# Coursework
# Module Name (CMM...)

Your Name, *email@rgu.ac.uk*

April 22, 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

### 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]

### 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]

### 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]

### 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]

# 2 Data Streams

The dataset analysed here is the Dota 2 game results dataset, available at
https://archive.ics.uci.edu/ml/datasets/Dota2+Games+Results
The data is taken from the popular online multiplayer game Dota 2. This game is played between two teams of five players each. Each individual player plays as a 'hero' (selected by the player themselves). The two teams then compete to defend their respective bases. The first team to destroy the others base wins.

The dataset was chosen because Dota 2 is widely seen as a challenging problem in the field of AI. Following the success of Alphago at the boardgame go in 2016, Dota 2 is seen by many as the next challenge as no AI has yet been able to beat a leading human team. As with any well designed computer game, balancing is key. In this case, it is a requirement that any given team of heroes should perform equally well as any other team (given players of equal skill). With over 100 different heroes available, each with their own strengths and weaknesses, balancing is a challenging problem for the game design team. If any given set of heros can be shown to have an advantage over another set, this could be used by human, or AI, players to gain an advantage over the other team. This analysis, therefore doesn't seek to predict whether a team will win with a high degree of accuracy (if that were possible, Dota 2 would be a poorly balanced and unsuccessful game); rather we seek to find any marginal advantage beter than random guessing.

With 116 dimensions, the dataset is large enough to demonstrate the potential benefits of streaming, but small enough to classify in a non-streaming manner on a single machine.

## 2.1 Data Exploration

The dataset itself contains data from 102944 individual games. The dependant variable is a factor with levels 1 and -1, with 1 representing a win for the first team and -1 representing a win for the second team. 114 columns each represent an individual hero. If that hero was present in the first team, the respective heros column has a value of '1'; or '-1' if it was present in the second team. A '0' in a heros column means that hero was not present in either of the teams for that game. Other independant variables are present in the form of game type, game mode and geographic data; but these were excluded for the final analysis as this study focuses solely on game balancing through choice of hero.

To study the dataset in more detail, let's load it into the workspace, along with the required libraries. The original dataset, as downloaded, is already split into a training and testing set. However, in order to ensure the split as indeed random and stratified, the data was re-combined into a single dataset.

```
> library(dplyr)
> library(rjson)
> library(caret)
> library(randomForest)
> library(ROCR)
> library(dummies)
> #read in the train and test datasets and combine
>
> df1 <- read.csv('dota_dataset\\dota2Train.csv')
> df2 <- read.csv('dota_dataset\\dota2Test.csv')
> names(df2) <- names(df1)
> #combine the original datasets to a single dataframe
> df <- rbind(df1,df2)
> rm(df1,df2)
```

Looking at the basic structure of the data we note the following (outputs not shown in pdf due to size):

- The typing is incorrect as imported - the types should all be factors
- The dataset is quite sparse

- The columns are not named. They are defined in the readme for the dataset so we will add names later

```
> library(xtable)
> print(xtable(head(df)))
> str(df)
> names(df)
```

Let's also check the size of the dataset:

```
> #record number of rows and columns
> paste('number of columns =' , ncol(df) , '. number of rows =', nrow(df))
```

```
[1] "number of columns = 117 . number of rows = 102942"
```

Now let's check the class distribution (ie do we have an approximately equal number of team 1 wins as team 2 wins?). The distribution is indeed close to equal so we do not need to worry about class imbalance later on. It is interesting to note there is some imbalance, the reason for this is not clear but probably has something to do with the manner in which the data was collected.

```
> #check value distribution of response variable column
> table(df[,1])
```

| | V1 |
|---:|---:|
| -1 | 48658 |
| 1 | 54284 |

An advantage of this dataset is there are no missing values. This is not surprising as the data was captured using the Dota 2 API. Here we also check that each team has an equal number of heroes which is a requirement for a game:

```
> #check for missing values (there are none)
> sum(complete.cases(df))==nrow(df)
```

```
[1] TRUE
```

```
> #check there are equal number of heroes present in each team (hero values for all rows should sum to
> sum((rowSums(df[5:ncol(df)]))) == 0
```

```
[1] TRUE
```

As discussed above, some preprocessing is required. First, we convert all numeric columns to factors:

```
> df[sapply(df, is.numeric)] <- lapply(df[sapply(df, is.numeric)], as.factor)
```

The column names are added in two stages. The first four columns are named manually as specified in teh dataset description.
The remaining columns represent the individual heros. The column names are extracted from the .json file supplied with the dataset:

```
> #add column names to the first few columns
> names(df)[1:4] <- c('won','cluster_id','game_mode','game_type')
> #populate the other column names
> #load in json file and use it to populate column names
> hero_names <- fromJSON(file ="dota_dataset\\heroes.json")
> for ( i in c(1:length(hero_names$heroes))){
+   names(df)[i+4] <- hero_names$heroes[[i]]$name
+ }
```

Here we drop columns that are not related to heros (this analysis is concerned only with hero choice so we drop game type and location related columns). One column has the same value for every entry. This was noticed during the analysis stage. The column represents a mandatory hero which muct be included in the team. Since it conveys no information it is dropped:

```
> #drop unwanted columns
> df <- df%>%
+   select(-c(lina, cluster_id , game_mode , game_type))
```

The good thing about the above approach is you can always refer to the table in your document. For example 'As can be seen in Table **??**,...'

**Figures:** are important part of your analysis, and also good way to give insight about the dataset you are working with. You will use packages like *ggplot2* to produce some visuals. Lets start with a simple example to show how to present your visuals in the report with proper labels and captions. Remember, I need to see the code the produced the figure as well.

Suppose, I just want to create a plot that shows the relation between Petal width and length and map the colour and shape to the Species (class label in my dataset).

First, I will write my code, but notice that my chunk definition is set as follows
*(■warning=FALSE,message=FALSE,eval=FALSE■=)*, I set warning and message to FALSE, because I don't want these warnings/ messages to appear in my final output.

```
> library(ggplot2)
> p <- ggplot(iris, aes(x=Petal.Length,
+                       y = Petal.Width,color=Species) )
> p <- p+ geom_point(aes(shape=Species))
> p <- p + xlab('Petal Length')
> p <- p + ylab('Petal Width')
> p <- p + theme_bw()
> p
>
```

Make sure that your code is running, and once everything is OK, then you need to insert the above code within a Latex code used to insert images (check the .rnw file to see how we achieved this). Again, remember the caption and the label which allows you to refer to this figure from anywhere in your document (Figure 1). Notice the header of the chunk code in the *.rnw* file.

Figure 1: Petal Length /Width per species in IRIS set

## 2.2 Build Classifier

The dataset is now split into training, testing and validation sets. 60% of the data is used for training with 20% used for testing and validation respectivley:

```
> set.seed(57)
> # Split into train test and validation sets
> index <- sample(c(1:3), size = nrow(df), replace = TRUE, prob = c(.6, .2, .2))
> df_train <- df[index == 1,]
> df_test <- df[index == 2,]
> df_valid <- df[index == 3,]
```

A random forest classifier was used to classify the data. This model was chosen for two main reasons:

- With only 3 main tuneable parameters, random forests often require less tuning than other model types.

- For high dimensional data, a SVM would typically perform well, however with over 100000 rows in the dataset, runtime would probably be significant. Random forests typically also perform well on high dimensional data.

Rather than simply training up and testing a single model, some parameter tuning was carried out. The mtry and nodesize parameters were tuned using a tuning grid. Although this gave only 9 different combinations of hyperparameters, it serves as a demonstration of how model selection is achieved.
Here, a parameter grid is defined and stored as a dataframe to be accessed as each candidate model is trained

```
> num_attributes <- ncol(df_train)
> #build a parameter grid
> # Random forest
> mtry <- as.integer(c(num_attributes * 0.25, num_attributes / 3, num_attributes * 0.5, num_attributes *
> nodesize <- c( 3, 5, 20)
> ntree <- c(200)
> PG<- as.data.frame(expand.grid(mtry=mtry, nodesize=nodesize,
+                                           ntree=ntree, stringsAsFactors=F))
```

Now, we train up a random forest on the training set for each combination of parameters in the parameter grid.
For each combination of parameters, the resulting model is tested on the evaluation set and the accuracy calculated. The resulting accuracy from each parameter combination is saved in the 'Accuracy' column of the parameter grid table:

```
> for (i in c(1:nrow(PG))){
+   mtry <- PG[i, "mtry"]
+   nodesize <- PG[i, "nodesize"]
+   ntree <- PG[i, "ntree"]
+
+   temp_model <- randomForest(formula(df_train) , data=df_train, mtry=mtry, ntree=ntree, nodesize=nodes
+   predictions <- predict(temp_model , df_valid)
+   cm<- confusionMatrix(predictions,df_valid$won)
+   accuracy <- cm$overall['Accuracy']
+   PG[i,'Accuracy'] <- accuracy
+   print(i)
+ }
```

The parameters which produced the best model are selected and that model is evaluated on the testing set to give a final accuracy for the random forest classifier.

```
> #assess best model on test set
> PG <- PG%>%
+   arrange(desc(Accuracy))
> mtry <- PG[1, "mtry"]
> nodesize <- PG[1, "nodesize"]
> ntree <- PG[1, "ntree"]
> best_model <- randomForest(formula(df_train) , data=df_train, mtry=mtry, ntree=ntree, nodesize=nodesiz
> predictions <- predict(temp_model , df_valid)
> confusionMatrix(predictions,df_valid$won)
```

## 2.3 Build Stream Classifier

Same as above. Complete this part as required by the coursework sheet. Again, be clear, visuals always helps in communicating results. Justify your choices and explain your methods.

# 3 Text Classification

This section involves analysing a collection of tweets collected during the EU referendum. They are labelled ’leave’ or ’remain’.

## 3.1 Preprocessing

First, the dataset is loaded into the workspace, along with all the required libraries:

```
> library(dplyr)
> library(tm)
> library(SnowballC)
> library(RColorBrewer)
> library(wordcloud)
> library(ggplot2)
> library(gridExtra)
> library(graph)
> library(Rgraphviz)
> library(RTextTools)
> df <- read.csv("leaveRemainTweets_CW.csv")
```

The ’leave’ and ’remain’ tweets are seperated and stored in different objects. We also check that we have captured all tweets by seperating by label (ie there a no unlabelled tweets, or any with labels outwith ’leave’ or ’remain’):

```
> #seperate dataframes for leave and remain tweets
> leave_tweets <- df%>%
+   filter(label=="Leave")
> remain_tweets <- df%>%
+   filter(label=="Remain")
> #check we have captured all the tweets
> nrow(df)==(nrow(leave_tweets) + nrow(remain_tweets))
```

```
[1] TRUE
```

It is important to ensure the dataset is balanced, ie there are a similar number of leave and remain tweets. This is indeed the case:

```
> nrow(leave_tweets)
```

```
[1] 1254
```

```
> nrow(remain_tweets)
```

```
[1] 1029
```

In order to analyse the text, a corpus will be built. This will be achieved through a dedicated function defined below. This function performs the following pre-processing steps:

- convert the encoding to UTF-8
- convert all text to lowercase - the case doesn’t give us any extra information.
- remove punctuation, numbers and URLs
- remove the following common words, none of which convey any real informatiom: ”RT”,”rt”,”EU”,”eu”,”brexit”
- remove words CONTAINING the following terms (these are often used in conjunction with a hashtag): ’vote’, ’leave’, ’remain’, ’stronger’.

- remove all standard English stopwords
- strip out any whitespace.

```
> buildCorpus <- function(someText){
+
+   # build a corpus, and specify the source to be character vectors
+   myCorpus <- Corpus(VectorSource(someText))
+
+   myCorpus <- tm_map(myCorpus,
+                      content_transformer(function(x) iconv(x, to='UTF-8',sub='byte')))
+   myCorpus <- tm_map(myCorpus, content_transformer(tolower))
+   # remove punctuation
+   myCorpus <- tm_map(myCorpus, removePunctuation)
+   # remove numbers
+   myCorpus <- tm_map(myCorpus, removeNumbers)
+   # remove URLs
+   removeURL <- function(x) {
+     gsub("http[[:alnum:]]*", "", x)
+   }
+   myCorpus <- tm_map(myCorpus, removeURL)
+   myCorpus <- tm_map(myCorpus, content_transformer(removeURL)) #??
+   # add stopwords
+   myStopwords <- c(stopwords("english"), "RT","rt","EU","eu","brexit")
+   #remove words CONTAINING certain terms
+   myCorpus <- tm_map(myCorpus, content_transformer(gsub), pattern = "*vote*|*leave*|*remain*|*stronger
+   # remove stopwords from corpus
+   myCorpus <- tm_map(myCorpus, removeWords, myStopwords)
+   #
+   myCorpus <- tm_map(myCorpus, stripWhitespace)
+   # Return the text corpus
+   return(myCorpus)
+ }
```

In order to explore the tweets, we will construct a document-term matrix. This will be achieved with the following function. The function returns several useful objects:

- the document-term matrix
- the term-document matrix
- a frequency table for individual words (used to make wordclouds)
- the text corpus, as produced by the 'buildCorpus' function

```
> #function to build term document matrix, along with other useful objects for analysis
> make_tdm <- function(text_column,minfreq=1){
+   text_column <- iconv(text_column, 'UTF-8', 'ASCII')
+   corpus <- buildCorpus(text_column)
+   # keep for later
+   corpus_copy <- corpus
+   # stem words
+   corpus <- tm_map(corpus, stemDocument)
+   tdm <- TermDocumentMatrix(corpus,control=list(bounds = list(global = c(minfreq,Inf))))
+   dtm <- DocumentTermMatrix(corpus,control=list(bounds = list(global = c(minfreq,Inf))))
+   m <- as.matrix(tdm)
+   v <- sort(rowSums(m), decreasing = TRUE)
+   d <- data.frame(word = names(v), freq = v)
+   return(list(corpus=corpus,freq_table=d,tdm=tdm , dtm=dtm))
+ }
```

As the 'maketdm' function outputs a word frequency table, we can use it to produce word clouds for the leave and remain tweets. To avoid repeating tasks, the process for making and plotting wordclouds is defined in dedicated functions.

```
> #make corpuses and tdms from the tweets
> all_tweets <- make_tdm(df$text)
> remain <- make_tdm(remain_tweets$text)
> leave <- make_tdm(leave_tweets$text)
> #function to make wordclouds
> make_wordcloud <- function(freq_tab){
+ wordcloud(words = freq_tab$word, freq = freq_tab$freq, min.freq = 5,max.words=2000, random.order=FALSE

> #make wordclouds from the frequency tables
> make_wordcloud(remain$freq_tab)
```

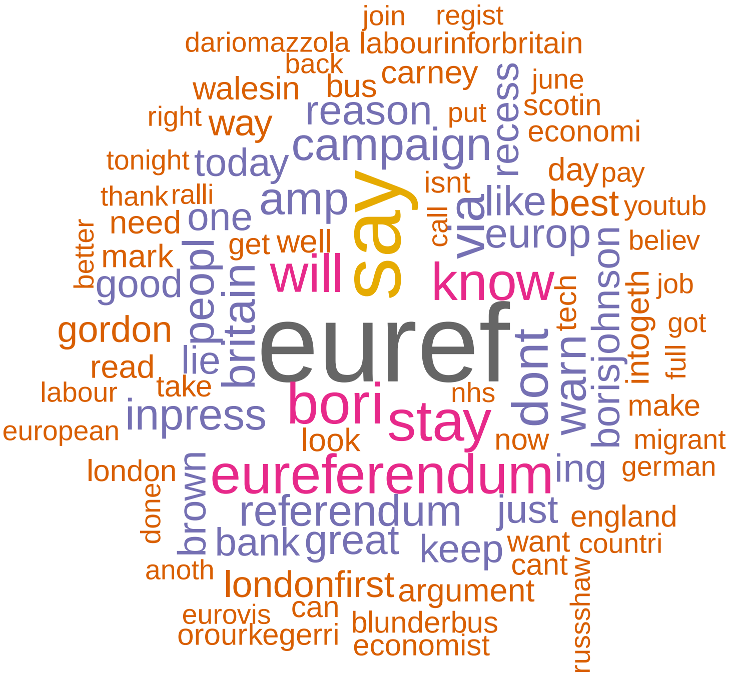

Figure 2: Wordclouds for the remain tweets

```
> make_wordcloud(leave$freq_tab)
```

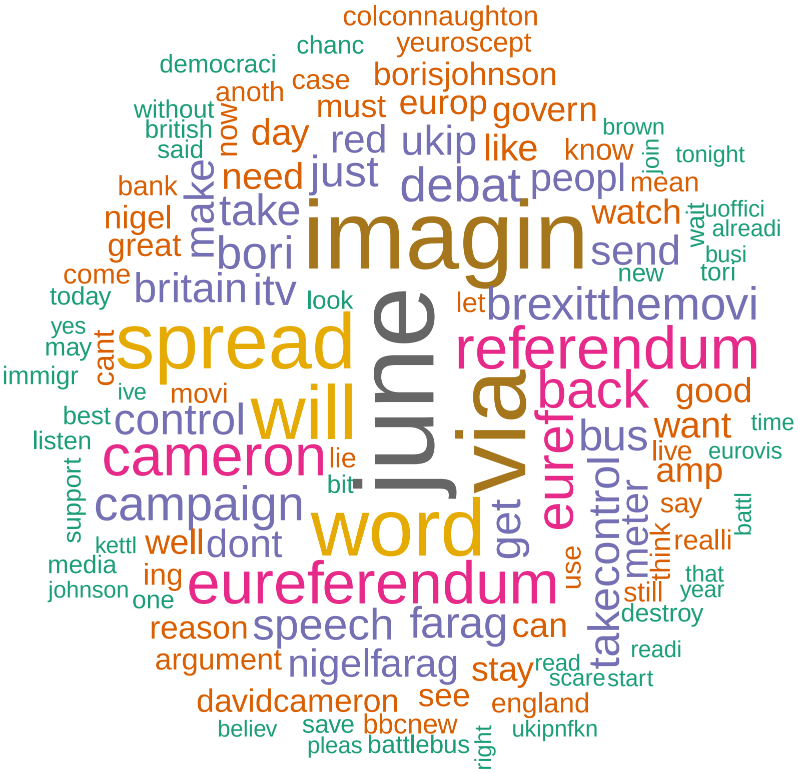

Figure 3: Wordclouds for the leave tweets

In addition to wordclouds, we can plot the most common words for each class of tweet. This is achieve with the function below:

```
> #function to plot wordclouds
> plot_wordcount <- function(freq_tab,title_text){
+   ggplot(head(freq_tab,20), aes(x=word,y=freq))+
+     geom_bar(stat="identity")+
+     labs(title=title_text)+
+     theme(axis.text.x = element_text(angle = 30, hjust = 1))
+ }
> a<-plot_wordcount(all_tweets$freq_table,"most common words for all tweets")
> b<-plot_wordcount(remain$freq_table,"most common words for remain tweets")
> c<-plot_wordcount(leave$freq_table,"most common words for leave tweets")
>
>
>
```

```
> grid.arrange(a,b,c, nrow = 3)
```

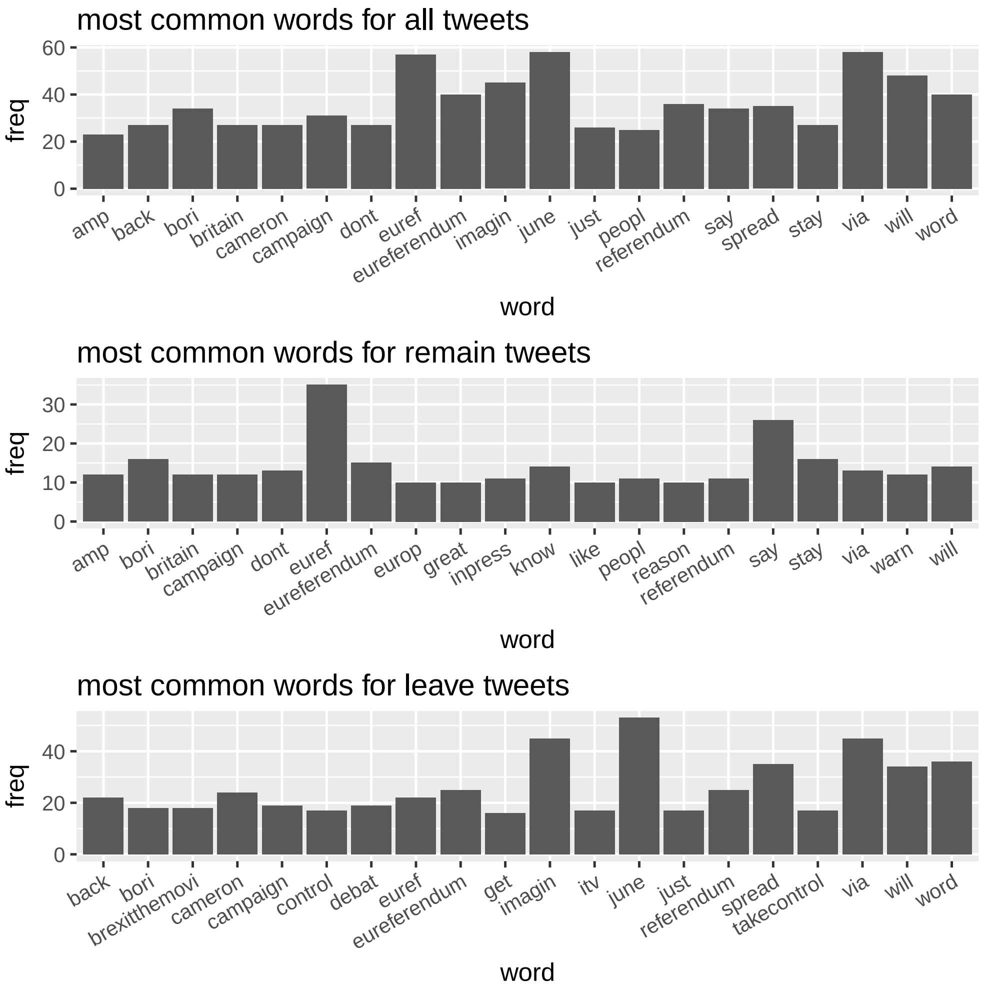

Figure 4: Wordcounts for the tweets

### 3.2 Text Analysis

Record, explain and justify your appraoch.Complete this part as required by the coursework sheet. Again, be clear, visuals always helps in communicating results. Justify your choices and explain your methods.

### 3.3 Text Classification

## 4 Reproducing Results

You don't need to have a section called **Reproducing Results**, your report itself will be the answer for this section.

## 5 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.