TwitterDMTools

Jakub Kryczka

March 2019

Short paragraph introducing the package and what it is used for. (will be written when i think of something good)

short aside that it is required to configure the twitter api before use: see link for how to do it: `https://rtweet.info`

1.0 Setting up

Short introduction about collecting tweets, it is possible to do it in the past but this package focuses on collecting live data and analysing it, for more info regarding streaming past tweets see: `?search_tweets()`

Provide examples of `streamTweets()` and its various inputs, provide an example, talk about local directory and making sure its saved, but you can also specifiy file path, for more info see `?streamTweets()`

Start by downloading the `TwitterDMTools` package from Github, for this to work make sure to have the `devtools` package download

```r
#load in devtools package
library(devtools)

#install TwitterDMTools package from Github repository
install_github("jakubk28/TwitterDMTools")

#load in TwitterDMTools package
library(TwitterDMTools)
```

Once the package has been downlaoded and has loaded in successfully, ensure the rtweet package credentials have been set up correctly for the Twitter API to work.

Finally, the last step required is to set the working directory of the R script file using the function `setwd` or manually within RStudio in the `Sessions` tab. The process of streaming and loading in tweets requies saving and loading in a local file, hence RStudio needs to know where to save the data file and where to load it in from.

```r
#set working directory for saving and importing files
setwd("~/")
```

1.1 Listener

To begin collecting Twitter data, use the `streamTweets()` function by inputting desired words or phrases to be listened for, and the desired amount of time to do this for. Search time is very relative to the popularity of the words and phrases serached for. Highly discuessed terms such as `Trump` will result in thousands of Tweets collected within a matter of a minute, whilst more niche topics such as `plastic pollution` will take significantly longer to give results.

The `streamTweets()` function can also search for multiple terms simultaneously by inputting `"trump,wall"`, this input will return any tweets that mention the words “trump” or “wall” or both. This approach can result in tweets including the word “wall” to be caught in the stream that are unrelated to our analysis, whilst

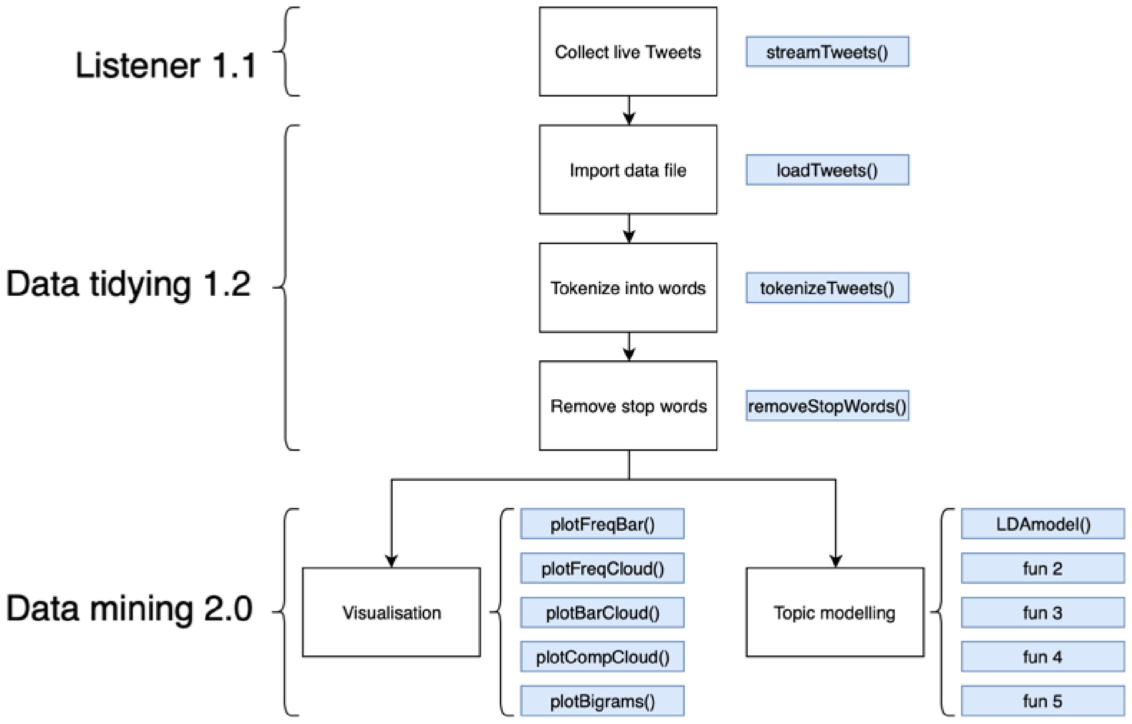


Figure 1: Flow Chart outlining sections of the package and the functions involved.

there are methods to cluster these unwanted tweets using topic modelling, it is important to keep in mind that words can have numereous meanings and connotations.

The final input for the function is the `filename`, this is what the file will be saved as in the chosen working directory, it is crucial to include the `.json` file extension to ensure the data is saved as a 'JavaScript Object Notation' file.

For simplicity and reproducibility we will serach for any tweets containing the phrase `trump` for `60` seconds and save the the streamed tweets in a file named `trumpTweets.json`.

```
#search for tweets mentioning "trump" for 60 seconds
streamTweets("trump",60,"~/Desktop/trumpTweets.json")
```

1.2 Data tidying

Once a json data file containing all of the tweets has been obtained, we are ready to import the file into RStudio and start tidying the data in preparation for mining. The function `loadTweets()` can be used to do just that. As before, the file path either needs to be specified in the function input or alternatively the working directory can be set within RStudio and the json file should be located in the same directory.

```
#load in json data file and save as a data frame object
tweetsDF <- loadTweets("~/Desktop/trumpTweets.json")
```

Whilst the data has been imported to an R friendnly format, the format of the actual data frame is not very useful for data mining purposes. Although it is important to mention that the raw data frame just created should be kept for later use, as it will become handy later on to see the full tweets.

The first step in tidying the data is to tokenize the text. Tokenization is the process of splitting up a string of text into individual,pairs, or triplets of words. This can easily be achieved using the `tokenizeTweets()` function. The function takes in a dataframe object as an input, specifically the output from the `loadTweets()` function. The other necessary input is the amount of words the tweets should be tokeized into, the function currently supports `n=1,2,3` as greater tokenization quantaties are effective for the twitter text as the length of tweets is extremely small in the first place.

```
#tokenize tweets into singular words
tweetsDFtoken1<-tokenizeTweets(tweetsDF,1)

#view the top entries in the tokenized data frame
head(tweetsDFtoken1)
#>      word
#> 1       i
#> 1.1 think
#> 1.2 about
#> 1.3   how
#> 1.4  that
#> 1.5   one
```

Similar results can be obtained by using a tokenization value of 2 or 3.

```
#pairs of words
#tokenize tweets into pairs of words
tweetsDFtoken2<-tokenizeTweets(tweetsDF,2)

#view the top entries in the pair tokenized data frame
head(tweetsDFtoken2)
#>     word1  word2
#> 1       i  think
#> 1.1 think  about
#> 1.2 about    how
#> 1.3   how   that
#> 1.4  that    one
#> 1.5   one solemn

#triples of words
#tokenize tweets into pairs of words
tweetsDFtoken3<-tokenizeTweets(tweetsDF,3)

#view the top entries in the triples tokenized data frame
head(tweetsDFtoken3)
#>     word1  word2   word3
#> 1       i  think   about
#> 1.1 think  about     how
#> 1.2 about    how    that
#> 1.3   how   that     one
#> 1.4  that    one  solemn
#> 1.5   one solemn awkward
```

The dataframes are now in better format, containing only the most important parts of the tweets: the words tweeted. Whilst it would be possible to perform analysis on this data, the results would not show much, lets check this theory using the `tokenFreq` function. The function takes a tokenized data frame as an input and returns the most frequently occuring word or pair/triple of words depending on the tokenization amount.

```
#display most frequent words
head(tokenFreq(tweetsDFtoken1))
```

```
#> # A tibble: 6 x 2
#>   word      n
#>   <chr> <int>
#> 1 the    1512
#> 2 trump  1197
#> 3 to     1113
#> 4 t.co   1014
#> 5 https  1012
#> 6 a       804
```

It is evident that these highly occuring words have no real meaning and are useless for the purposes of analysis. These words are called stop words. Removing stop words will create unreadable tweets, which is why it is important to keep the original raw tweet file for later cross referencing of certain words.

Stop words can easily be removed using the `removeStopWords()` function. The function takes an input of a tokenized data frame (the output of `tokenizeTweets()`).

```
#remove stop words from the dataframe
Token1<-removeStopWords(tweetsDFtoken1)

#most frequent words after removal of basic stop words
head(tokenFreq(Token1))
#> # A tibble: 6 x 2
#>   word          n
#>   <chr>     <int>
#> 1 trump      1197
#> 2 president   192
#> 3 amp         191
#> 4 donald       98
#> 5 melania      97
#> 6 people       97
```

Whilst the default stop words dictionary has removed the majority of unnecessary words for analysis, some still remain. We will start by understanding the context of the word "amp" and assess whether it should remain in the data set, to do this the `tweetContext()` function can be used. The function takes an input of the desired word to be contextualised and the original raw data frame.

```
#find the context of the word "amp"
SpecificWordContext<-tweetContext(tweetsDF,"amp")
SpecificWordContext
#> # A tibble: 177 x 1
#>    text
#>    <chr>
#>  1 I think about how that one, solemn, awkward yelp from Howard Dean derai~
#>  2 "Why is @realDonaldTrump pictured w/ the founder of a spa busted in a h~
#>  3 This strongly implies the alleged leader of a sex slavery ring was also~
#>  4 "FTA: \"Donald Trump now has a competent and unhampered attorney genera~
#>  5 "@FreeLion7 @pricklypear12 @tgradous @ChuckAnother @larryvance47 @pjame~
#>  6 @thehill Thank you Senator. That is something #trump lies about every s~
#>  7 Hope you'll join me with @kendisgibson on #MSNBC at 3 CT/4 ET re #ErikP~
#>  8 "@KlasicalLiberal Trump's center-left. Tariffs, gun bans, Executive pow~
#>  9 This strongly implies the alleged leader of a sex slavery ring was also~
#> 10 Despite the most hostile and corrupt media in the history of American p~
#> # ... with 167 more rows
```

From the results, it is possible to gather that the word in question is infact a technical term that appears on tweets when users share and quote other resources; as a result, it can now be excluded from the analysis be running the `removeStopWords()` function again and adding additional terms to be removed. The term "trump" has also been removed as it was the search ter, thus it is evident it it will be highly occuring whilst providing no further insight into the analysis.

```
#remove stop words from the dataframe
Token1<-removeStopWords(tweetsDFtoken1,c("trump","amp"))

#most frequent words after removal of basic stop words
head(tokenFreq(Token1))
#> # A tibble: 6 x 2
#>   word          n
#>   <chr>     <int>
#> 1 president   192
#> 2 donald       98
#> 3 melania      97
#> 4 people       97
#> 5 obama        92
#> 6 wall         89
```

This process can be repeated numerous times, adding additional terms each time until a data set that is suitable is reached.

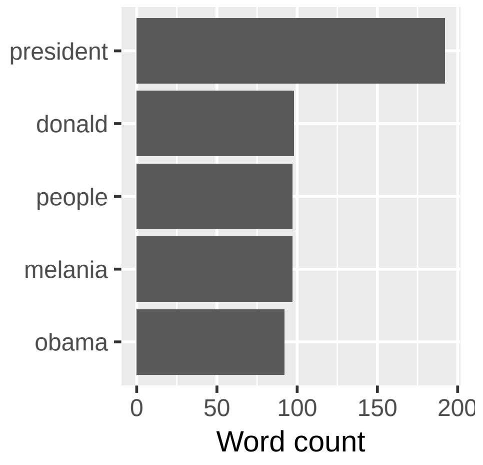
2.0 Data mining

The data mining process can be split up into 2 activities: -Visualisation (basic frequency and sentiment analysis) -Topic modelling

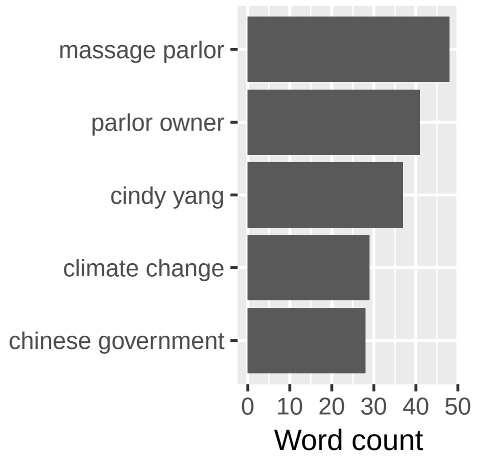
Visualisation

The most basic visual plot can be achieved using the `plotFreqBar()` function, the function takes an input of a tokenized data frame, preferablly one that has been tidied but as mentioned previously it does not matter. The function also has an additional input of how many of the top words to display (`default = 10`).

```
#plot frequency bar chart of most frequently occuring words
plotFreqBar(Token1,5)
```

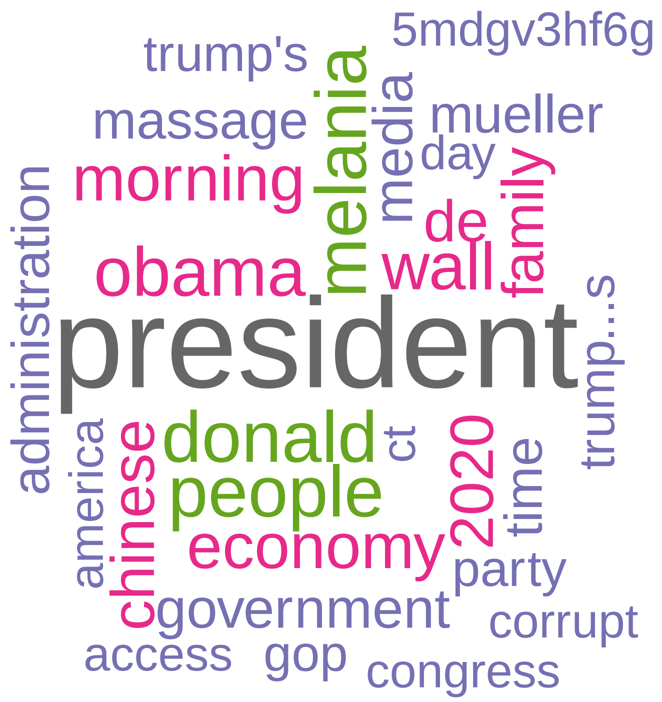


```
#plot frequency bar chart of most frequently occuring pairs of words
plotFreqBar(Token2,5)
```



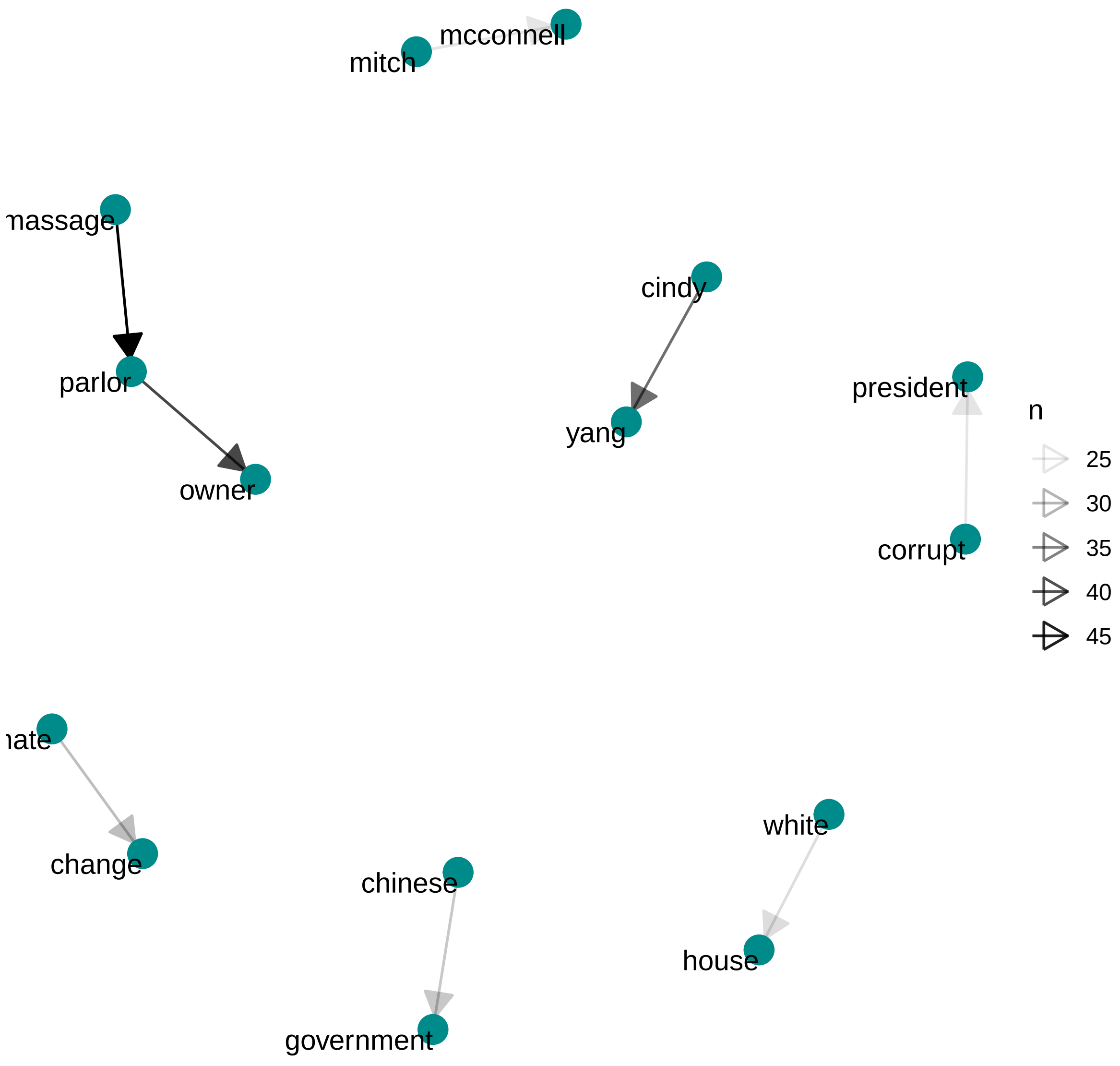
Another way to visualising word frequency is word cloud plots. Word clouds display words with varying sizes depending on frequency, this can easily be achieved within the package by using the `plotFreqCloud()` function. The function takes the same data input as the `plotFreqBar()` function (a tokenized dataframe), however, word cloud plots can only be generated for tokenizations of 1, so pair or triples of words will not work. One additional parameter associated with the `plotFreqCloud()` function is the `minWordFreq` variable, this indicates the minimum frequency a word must have for it to be plotted on the cloud plot.

```r
#plot frequency cloud of at most 30 words, occuring at least 2 times
plotFreqCloud(Token1,maxWords = 30,minWordFreq = 2)
```



The final basic visualisation function is `plotBigrams()`;the function produces a term network displaying how the most frequent terms occur within the text. The function takes in an input of a data frame tokenized specifically into pairs of words (`n=2`), as well as an optional parameter `minWordFreq` specifying the minimum frequency of a word to appear on the network plot.

```r
#plot term network using bigrams (tokenization of 2)
plotBigrams(Token2,minWordFreq = 25)
```



Sentiment analysis

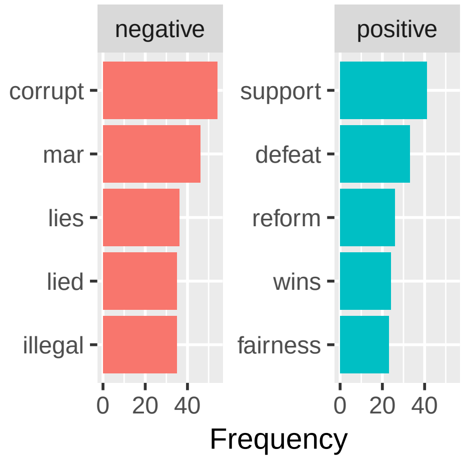
Basic visualisation functions like `plotFreqBar()` and `plotFreqCloud()` allow us to see the most frequently occuring meaningful words, this however does not tell the full story of the context behind the tweets. As a result, we can use sentiment analysis tools to attempt to extract further knowledge from the corpus.

Sentiment anaylsis is the process of extracting emotional context from a corpus, in this case, the emotions felt by writers within their tweets. The concept behind sentiment analysis is that most words have either a positive or negative conotation; by comparing the frequencies of these words, either locally within a tweet or globally within the corpus, it is possible to analyse the overall sentiment felt by the auther or authers respecitvely.

The two sentiment analysis `plotCompBar()` and `plotCompCloud()` operate by comparing a sentiment lexicon (a list of words with a corresponding binary term determining whether the word is positive or negative), this then gives each word in our data set a positive or negative value which can then be plotted. In the area of natural language processing, there numerous sentiment lexicons (bing, AFINN, nrc) each with different purposes; the functions in the package use the 'bing' lexicon, which uses a binary approach of either positive or negative. Conversely, the 'AFINN' lexicon expands on the 'bing' lexicon by having a scale of how positive and negative the word is from 1-5; for instance, the world 'like' has a value of 2 whilst the word 'love' has a value of 3, displaying a stronger positivity. (aside for JJ:i will put this in the report instead as its unrelated to the guide)

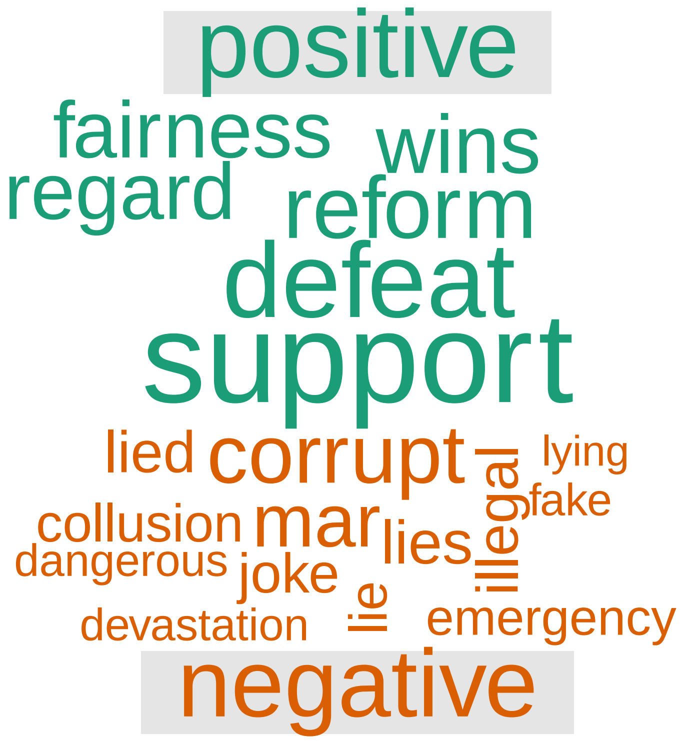
Similarly to the `PlotFreqBar()` function, the `plotCompBar()` function takes in a single word tokenized data frame, with optional parameters including minimum frequency for a word to be displayed, and maximum number of words displayed, essentially giving the plot upper and lower bounds.

```r
#comparison bar chart of most used positive and negative words
plotCompBar(Token1,maxWords = 5,minWordFreq = 2)
```



The function produces two frequency plots, displaying the top `n` highest frequency positive and negative words within the data set. As before, it is also to plot this result in a word cloud format using the `plotCompCloud()` function. As with the previous cloud plot function, the function takes an input of a single word tokenized data frame, with optional parameters dictating minimum word frequency and maximum number of words to appear.

```r
#comparison bar chart of most used positive and negative words
plotCompCloud(Token1,maxWords = 20,minWordFreq = 5)
```



It is important to understand that format of the cloud plot means that the size of each word represents the frequency of that word. However, in this case the size is respective to the proportion of that word within the positive or negative classes, not the entire corpus, which is why certain positive words might appear larger in the plot and mislead the readed; consequently, it is important to use both functions `plotCompBar()` and `plotCompCloud()` to gauge the sentiment of the corpus.

Topic modelling

Will be added once I convert into functions and update package.

footnotes[1]

[1] Ain case i need format for footnote.