TwitterDMTools

Jakub Kryczka

March 2019

`TwitterDMTools` is a suite of functions that are used to collect, analyse, and visualise live tweets. The package aims to streamline the process of gathering tweet data and extracting relevant information from it.

1 Setting up

To begin the data mining process it is essential to obtain, and set up the requried tools to perform the tasks. The easiest way to install the `TwitterDMTools` package is through the use of the `devtools` package, in particular the `install_github()` function. By providing the function with an appropriate link to a Github repository containing an R package, it will download and install everything necessary.

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

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

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

Alternatively, the `TwitterDMTools` package can be installed by downloading the package contents from the Github repository `https://github.com/jakubk28/TwitterDMTools` and using the `install.packages()` function to install the package from a local folder.

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

#install TwitterDMTools package from Github repository
install.packages("TwitterDMTools")

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

The process of collecting tweets involves the use of Twitter's API, a set of protocols that creates a gateway between an application and the Twitter server. In order to interact with this API and begin the tweet collection process it is necessary to set up a Twitter developer account `https://developer.twitter.com/en/apply-for-access.html`. Once an account has been successfully authorized, simply set up a project on the twitter website and link the credentials provided between the account and the third party application. For more information and more detailed steps regarding the autherization and linking process see `https://rtweet.info`.

2 Listener

To commence the collection of Twitter data, use the `streamTweets()` function by inputting desired words or phrases to be listened out for, and the desired amount of time for this process (seconds). It is imporatnt to consider that highly discuessed terms such as `Trump` will result in thousands of tweets being collected within a matter of minutes, whilst more niche topics, such as `plastic pollution`, will take significantly longer to

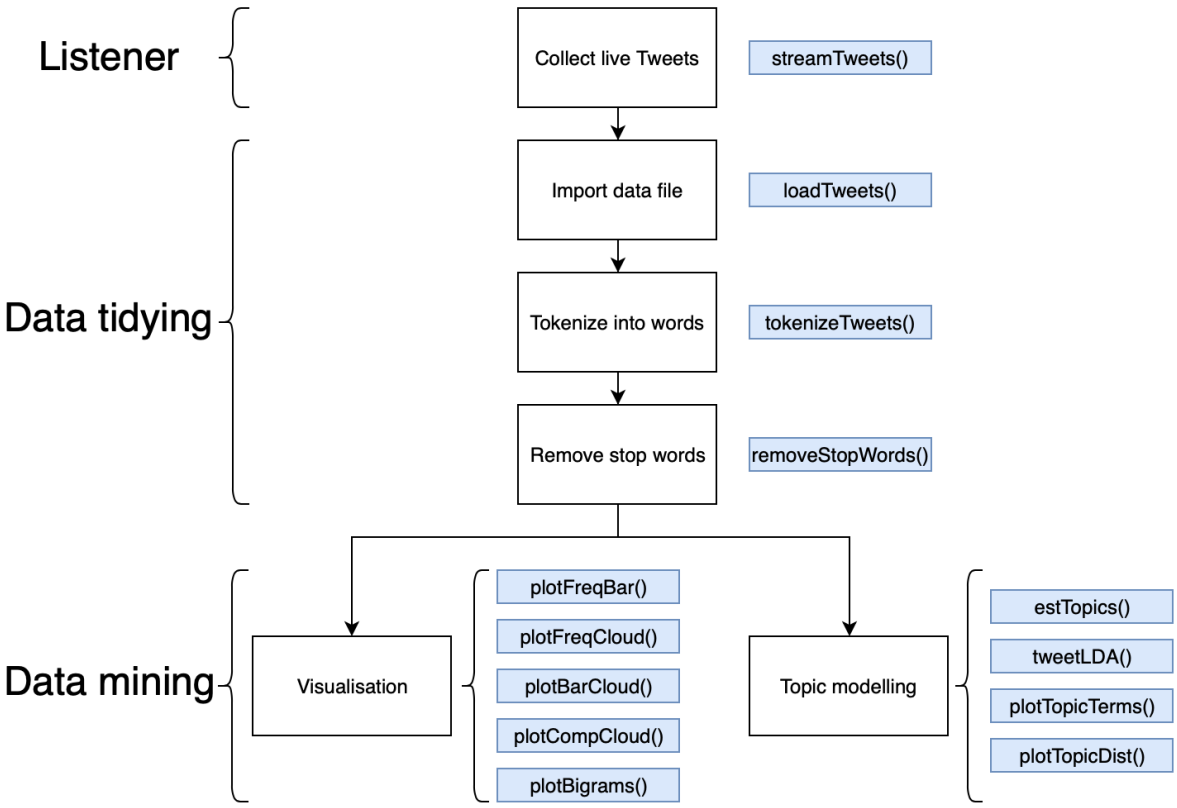


Figure 1: Flow Chart outlining sections of the package and the functions involved.

gather results. Thus, if the object is to find less spoken about terms, the search period will have to be much longer to provide sufficient data for effective mining results.

The `streamTweets()` function can also search for multiple terms simultaneously. The input argument equates `commas` and `spaces` to the Boolean opeartors `OR` and `AND` respectively. For example, inputing the search query "polar,bear" into the `streamTweets()` function will result in any tweets including the word polar OR bear, whilst the query `polar bear` would result in tweets containg both the words polar AND bear. This concept is important to remember when applying the function to multiple terms to ensure the correct tweets are being listened to and collected.

Finally, the output file name can be specified within the function to ensure unique files are being generated and that no data is overwritten and lost.

```
#search for tweets mentioning "plastic AND pollution" tweets for 60 seconds
streamTweets("plastic pollution",60,"conservationTweets")
```

3 Data tidying

Once a `.json` data file containing all of the tweets has been obtained, the data is ready to be imported in through the use of the `loadTweets()` function.

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

The `loadTweets()` function will return a data frame object that contains all information regarding every tweet that has been collected.

For reproducability, the data collection has been contucted before hand, using the search query `"palm oil,plastic pollution"`, the final data file called `ConservationTweets` is included with the package as a guide for the data mining process.

```
#load in included tweet data file
tweetsDF <- ConservationTweets
```

Whilst dataframes are very good for data mining purposes, the format of the actual data frame returned by the twitter API is not very useful for data mining. 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 full, unfiltered tweets.

3.1 Tokenization

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 n sets of words. This can easily be achieved using the `tokenizeTweets()` function. The function accepts a dataframe object as an input, specifically the output from the `loadTweets()` function. Whilst the `tokenizeTweets()` function does support tokenizations of $n > 0$, any $n > 2$ is not recommended for effective twitter data mining as the length of tweets is extremely small resulting in high tokenizations returning whole tweets, contradicting our desired approach of splitting texts up into tokens that have individual meanings.

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

#view the top entries in the tokenized data frame
head(tweetsDFtoken1)
#>     document      word
#> 1          1       dog
#> 1.1        1    owners
```

```
#> 1.2        1    warned
#> 1.3        1        of
#> 1.4        1 dangerous
#> 1.5        1      palm
```

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)
#>   document     word1     word2
#> 1        1       dog    owners
#> 2        1    owners    warned
#> 3        1    warned        of
#> 4        1        of dangerous
#> 5        1 dangerous      palm
#> 6        1      palm       oil

#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)
#>   document     word1     word2     word3
#> 1        1       dog    owners    warned
#> 2        1    owners    warned        of
#> 3        1    warned        of dangerous
#> 4        1        of dangerous      palm
#> 5        1 dangerous      palm       oil
#> 6        1      palm       oil    threat
```

The data has been parsed, 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 oil    6241
#> 2 palm   5997
#> 3 the    5071
#> 4 to     3659
#> 5 and    2564
#> 6 of     2562
```

It is evident that these highly occuring words have no real meaning and are useless for the purposes of analysis. These terms 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.

3.2 Stop 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 oil          6241
#> 2 palm         5997
#> 3 sustainable   700
#> 4 sign          403
#> 5 malaysia      394
#> 6 products      370
```

Whilst the default stop words dictionary conducts a good, basic level clean of unnecessary words within the data set, some still remain, along with other niche words that require manual checking and removal. To understand the context of some more uncommon words, a good approach is to use the `tweetContext()` function. The function takes an input of the desired word to be contextualised and the original raw data frame. In this example we will investigate the popularity of the term `chester` and assess whether it beneficial to the analysis.

```
#find the context of the word "chester"
SpecificWordContext<-tweetContext(tweetsDF,"chester")
SpecificWordContext
#> # A tibble: 81 x 1
#>    text
#>    <chr>
#>  1 "Yes Chester!!! Proud to call the world's first sustainable palm oil ci~
#>  2 @e_wright_teach @anfieldcarol @IcelandFoods @BBCNewsround @FerreroUK @C~
#>  3 @chesterzoo But what did we use before palm oil? Do we really need to u~
#>  4 Fantastic news! Chester declared first Sustainable Palm Oil City to pro~
#>  5 "Great news - following a campaign by @chesterzoo, #Chester has been na~
#>  6 @chesterzoo I hope there is no Palm Oil in valve oil!
#>  7 "Great news for the city and @chesterzoo - #Chester is officially the W~
#>  8 "Great news for the city and @chesterzoo - #Chester is officially the W~
#>  9 "Chester continues to be a trend setter. We are very proud to be part o~
#> 10 @chesterzoo @PHS_Geography our next mission; becoming palm oil sustaina~
#> # ... with 71 more rows
```

From the context it is clear to see that the city of Chester is the first sustainable palm oil city and there is a lot of news about it, concluding that it is important to keep in the analysis. The same approach can be repeated for other terms.

As the terms `palm oil` were included in the search query, it is sensible to assume that the terms will not provide any extra knowledge to the analysis as most tweets will contain the terms. As a result, it is appropriate to remove the terms `palm`,`oil`,and `palmoil` from the data set to give room for other terms to shine and not be overpowered by the search term.

```
#remove stop words from the dataframe
Token1<-removeStopWords(tweetsDFtoken1,c("palm","oil","palmoil"))
```

```
#most frequent words after removal of basic stop words
head(tokenFreq(Token1))
#> # A tibble: 6 x 2
#>   word            n
#>   <chr>       <int>
#> 1 sustainable   700
#> 2 sign          403
#> 3 malaysia      394
#> 4 products      370
#> 5 petition      365
#> 6 lots          353
```

This process can be repeated numerous times, adding additional terms each time until a data set that is suitable is reached. The data is now in a suitable format to begin the data mining process.

4 Data mining

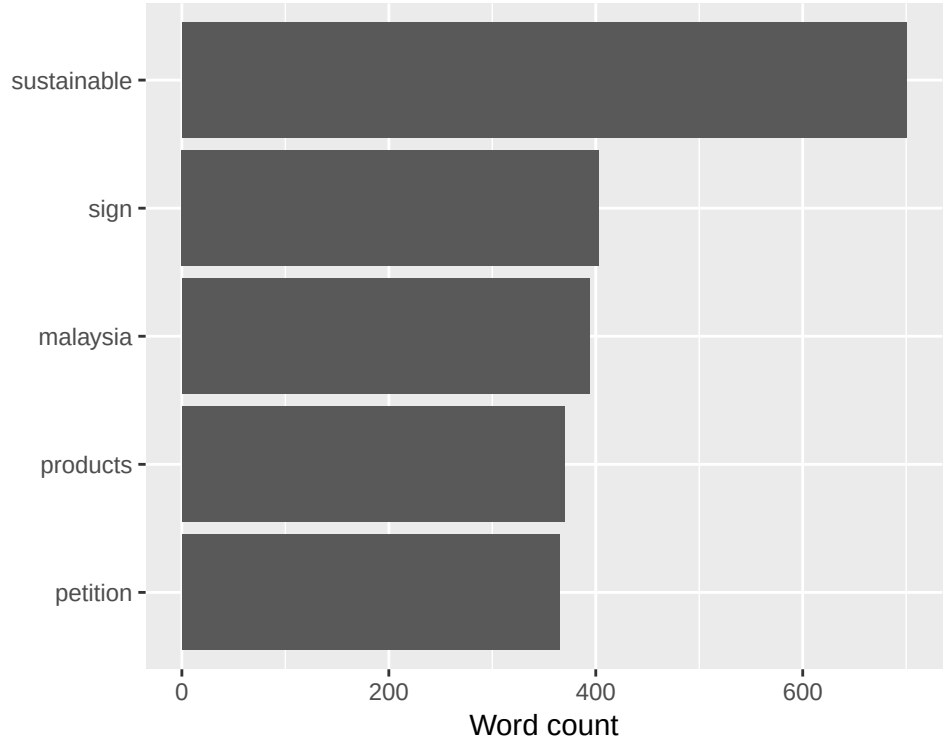
The data mining process can be split up into 2 activities: *Visualisation (Frequency plots, word clouds, and sentiment analysis)* Topic modelling

4.1 Visualisation

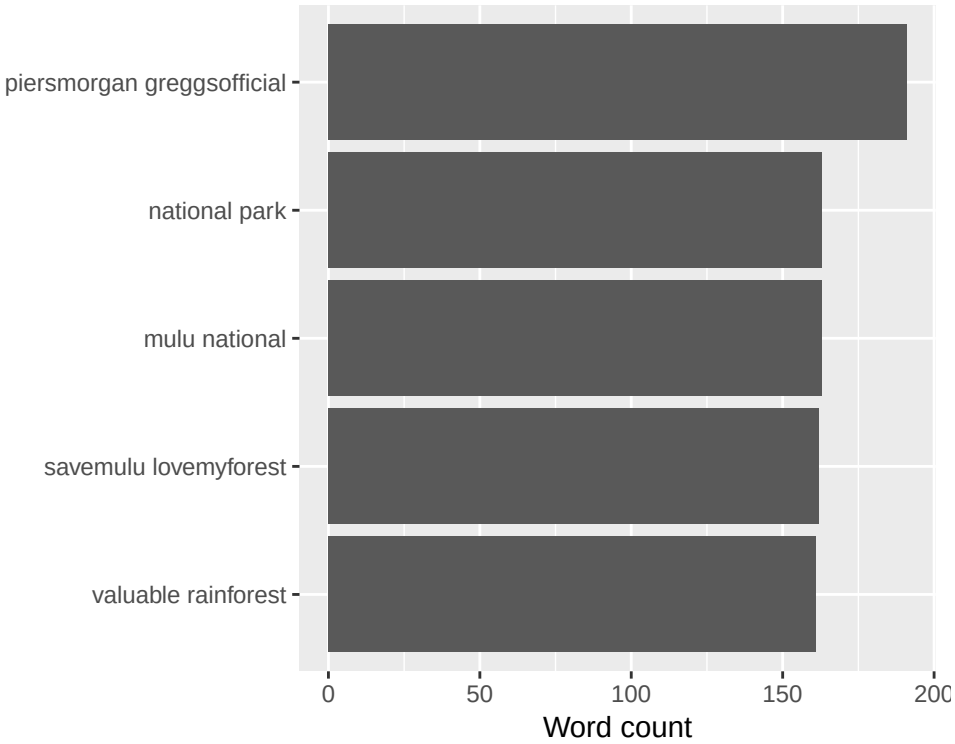
4.1.1 Frequency plots

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)
```



4.1.2 Frequency clouds

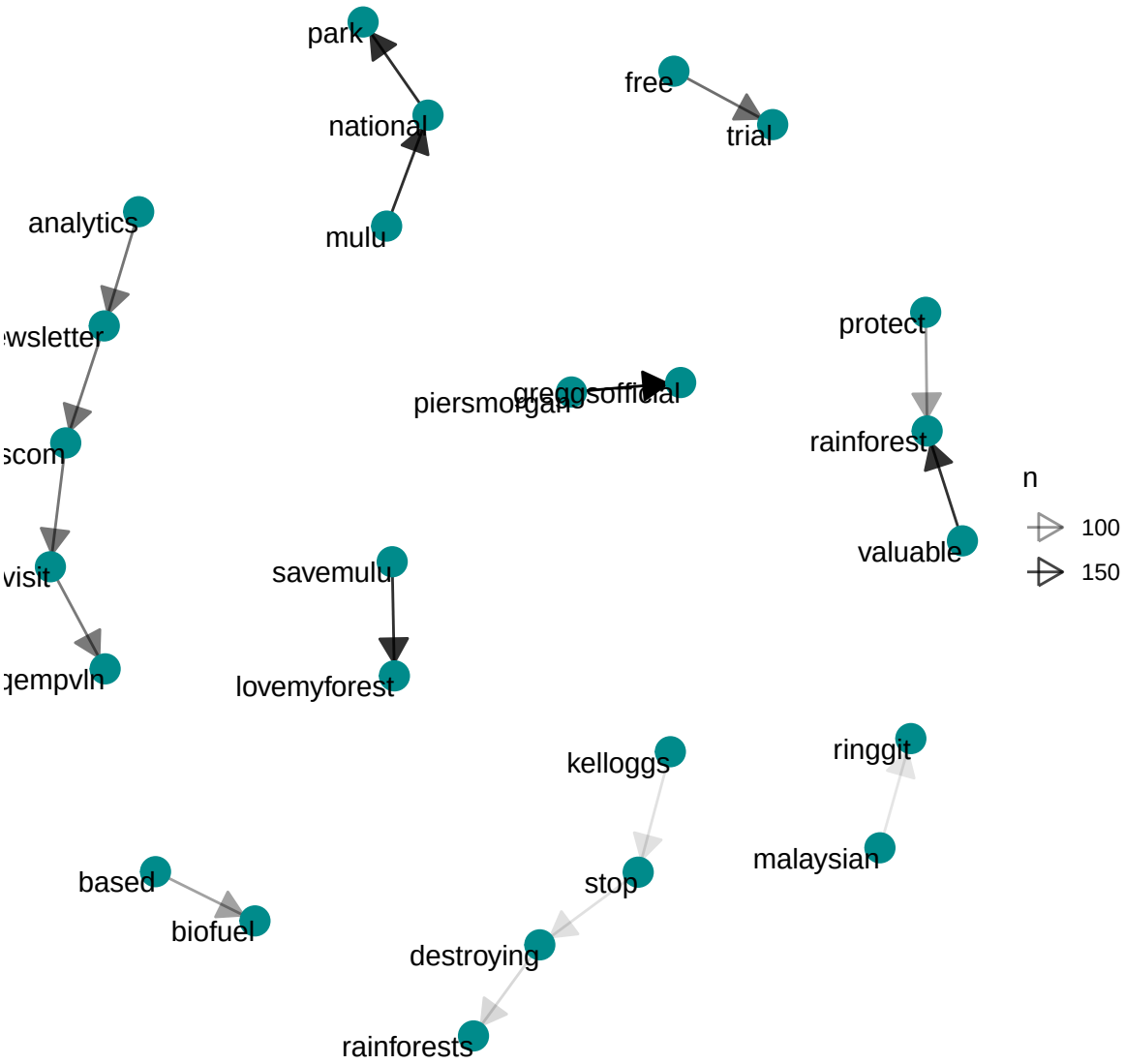
Another way to visualising word frequency is word cloud plots. Word clouds display words with varying sizes depending on frequency, this can easily be achieved within the package by using the `plotFreqCloud()` function. The function takes the same data input as the `plotFreqBar()` function (a tokenized 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.

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

4.1..3 Term networks

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.

```
#plot term network using bigrams (tokenization of 2)
plotBigrams(Token2,minWordFreq = 50)
```



4.2 Sentiment analysis

Basic visualisation functions like `plotFreqBar()` and `plotFreqCloud()` allow us to see the most frequently occuring meaningful words, this however does not tell us 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.

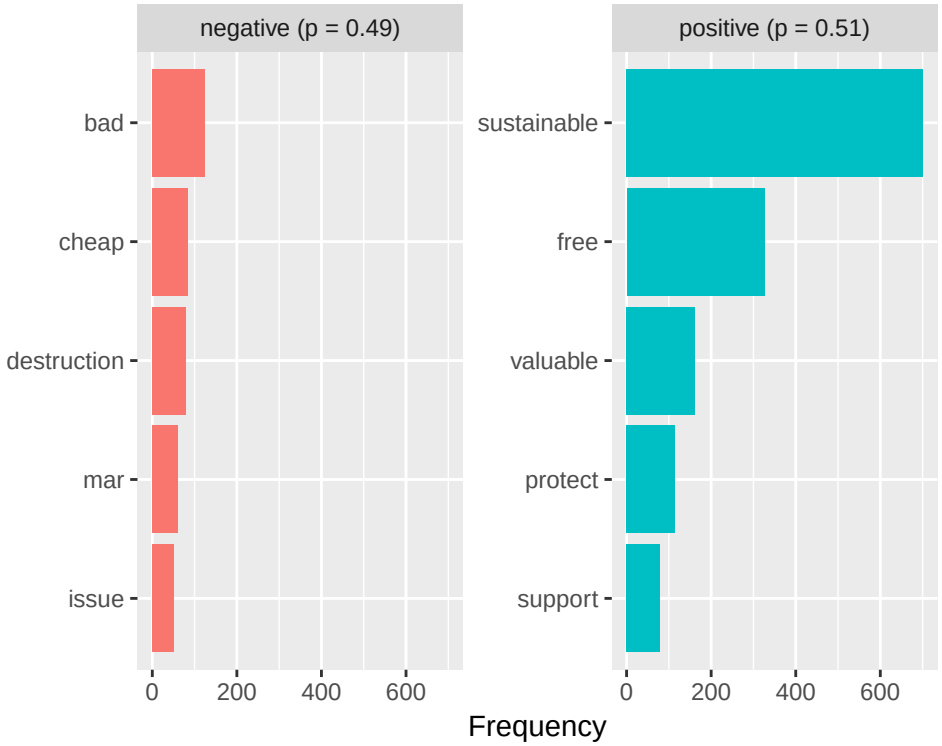
Sentiment anaylsis is the process of extracting emotional context from a corpus, in this case, the emotions felt by writers within their tweets. The concept behind sentiment analysis is that most words have either a positive or negative conotation; by comparing the frequencies of these words, either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

The two sentiment analysis functions `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) with our tweets, this then gives each word in our data set a positive or negative value which can then be plotted.

4.2.1 Sentiment frequency plots

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.

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



The plot gives an indication of the most prominent positively and negatively used words, but also gives a ratio of the frequency of positive and negatives words used within the data set (51% and 49% respectively).

4.2.2 Sentiment clouds

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.

```
#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.

4.3 Topic modelling

In text mining it is often interesting and worthwhile to investigate and identify distinct topics being discussed within the text corpus. Topic modelling achieves this by performing unsupervised clustering algorithms on a corpus that consists of numerous documents in order to identify natural groups of words that fit together to form topics.

Whilst there are numerous algorithms within the topic modelling scene, this package will focus on using the Latent Dirichlet allocation (LDA) method, one of the most popular topic modelling algorithms [ref]. The method treats each document within the data set as a mixture of topics, and each topic as a mixture of words. This approach is particularly useful when analysing organic texts, such as tweets; authors often mention a number of topics within one tweet rather than just one, meaning that documents overlap in terms of topics within them.

The following equation governs the topic assignment process:

$P(w_i \mid d) = \sum_{j=1}^{K} P(w_i \mid k_i = j) P(k_i = j \mid d)$

Where $P(w_i \mid d)$ is the probability of the ith word w_i of the given document d, this is then determined by summing the probability of a word w_i appearing within topic j (β) and the probability of picking a word w_i from the document d (γ) over the total number of specified topics K.

To expand on this definition, each word has a probability of being generated from each topic,β. For example the a word can have 20% chance of being generated from topic 1, 30% from topic 2 and 50% from topic 3. The advantage of these β terms is that is gives the model a good way of representing natural text, as often words can be related to a number of topics.

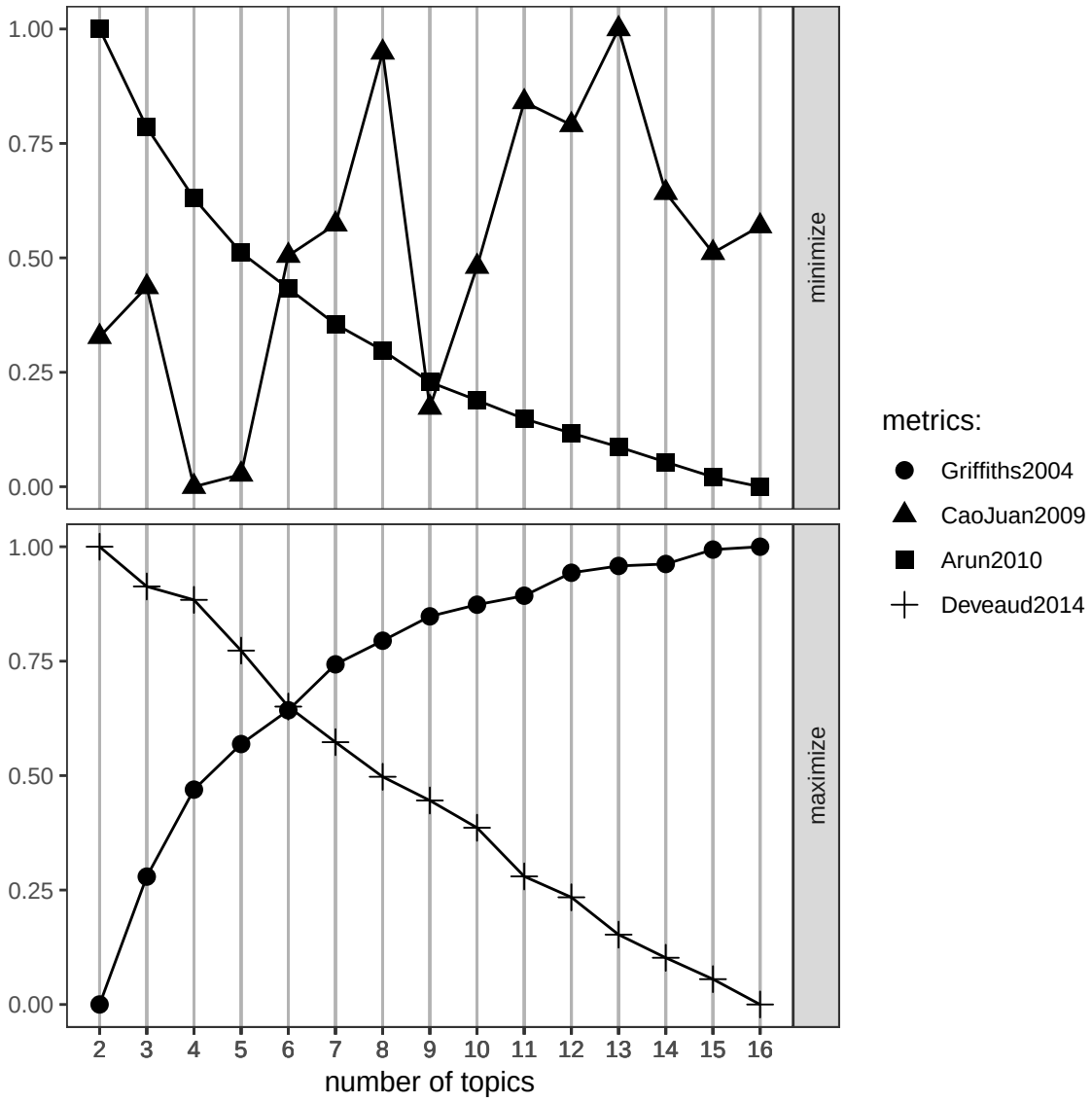
The γ term indicates the proportion of words within a document that are generated from a topic. For example, a document could have a γ value close to zero for topic 1 and a γ value close to 1 for topic 2, indicating that the majority of the words within the document are generated from topic 2. In practise this is often a lot more difficult to assess, for instance, documents often contain a mix of topics, resulting in much lower γ values.

4.3.1 Topic estimation

In order to perform the LDA algorithm, number of clustering groups must be specified. Whilst this would be an easy feat in an example such as a newspaper (as it would contain distinct countable topics: sport, finance, politics) the number in our case must be estimated due to the unknown nature of the topics contained within the tweets. There are a number of ways to estimate number of topics for a text [ref], this package however will use a neat compliation package `ldatuning` [ref] which takes various performance metrics and trials them on a range of topic numbers specified, returning a intuative result for a possible range of suitable topics.

To begin topic estimation, simply pass a tokenized tweet data frame into the `estTopics()` function and specify a range of topic numbers to trial. In this case we will trial topics ranging from 2 to 16 in intervals of 1.

```
#estimate number of topics in topic model
estimatedTopicNumber <- estTopics(Token1,2,16,1)
```



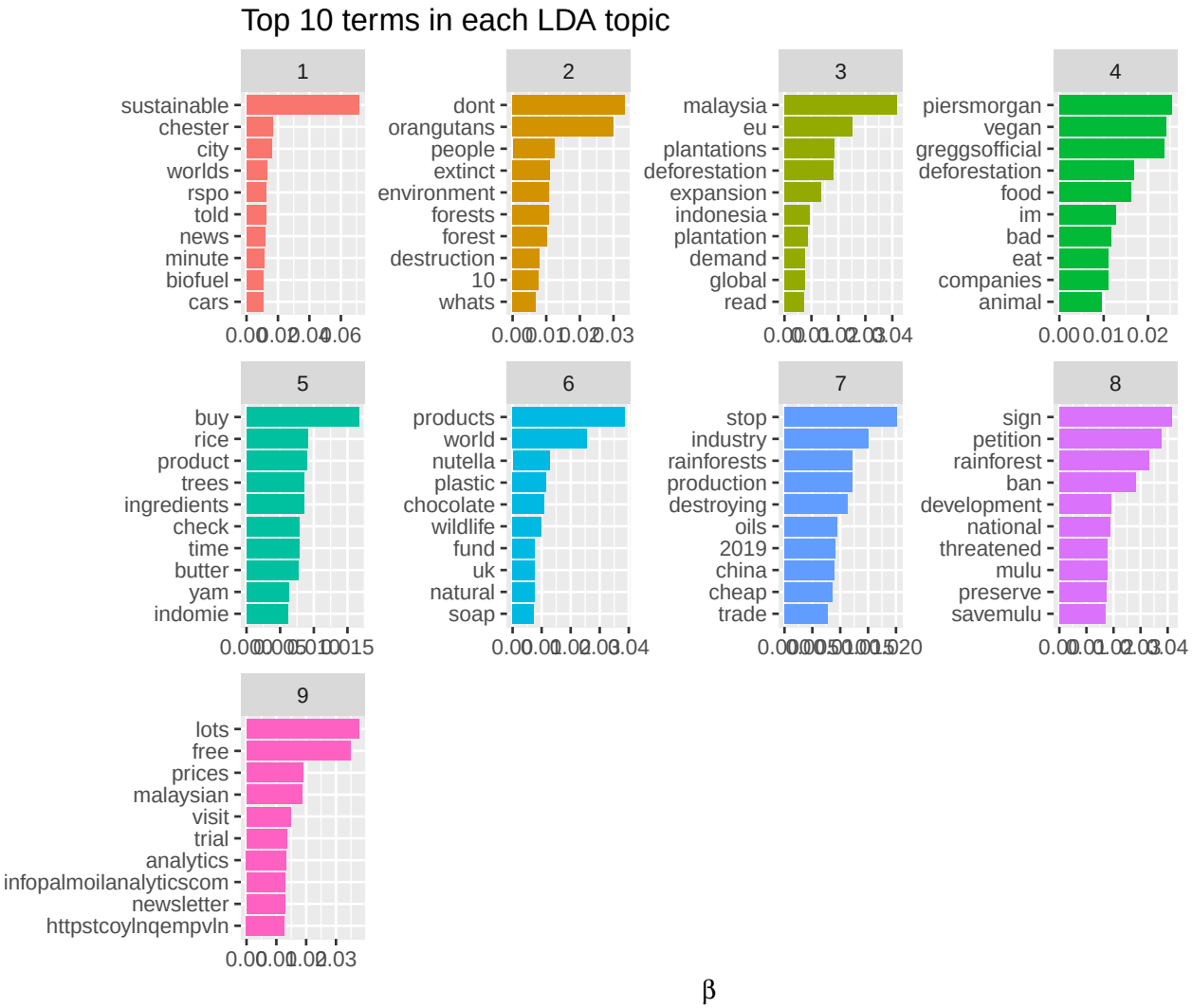
When optimizing hyperparameters for classification and clustering algorithms, it is often the case that there is no optimal value [ref]. Instead, like in this case, a range of possible values are obtained that should be tried in practise. By assessing figure 4.3.1 possible topic numbers could include: 4, 5, and 9.

4.3.2 Creating a topic model

```
#create LDA model with 9 topics
ExampleLDA<-tweetLDA(Token1,9)
```

Once a LDA model has been fitted, it is a good idea to check what topics have been found from the corpus of tweets and whether they make contextual sence in terms of our keywords. The `plotTopicTerms()` function can be used to plot the top terms associated with each topic, along with the term's respective β value.

```
#plot words for 9 topics of model
plotTopicTerms(ExampleLDA)
```

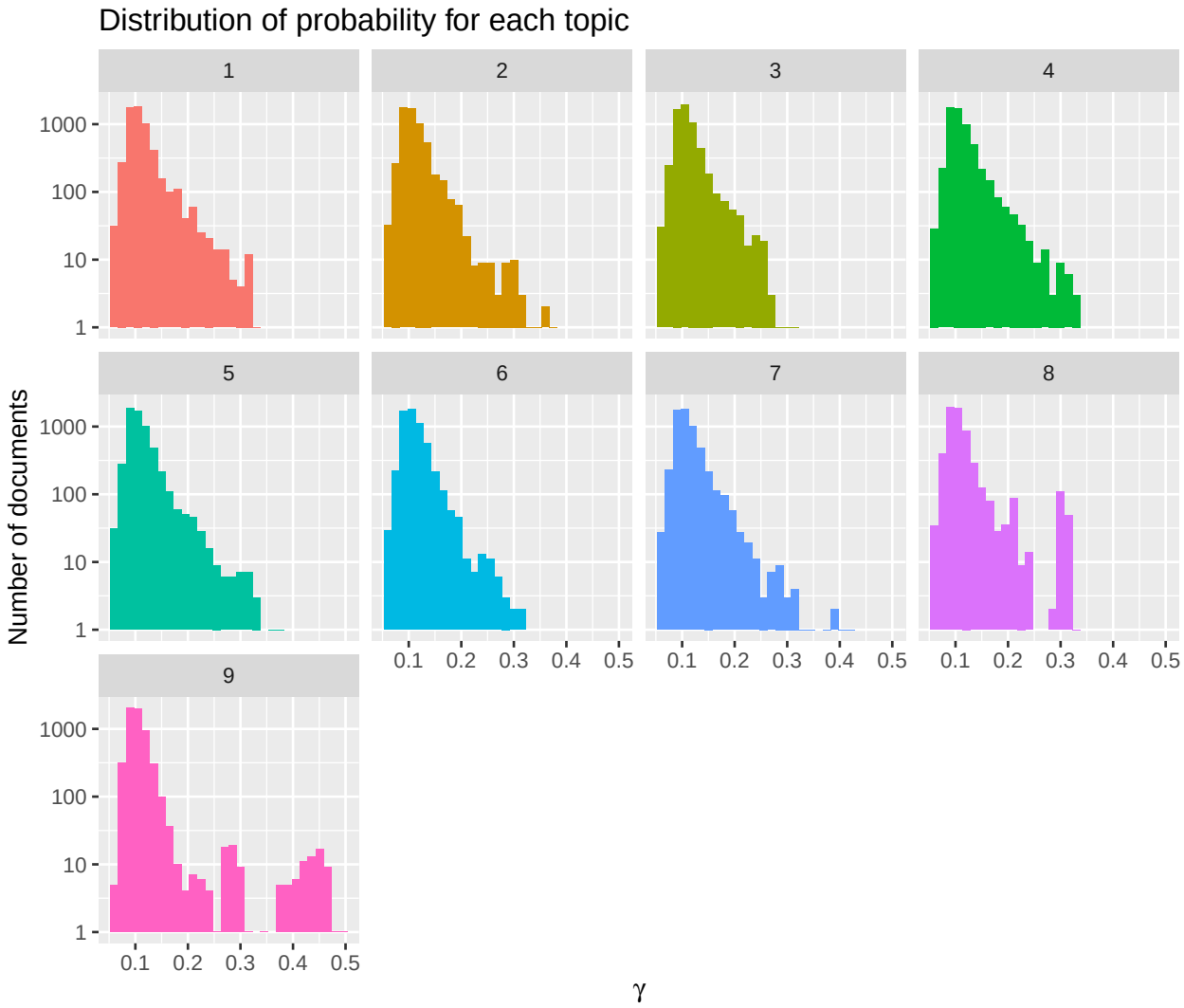


Assessing the highest β terms within a topic shows the words that are most likely to be generated from this topic, giving us an indication of key words surrounding this topic.

Most of the topics make sense and seem appropraite to our search for tweets including "palm oil" and "plastic pollution", one clear result is that not a lot of tweets associated with plastic pollution were detected, as indicated by the lack of the terms within the topics. A lot of the topics plotted make sense, topic 1 is focused around the news of the city of Chester being the first to be palm oil sustainable [ref], topic 6 being about products within the uk which still involve palm oil such as nutella [ref], and topic 4 focusing around Gregg's new vegan sausage roll and introduction of new vegan alternatives within food brands [ref]. Some topics such as 9 are not as clear, to get a better understanding of some of these terms we can refer to the term network plotted earlier, the term network suggests that the news letter is an analytical newsletter and there are tweets raising awareness for it, indicated by the "visit" term. This is also enforced by the "infopalmoilanalyticscom" term suggesting a link to a analytical website regarding palm oil.

The topics seem sensible, one final measure to gauge the accuracy of the model is to plot the γ distribution of the documents within the topics using the `plotTopicDist()` function.

```
#plot distribution of documents within topics
plotTopicDist(ExampleLDA)
```



Each plot shows a distrubtion of the γ values for each of the documents in the corpus respective to that topic, where γ is the estimated proportion of words from that document that are generated specifically from that topic. High spikes close to $\gamma = 0$ in the majority of the topics indicates that there is a high proportion of words in those documents not being generated from that topic; this is expected as most documents will not belong in every topic. Higher γ value spikes indicate that there is a moderate proportion of words within a document that are generated specifically from that topic. For example, topic 9 has a group of documents with γ values relatively higher than the rest of the documents, indicating that around 40% of the words within those documents are generated exclusively from topic 9. Taking this idea back to the terms of topic 9, many were raising awareness and advertising a newsletter about palm oil, in which case the majority of the tweet would be focused around advertising the newsletter.