rows are the same, the first 10 classes of each are displayed. Note there are 105 features from the binary dataset, so displaying them all will take up a lot of pages, so only the class is shown for simplicity.

```
head(sample$class,10)
head(df$class,10)
```

The first 10 classes of the sample chunk

```
##  [1] <=50K <=50K <=50K <=50K <=50K <=50K <=50K >50K  >50K  >50K
## Levels: <=50K >50K
```

The first 10 classes of the df dataframe.

```
##  [1] <=50K <=50K <=50K <=50K <=50K <=50K <=50K >50K  >50K  >50K
## Levels: <=50K >50K
```

With all the streaming data now setup, the model can be trained using the 1st chunk of data and then iterated against the whole of the data stream.

```r
##Train the first chunk

myboostedclasifier <- trainMOA(model=mymodel,
                      formula = class~.,
                      data = datastream_dataframe(sample))

#Now iterate ove the whole stream
for (i in 1:turns){
  #next sample

  sample <- dfStream$get_points(dfStream, n =chunk,
                                outofpoints = c("stop", "warn", "ignore"))

  #update the trained model with the new chunks
  myboostedclasifier <- trainMOA(model = myboostedclasifier$model,
                        formula = class~.,
                        data = datastream_dataframe(sample),
                        reset = FALSE, trace=FALSE)

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

Now that first sample has been tested, lets make some predictions

```r
##Do some prediction to test the model

predictions <- predict(myboostedclasifier, sample)
table(sprintf("Reality: %s", sample$class),
      sprintf("Predicted: %s", predictions))


predictions <- as.factor(predictions)
confusion.mstream <- confusionMatrix(predictions, sample$class)

cat("Accuracy is: ", confusion.mstream$overall["Accuracy"])
```

```
Accuracy is:  0.86
```

With a sample of the data stream tested, lets do the same with the whole stream.

```r
#Hold results in a vector
accuracies <- c()
dfStream$reset()

for(i in 1:turns){
  #next sample
```

```r
  sample <- dfStream$get_points(dfStream, n=chunk,
                                outofpoints = c("Stop", "warn", "ignore"))
  predictions <- predict(myboostedclasifier, sample)
  #Convert predictions to a factor so it will work with confusion matrix
  predictions <- as.factor(predictions)

  #caculate accuracy
  confusion.mstream <- confusionMatrix(predictions, sample$class)
  accuracies[i] <- confusion.mstream$overall["Accuracy"]

  cat(accuracies[i],"%","\n")

}
```

Check the head of the accuracies

```r
head(accuracies)
```

```
## [1] 0.8181818 0.8300000 0.8300000 0.8400000 0.8300000 0.8300000
```

Finally lets plot the accuracy vs the chunk number

```r
plot(accuracies,type='l',col='red',
     xlab="Chunk Number",ylab="Accuracy",frame=FALSE)
```

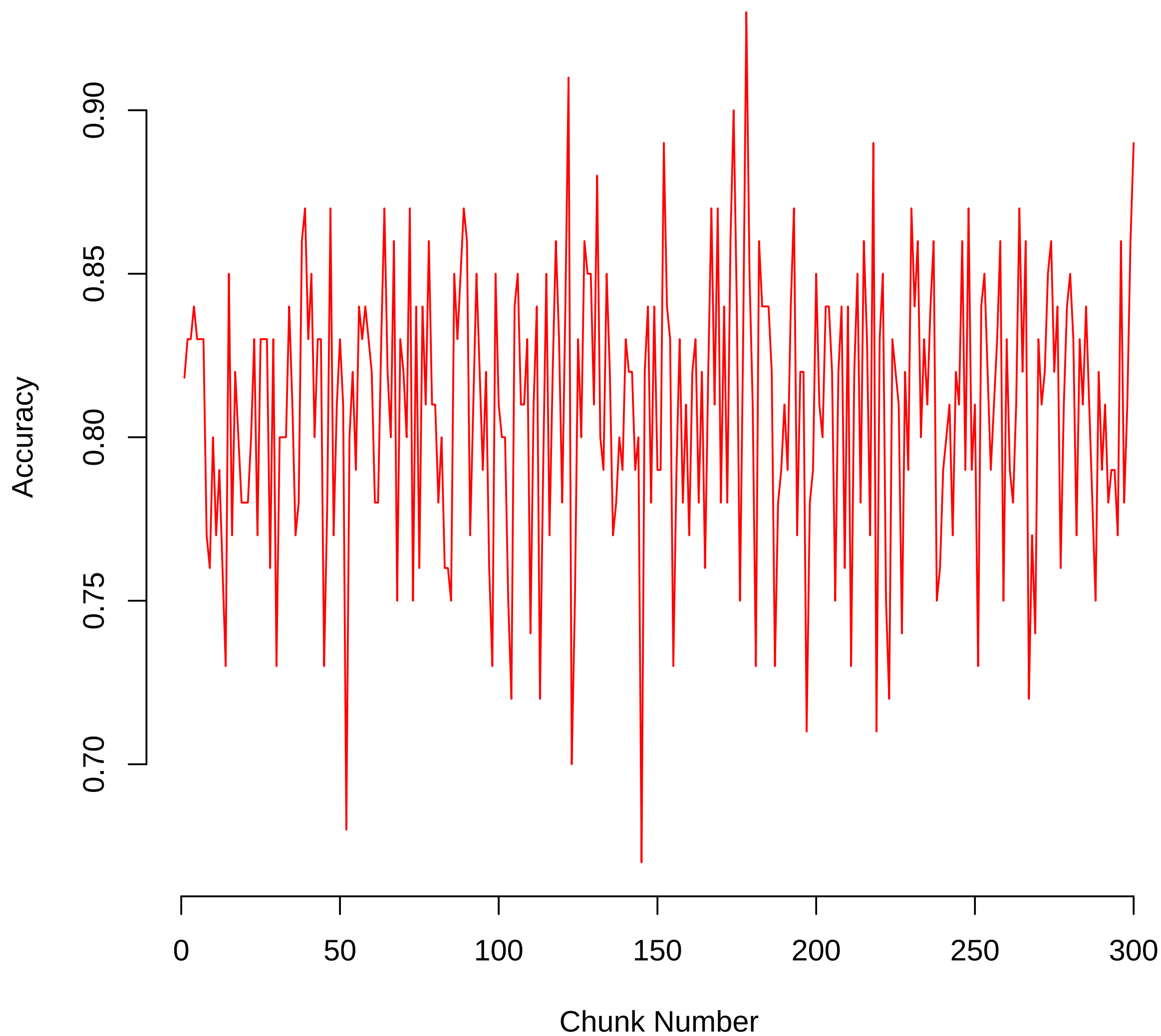

Figure 2: Chunk Number vs Accuracy

As the plot above shows, the overall accuracy varies as the chunk number increases. Although it drops to bellow 70% just after 50 chunks and again just before 150 chunks. The highest accuracy value is between 170 and 190 chunks.
The limitation for data streaming is that the adult dataset is unbalanced, also to improve classification models more computing resources would be required. As this is a length process that is restricted to the computing power provided on my laptop.

3 Text Classification

For this task a csv of leave/remain tweets of the brexit campaign was provided.
The was first imported:

```
leaveRemainTweets <- read.csv("data/leaveRemainTweets_CW.csv", header=TRUE)
```

Then the tweets are split based on the label(leave/remain)

```
#split dataset into leave and remain dataframes
tweets = split(leaveRemainTweets, leaveRemainTweets$label)
remainTweets = tweets$Remain
leaveTweets = tweets$Leave
```

Next lets check how many leave/remain tweets they are

```
nrow(remainTweets)
nrow(leaveTweets)
```

Number of remain tweets

```
## [1] 1029
```

Number of leave tweets

```
## [1] 1254
```

3.1 Preprocessing

In order to pre-process the leaveRemain tweets, I built a custom corpus function. Which can be found below.

```
buildCorpus <- function (tweets, wordCloud, wholeDataSet, wordAssocation, dtm){

  #Check the optional parameters are set
  wordA <-hasArg(wordAssocation)
  wds <- hasArg(wholeDataSet)
  wordc <-  hasArg(wordCloud)
  dtmcheck <- hasArg(dtm)

  corp <- Corpus(VectorSource(tweets$text))


  corp <- tm_map(corp,
              content_transformer(function(x) iconv(x, to='ASCII',
                                                    sub='byte')))
  # remove stop words and other preprocessing

  corp <- tm_map(corp, content_transformer(tolower))
  corp <- tm_map(corp, removeNumbers)

  toSpace = content_transformer( function(x, pattern) gsub(pattern," ",x) )
```

```
##Tweet cleaning

#credit to https://stackoverflow.com/a/31352005/8816204
corp <-tm_map(corp, toSpace, "(RT|via)((?:\\b\\W*@\\w+)+)")
corp <-tm_map(corp, toSpace, "@\\w+")
corp <-tm_map(corp, toSpace, "&amp")

##Remove URLS from tweets
corp <-tm_map(corp, toSpace, "httpsw+")
corp <-tm_map(corp, toSpace, "http:\\w+")
corp <-tm_map(corp, toSpace, "https:\\w+")

corp <-tm_map(corp, toSpace, "[ \t]{2,}")
corp <-tm_map(corp, toSpace, "^\\s+|\\s+$")


# ##Remove obvious words /stopwords
if(wds){
  if(wholeDataSet){
    corp <- tm_map(corp, function(x)removeWords
                  (x,c(stopwords("english"),"amp", "will", "%Û_", "https", "http", "httpsdb"
                       , "eu", "brexit", "rt", "leave", "remain", "vote")))
  }
}
else{
  corp <- tm_map(corp, function(x)removeWords
                (x,c(stopwords("english"),"amp", "will", "%Û_", "https", "http", "httpsdb")))
}



#remove punctuation last so urls are removed correctly
corp <- tm_map(corp, removePunctuation)


#If wordAssocation is true then create term document  matrix
if(wordA){
  if(wordAssocation){
    tdm <- TermDocumentMatrix(corp)
  }
  else{
    if(dtm){  # if dtm is true the create document term matrix
      dtm <- DocumentTermMatrix(corp)
    }
  }

}

else if(!wordA){ # If no  wordAssocation then create TermDocument Matrix
  tdm <- TermDocumentMatrix(corp)

  m <- as.matrix(tdm)
  v <- sort(rowSums(m), decreasing = TRUE)
```

```r
    d <- data.frame(word = names(v), freq = v)
    d$word <- gsub("~", " ", d$word) ## Edit 2


    if(wordc){
      if(wds){
        tweets <- d$word #whole dataset return df of processed tweets
      } else{
        tweets <- d$word
        if(wordCloud){ #return wordcloud if TRUE
          wordcloud(words =tweets, freq = d$freq, min.freq = 3,
                    max.words=2000, random.order=FALSE, rot.per=0.2,
                    colors=brewer.pal(8, "Dark2"))
        }
      }

    }

  }

}
```

By using the buildCorpus function with the word cloud parameter set to TRUE, it will create a word cloud for us.

```r
leaveWordCloud <- buildCorpus(leaveTweets,TRUE)

remainWordCloud <- buildCorpus(remainTweets,TRUE)
```

Please see (Figure 3) for leave tweets word cloud and (Figure 4) for remain tweets wordcloud.

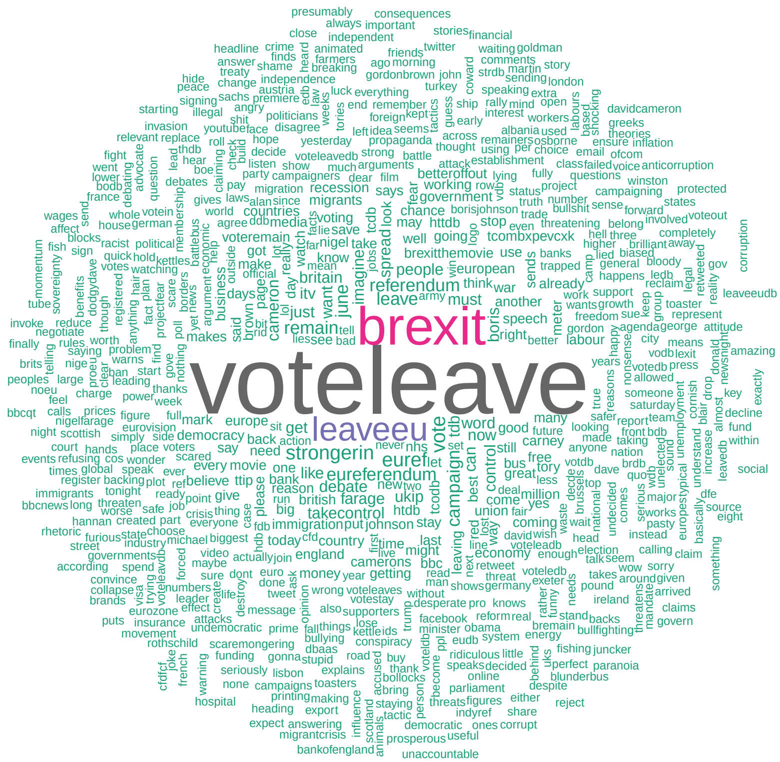

Figure 3: Word cloud of leave tweets

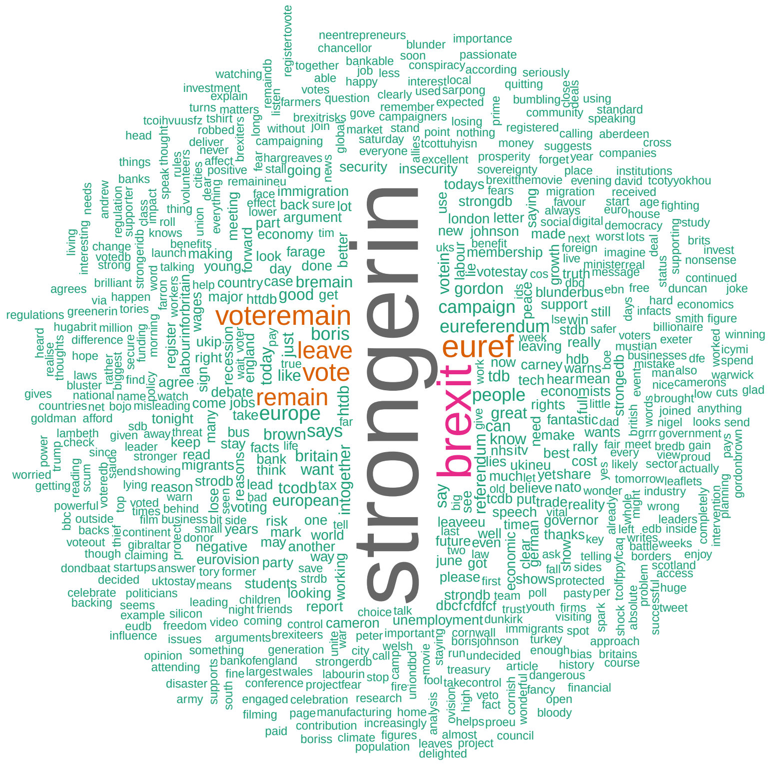

Figure 4: Word cloud of Remain tweets

3.2 Text Analysis

3.2.1 Most Frequent word in Tweet Collection

To find the most frequent word in the collection of tweets, the whole leaveRemainTweets dataframe was based into the buildCorpus function with the wordCloud option set to FALSE and wholedataset option set to TRUE.

```
#Get Most frequent word
tweetWordCloud <-buildCorpus(leaveRemainTweets, FALSE, TRUE)
head(tweetWordCloud,1)
```

```
## [1] "voteleave"
```

3.2.2 Word Association

To find out which words appear together often, I used the buildCorpus function to return a termDocumentMatrix. The most frequent terms with a lowfreq value of 50 is also created.

```
wordAssoication <- buildCorpus(leaveRemainTweets,FALSE,TRUE,TRUE)
freq.terms <- findFreqTerms(wordAssoication, lowfreq =50)
```

Lets inspect the top 30 most frequent words in the tweet collection

```
head(freq.terms,30)
```

```
##  [1] "strongerin"   "voteleave"    "referendum"   "tdb"
##  [5] "boris"        "says"         "leaveeu"      "like"
##  [9] "tcodb"        "want"         "europe"       "get"
## [13] "britain"      "know"         "imagine"      "june"
## [17] "now"          "euref"        "eureferendum" "think"
## [21] "can"          "campaign"     "voteremain"   "debate"
## [25] "farage"       "just"         "must"         "people"
## [29] "european"     "good"
```

Now to plot (Figure 5) the word association (term document matrix) against the frequent terms.

```
plot(wordAssoication, term = freq.terms, corThreshold = 0.12, weighting = T)
```

In the plot, the thicker the line between words the greater the association. In this case June as a strong association with imagine and vote leave.
Similar with European with 'must' and 'now'. Also 'Farage' and 'debate'.

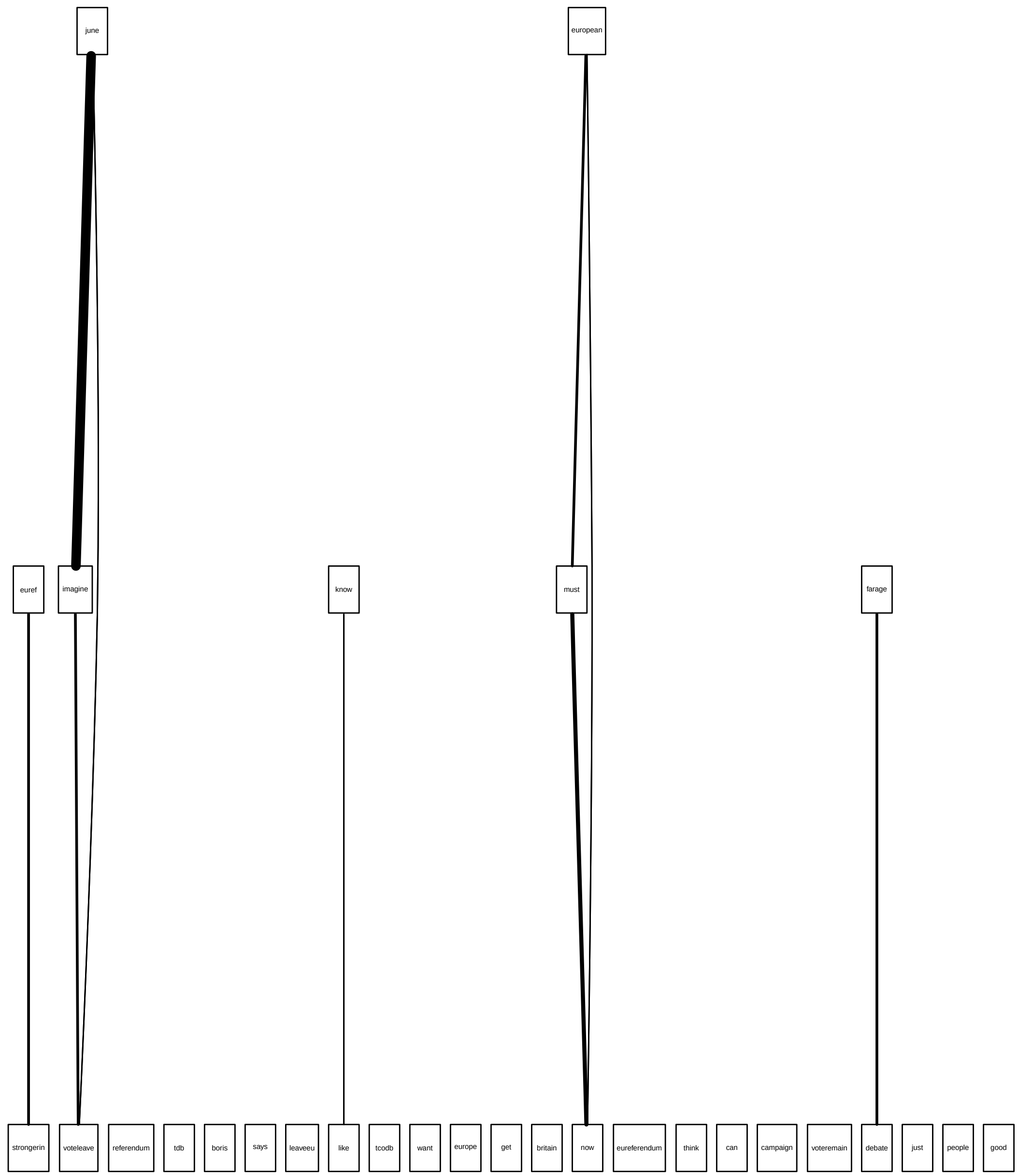

Figure 5: Word Association Plot (Zoom to see words)

3.2.3 Most Frequent Words in Both Leave & Remain Tweets

```
#Leave
leaveWords <- buildCorpus(leaveTweets, FALSE,TRUE)
head(leaveWords, 10)
```

```
##  [1] "voteleave"    "leaveeu"      "strongerin"   "euref"
##  [5] "eureferendum" "june"         "referendum"   "campaign"
##  [9] "now"          "people"
```

```
#Remain
remainWords <- buildCorpus(remainTweets, FALSE,TRUE)
head(remainWords, 10)
```

```
##  [1] "strongerin"   "euref"        "voteremain"   "europe"
##  [5] "says"         "eureferendum" "campaign"     "people"
##  [9] "boris"        "britain"
```

3.3 Text Classification

3.3.1 kNNModel

For the text classification of the brexit tweets, I decided to use the kNN Model again, as it has proven to be good starting/base line in previous exercises.
First off I used my buildCorpus function again but set the wholedataset parameter to TRUE, along with the dtm parameter to TRUE.
This creates a document term matrix, that is changed to a matrix and dataframe in turn. The tweet label is added to the new dataframe.

```
#Create a document term matrix from build corpus function.
dtm <-  buildCorpus(leaveRemainTweets,FALSE,TRUE,FALSE,TRUE)

#Convert dtm to matrix
dtm <- as.matrix(dtm)
#Then a data frame
tweets <- data.frame(dtm)
#Add label to dataframe
tweets$label <- leaveRemainTweets$label
```

The tweets are then separated into 70% training and 30% testing subsets (prevent over fitting from occurring).

```
#Split data into training and testing subsets
#Divide the dataset into 70% training and 30% testing, to prevent overfitting from occurring
inTrainTweets <- createDataPartition(y=tweets$label, p=0.7, list=FALSE)

#Assign indexes to split the Tweets DTM into training and testing
training <- tweets[inTrainTweets,]
testing <- tweets[-inTrainTweets,]
```

The kNN Model is then setup with cross validation (5 fold) and a tune length of 5.

```
train_control<- trainControl(method="cv", number=5,verboseIter=FALSE)

kNNTweets <- train(label~., data = training,
                  trControl = train_control,
                  tuneLength =5,
                  method = "knn"

)
```

```
kNNTweets
```

```
## k-Nearest Neighbors
##
## 1599 samples
## 6027 predictors
##    2 classes: 'Leave', 'Remain'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1280, 1280, 1278, 1279, 1279
```

```
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    5  0.8480166  0.6848774
##    7  0.8423955  0.6728668
##    9  0.8442646  0.6769512
##   11  0.8455068  0.6795951
##   13  0.8417646  0.6717658
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 5.
```

At first inspect of the kNN Model it produces an accuracy of 84.80% and a kappa value of 68.48%

```
#Evaluate

predictkNNTweets <- predict(kNNTweets,testing)
confusionMatrix(predictkNNTweets, testing$label)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Leave Remain
##     Leave    372    108
##     Remain     4    200
##
##                Accuracy : 0.8363
##                  95% CI : (0.8064, 0.8632)
##     No Information Rate : 0.5497
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6588
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9894
##             Specificity : 0.6494
##          Pos Pred Value : 0.7750
##          Neg Pred Value : 0.9804
##              Prevalence : 0.5497
##          Detection Rate : 0.5439
##    Detection Prevalence : 0.7018
##       Balanced Accuracy : 0.8194
##
##        'Positive' Class : Leave
##
```

When predicted against the testing subset the kNNModel, it produces an accuracy of 83.63%, which isn't too bad but not excellent either. As it incorrectly classifies 108 leave tweets and 4 remain tweets incorrectly.

3.3.2 Improved SVM Model

To improve the classification model, I decided to use the supported vector machine algorithm. The parameters are the same as the knnModel to make a fairer comparison.

```
#Set Train Control to cross-validation 5 fold
train_control<- trainControl(method="cv", number=5,verboseIter=FALSE)

#Set model to SVM with tunelength of 10

svmModel <- train(label~ ., data = training,
                  trControl = train_control,
                  tuneLength =10,
                  method = "svmRadial"

)
```

```
svmModel
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 1599 samples
## 6027 predictors
##    2 classes: 'Leave', 'Remain'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1279, 1279, 1278, 1280, 1280
## Resampling results across tuning parameters:
##
##   C       Accuracy   Kappa
##     0.25  0.8986713  0.7986826
##     0.50  0.8949076  0.7912300
##     1.00  0.8967806  0.7946679
##     2.00  0.9024134  0.8043444
##     4.00  0.9049076  0.8093912
##     8.00  0.9049076  0.8093912
##    16.00  0.9049076  0.8093912
##    32.00  0.9049076  0.8093912
##    64.00  0.9049076  0.8093912
##   128.00  0.9049076  0.8093912
##
## Tuning parameter 'sigma' was held constant at a value of 0.06818182
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.06818182 and C = 4.
```

The highest accuracy the SVM model produces is 90.49% at the c value of 4.

```r
#Evaluate

predictSVM <- predict(svmModel,testing)
confusionMatrix(predictSVM, testing$label)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Leave Remain
##     Leave    346     20
##     Remain    30    288
##
##                Accuracy : 0.9269
##                  95% CI : (0.9048, 0.9453)
##     No Information Rate : 0.5497
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8528
##  Mcnemar's Test P-Value : 0.2031
##
##             Sensitivity : 0.9202
##             Specificity : 0.9351
##          Pos Pred Value : 0.9454
##          Neg Pred Value : 0.9057
##              Prevalence : 0.5497
##          Detection Rate : 0.5058
##    Detection Prevalence : 0.5351
##       Balanced Accuracy : 0.9276
##
##        'Positive' Class : Leave
##
```

When compared against the testing subset, the Accuracy is 92.69% , which is a nearly 10% performance increased against the kNNModel. With only 20 leave tweets and 30 remain tweets incorrectly classified.

4 References

[1] M.S.B. PhridviRaj and C.V. GuruRao. “Data Mining – Past, Present and Future – A Typical Survey on Data Streams”. In: *Procedia Technology* 12 (2014). The 7th International Conference Interdisciplinarity in Engineering, INTER-ENG 2013, 10-11 October 2013, Petru Maior University of Tirgu Mures, Romania, pp. 255 –263. ISSN: 2212-0173. DOI: https://doi.org/10.1016/j.protcy.2013.12.483. URL: http://www.sciencedirect.com/science/article/pii/S2212017313006683.