Coursework
Advanced Data Science (CMM536)

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

April 26, 2018

---

1 Research

[your text goes here] The paper that was chosen for this work is [1]. The authors provided a full review on different streaming algorithms and methos that handle concept drfit?. Below is a review of this paper that includes problem statement, related work and methods applied.

Notice how I cited the paper, and how does it appear in the document, to do so, you need to have a file in your working director (references.bib), this file simply contains the bibtext items for the papers you chose. These BibTex items are often avaiable to download from publishers website, see Figure 1

1.1 Problem Statement

What is this paper about? your text goes here, your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here

1.2 Relevant Work

[your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here]

1.3 Methods

[your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here]

1.4 Results

[your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here]

1.5 Conclusion

[your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here your text goes here]

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 avaiable on the UCI repository.
https://archive.ics.uci.edu/ml/datasets/adult
The dataset that has been chosen for this part of the course work is IRIS. This is available on the UCI repository. The set was chosen because of .... It proves to be a good set for evaluating 'x' methods ....

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 libaries.

```
#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))
```

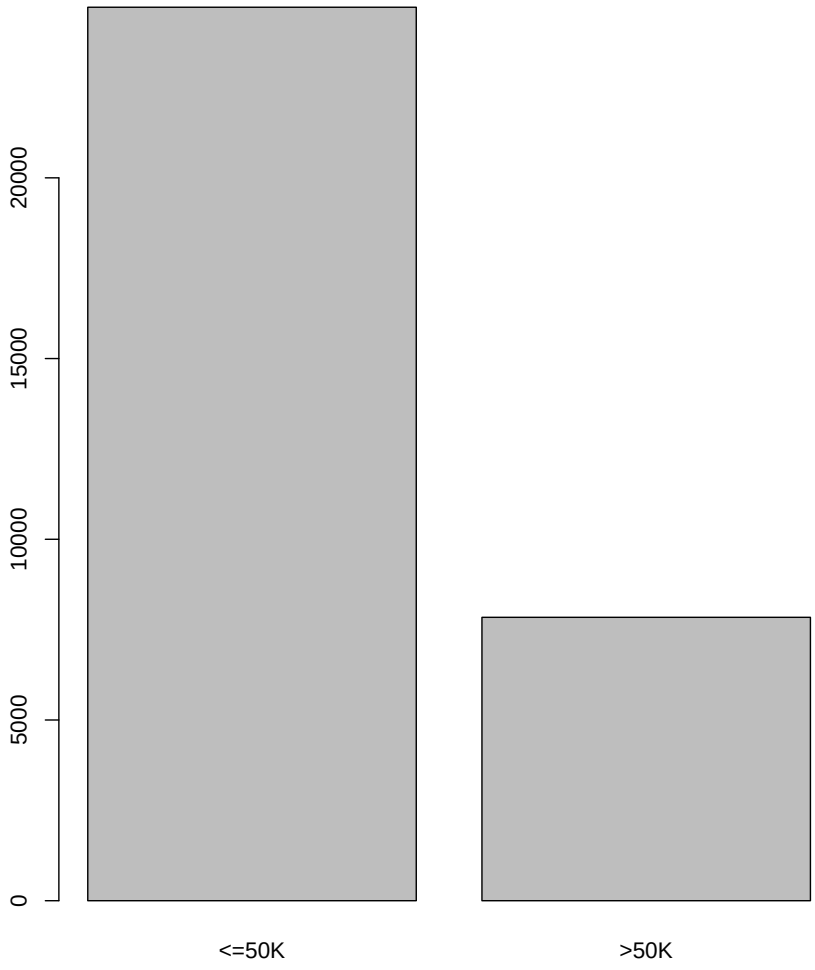


Figure 1: Barplot of Class Distribution

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 transforme dinto 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 distrubtion is unbalenced, 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.

```
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
```

2.3.3 Evalulation

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
##
```

2.4 Build Stream Classifier

For the data streaming classifer, 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 datastream 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.

```
##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

```
##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"])
```

```
## Error in .jcall("RJavaTools", "Z", "hasField", .jcast(x, "java/lang/Object"), :  java.lang.NullPointe
## Error in sprintf("Predicted:  %s", predictions):  object 'predictions' not found
## Error in is.factor(x):  object 'predictions' not found
## Error in confusionMatrix(predictions, sample$class):  object 'predictions' not found
## Error in cat("Accuracy is:  ", confusion.mstream$overall["Accuracy"]):  object 'confusion.mstream'
not found
```

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

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

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

  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")

}
```

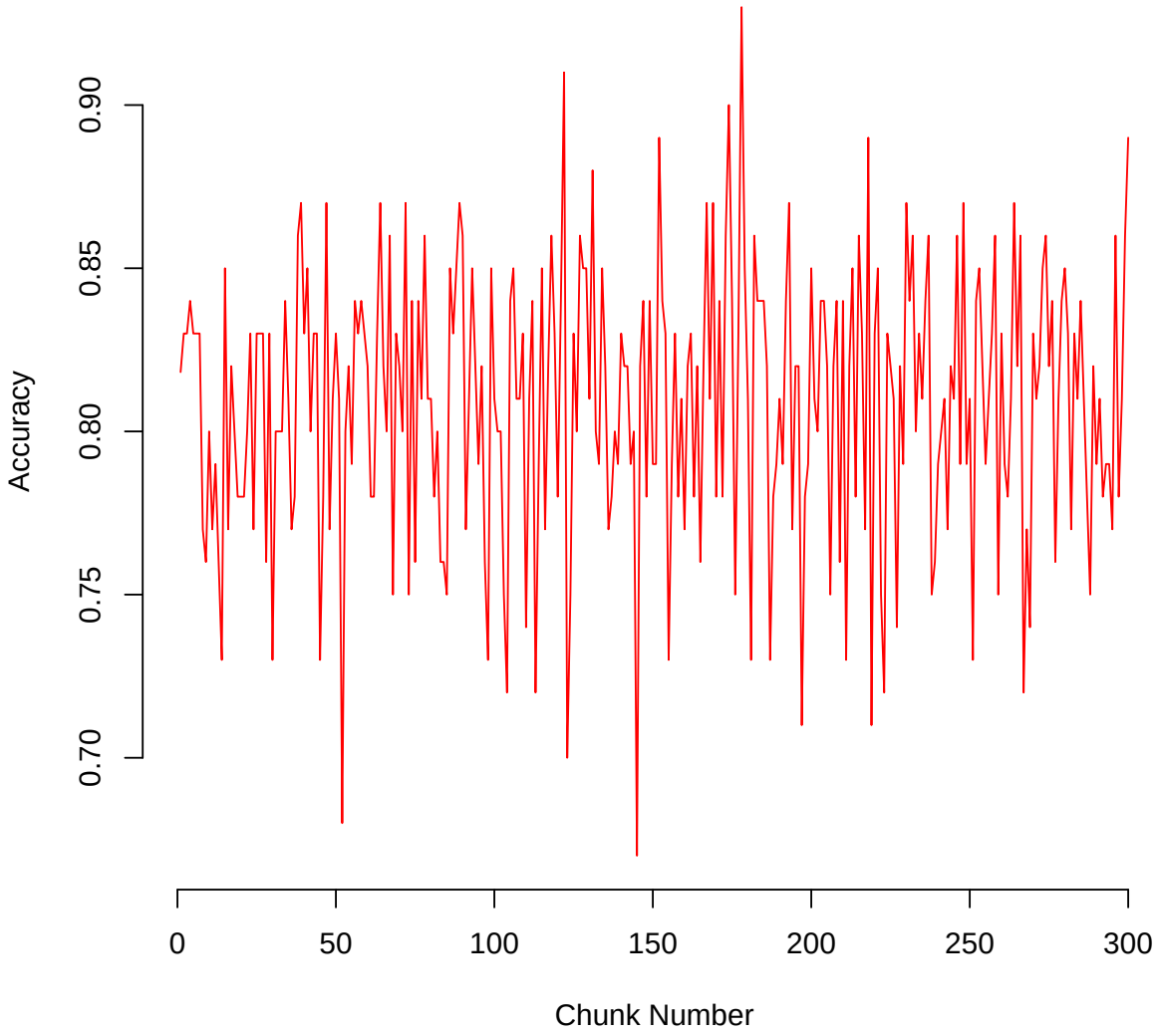
Check the head of the accuracies

```
head(accuracies)
```

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

Finnally lets plot the accuracy vs the chunk number

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


Accuracy
0.70
0.75
0.80
0.85
0.90
0
50
100
150
200
250
300
Chunk Number

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 tweeets

```
## [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){

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

  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"
   }
  } else{
    corp <- tm_map(corp, function(x)removeWords(x,c(stopwords("english"),"amp", "will", "%Û_", "https",
}



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

  #If wordAssocation is true then create document term matrix
  if(wordA){
    if(wordAssocation){
      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)
    d <- data.frame(word = names(v), freq = v)
    d$word <- gsub("~", " ", d$word) ## Edit 2


    if(wordc){
      if(wds){
        tweets <- d$word
      } else{
        tweets <- d$word
        if(wordCloud){
          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.

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

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

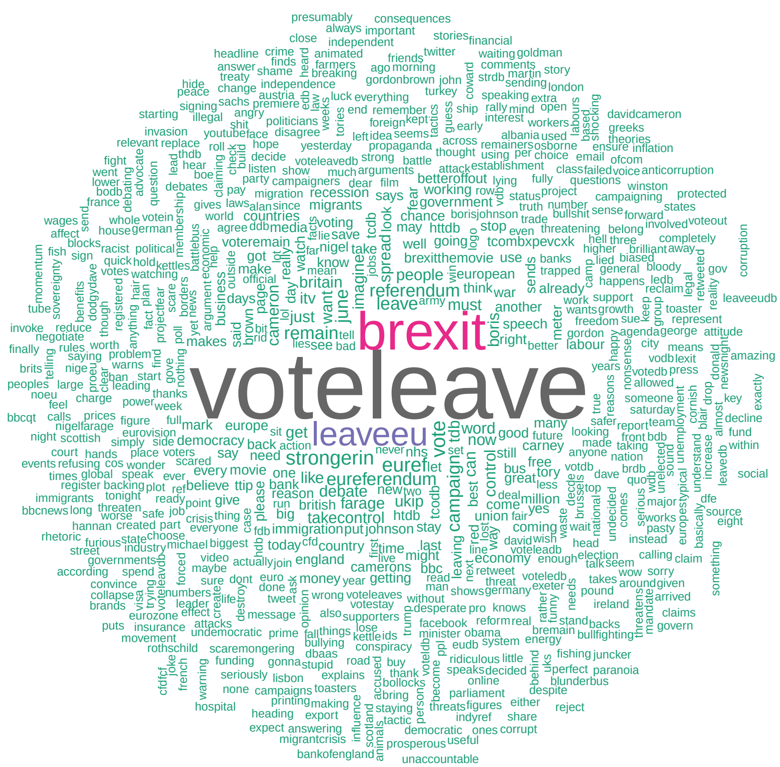


Figure 2: Word cloud of leave tweets

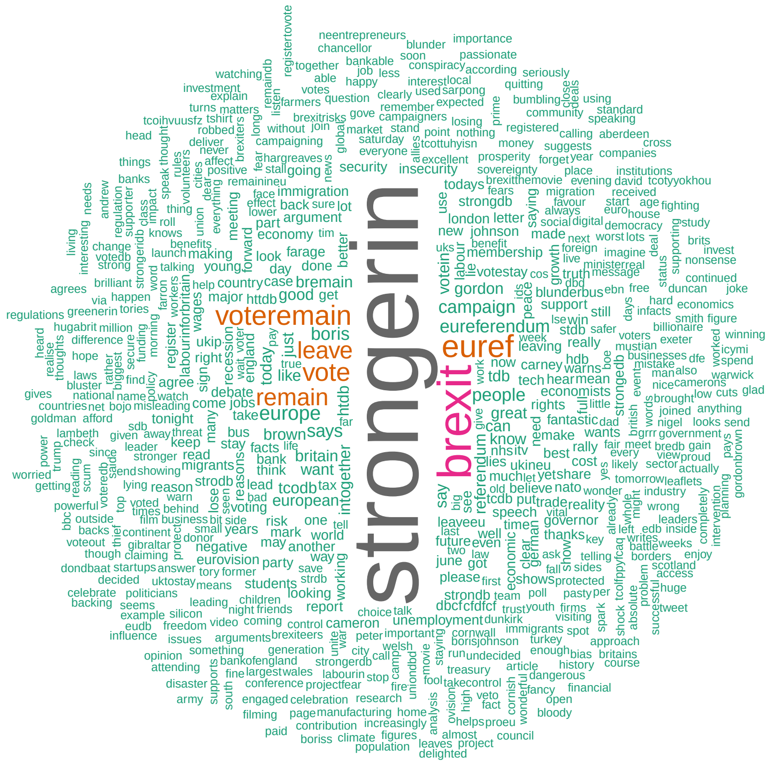


Figure 3: Word cloud of Remain tweets

3.2 Text Analysis

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"
```

3.3 Text Classification

3.3.1 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"
```

3.3.2 Word Association

To find out which words appear together often, I used the buildCorpus function to return a termDocument-Matrix. 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 the word association (term document matrix) against the frequent terms.

3.3.3 Most Frequent Words in Both Leave & Remain Tweets

4 References

[1] P. B. Dongre and L. G. Malik. "A review on real time data stream classification and adapting to various concept drift scenarios". In: *2014 IEEE International Advance Computing Conference (IACC)*. 2014, pp. 533–537. DOI: 10.1109/IAdCC.2014.6779381.