Tweet analytics using NLP

Prajwal Shreyas Follow

Jul 23 · 9 min read ★

reviews, news feeds etc.

What is Text Analytics?

Text analytics is the processes of synthesising unstructured data to help discover patterns and enable decision making. Until recent years text analytics had to be performed the old fashioned way i.e. eyeballing and manual categorisation of text, which is inefficient and time consuming. Also this is not a practice solution when dealing with millions of documents such as twitter data.

Twitter data (also know as tweets) is a rich source of information on a large set of topics. This data can be used to find trends related to a specific keyword, measure brand sentiment or gather feedback about new products and services. This post will provide a step by step guide for text analytics on twitter data.

How to perform text analytics on twitter data?

The steps involved in text analytics are:

Step 1: Collect tweets
Step 2: Pre-process tweets
Step 3: Apply sentiment analysis
Step 4: Apply named entity recognition
Step 5: Cluster tweets
Step 6: Visualise analysis

Step 1: Collect tweets

The tweet data can be programmatically accessed in two ways i.e. Twitter Rest API or Streaming API.

The Rest API enables you to collect the list of all tweets or followers for a particular user. You can also query a user's account and even modify them provided you have the required permissions. Where as the Streaming API allows you to collect tweets on a real-time basis based on search terms, user ids or locations.

In order to access the API, you will need 4 pieces of information from Twitter i.e. API key, API secret, Access token and Access token secret. The detailed steps for setting up an account and connecting to the twitter API services using python packages are covered in the blog here.

For our analysis I have used the twitter streaming API service and collected tweets for twenty FTSE 100 companies over a week starting from 1st October 2016. This has resulted in a total of 37,313 tweets. The output of the streaming API needs to be parsed into a suitable format before any analysis. In the next section we cover the steps involved in pre-processing of tweets.

Step 2: Pre-Process tweets

Here we parse the response from the twitter API into a structured table. The response from twitter streaming API's is in the below format:

The detailed code for parsing the output from the twitter API is below. The output is structured into key fields such as "doc_id", "username", "text" etc. The parsed response can be stored in a database or a JSON file.

```
import simplejson as json
```

```
  #hashtags are identified and provided as a field in the tweet
    tweetItem['hashtags'] = map(lambda x: x['text'],doc['entities']
['hashtags'])

    # user_mentiones are identified and provided as a field
    tweetItem['user_mentions'] = map(lambda x:
x['screen_name'],doc['entities']
    ['user_mentions'])
    # symbols e.g. $APPL are identified and provided as a field
    tweetItem['symbols'] = map(lambda x: x['text'],doc['entities']
['symbols'])
    tweetItem['coordinates'] = doc['coordinates']
    tweetItem['user_id'] = doc['user']['id']
    tweetItem['user_name'] = doc['user']['name']
    try:
        tweetItem['retweet_id'] = doc['retweeted_status']['id']

    except KeyError as e:
        tweetItem['retweet_id'] = 0
        pass

  return tweetItem
```

Step 3: Sentiment Analysis

People express their opinions via tweets and these usually have sentiment associated with it i.e. positive, negative or neutral. Analysis of this sentiment may lead to some useful insight on the topic or company being discussed. Hence we suggest use of sentiment analysis algorithms to perform this analysis. I have created a separate blog with details on performing sentiment analytics using the deep learning algorithm here.

The sentiment is a useful indicator, however tweets contain additional information such as names of people, places and organisations. The data when extracted may lead to very useful insights on your product or company. So in the next step we cover the usage of named entity recognition algorithms, which is designed to extract this information.

Step 4: Named Entity Recognition

The Named Entity Recognition algorithm will identify key tags such as persons, locations or organisations contained within the tweets. For example, if there's a mention of "London" in your tweet the algorithm would tag that as "Location". However as tweets generally do not follow the standard english syntax, the accuracy of the tags is around 60% to 70%.

You can use the Stanford NER tagger to obtain the entity recognition for the tweets. In the example below I have used a tweet by Donald Trump.

```
# -*- coding: utf-8 -*-

from nltk.tag import StanfordNERTagger
```

```
#a tweet by Donald Trump
text = 'Just had a very good call with @SwedishPM Stefan Löfven who
assured me that American citizen A$AP Rocky will be treated fairly.
Likewise, I assured him that A$AP was not a flight risk and offered
to personally vouch for his bail, or an alternative....'

tokenized_text = word_tokenize(text)
classified_text = st.tag(tokenized_text)

print(classified_text)
```

Step 5: Clustering of similar tweets

A number of tweets are very similar to each other, this may be due to people tweeting about the same topic this causes a lot of noise in the data. Hence in order to reduce the noise it would be useful to cluster the tweets into groups based on their similarity. This can be done using the clustering algorithm.

The inner workings of the algorithms is based on a machine learning concept know as tf-idf (term frequency — Inverse document frequency) matrix. Here the each tweet is treated as a document (d) and the words within the tweet as term (t).

Term Frequency-Inverse Document Frequency is a numerical statistic that demonstrates how important a word is to a corpus.

Term Frequency is just ratio number of current word to the number of all words in document/string/etc.

$$tf(t,d) = \frac{n_t}{\sum_k n_k}$$

Term Frequency Formula

Frequency of term t_i, where n_t — the number of t_i in current document/string, the sum of n_k is the number of all terms in current document/string.

Inverse Document Frequency is a log of the ratio of the number of all documents/string in the corpus to the number of documents with term t i.

$$idf(t,D) = \log \frac{|D|}{|\{ d_i \in D \mid t \in d_i \}|}$$

Inverse Document Frequency Formula

tf-idf(t, d, D) is the product tf(t, d) to idf(t, D). Additional details on tf-idf is available in the blog here.

$$tf\_idf(t,d,D) = tf(t,d) \cdot idf(t,D)$$

threshold in the function *"get_clusters(processedTweets, similarity_threshold)"*. Hence all tweets, which are 80% similar will be clustered together based on similarity measure known as cosine similarity.

```
def cluster_text(dist_text,processedTweets,dist_threshold_txt):
    cluster_id = 1
    for i in range(dist_text.shape[0]):
        try:
            doc_dist_array_txt = dist_text[i,]
            # identify document ids where the tweets are similar
            similarDocs = np.where(doc_dist_array_txt <=
dist_threshold_txt)[0]

processedTweets.ix[processedTweets.index.isin(similarDocs) &
processedTweets['cluster_ref'].isin([None]), 'cluster_ref'] =
cluster_id
            cluster_id = cluster_id + 1
        except ValueError:
            continue

    return processedTweets



# %%%%%%%% Calculate Cosine distance of documents %%%%%%%%%%%%

def tokenize_only(text):
    # first tokenize by sentence, then by word to ensure that
punctuation is caught as it's own token
    tokens = [word.lower() for sent in nltk.sent_tokenize(text) for
word in nltk.word_tokenize(sent)]
    filtered_tokens = []
    # filter out any tokens not containing letters (e.g., numeric
tokens, raw punctuation)
    for token in tokens:
        if re.search('[a-zA-Z]', token):
            filtered_tokens.append(token)
    return filtered_tokens

def get_clusters(processedTweets, similarity_threshold):
       tweets = processedTweets['text'].tolist()

       #Define count vectorizer parameters
       vectorizer =  CountVectorizer(max_df=1.0, min_df=1,
stop_words=stopwords, tokenizer=tokenize_only)

       # Get document term frequency matix
```

```
        processedTweets['cluster_ref'] = None # Tweets that are 1-
similarithy% similar, as defined by dist_threshold
        # Clustering of tweets based on text
        processedTweets =
cluster_text(dist_text,processedTweets,dist_threshold_txt = (1-
similarity_threshold))

        return processedTweets
```

The results will contain an additional column “cluster_ref”. The cluster_ref will group all the similar documents into a single group for example if document 1 and document 2 are similar, hence both have cluster_ref of 1.

Step 6: Visualisation of results

The visualisation of the above results would help drive insights into the consumer opinion about the product or company. In the below section I will be using Tableau for visualisation purposes. Please see the link to the dashboard here.

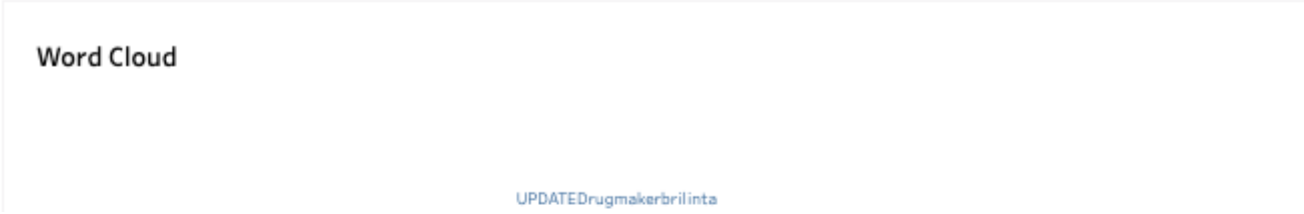
The first graph shown below is the positive, negative and average sentiment (Blue line) of each company. The bars represent the average sentiment for the company and is not weighted for the number of tweets. This **unweighted** sentiment does not provide an accurate comparison between companies as it does not take into account the number of tweets.

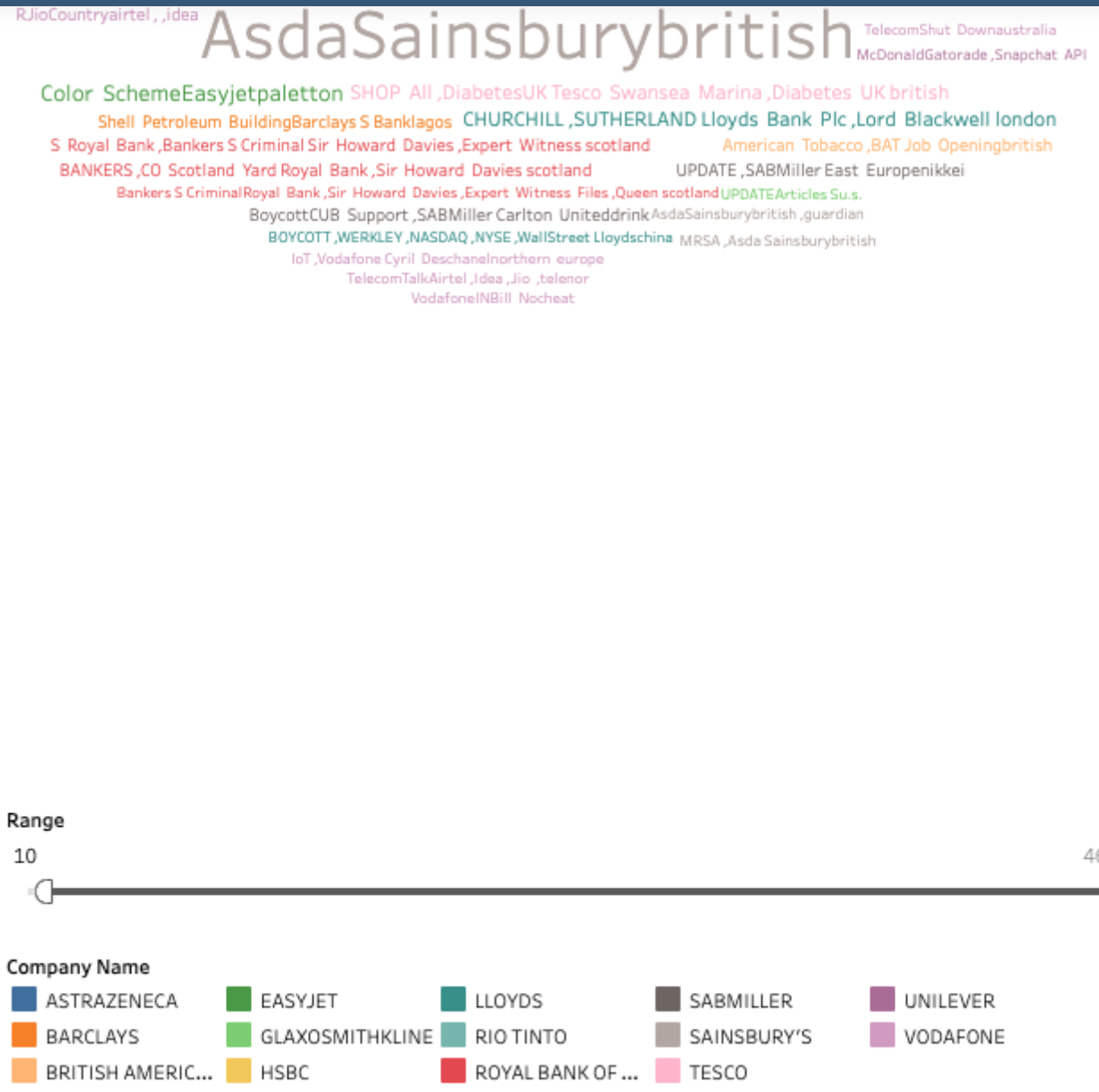
much greater than the negative sentiment.

Sentiment Weighted

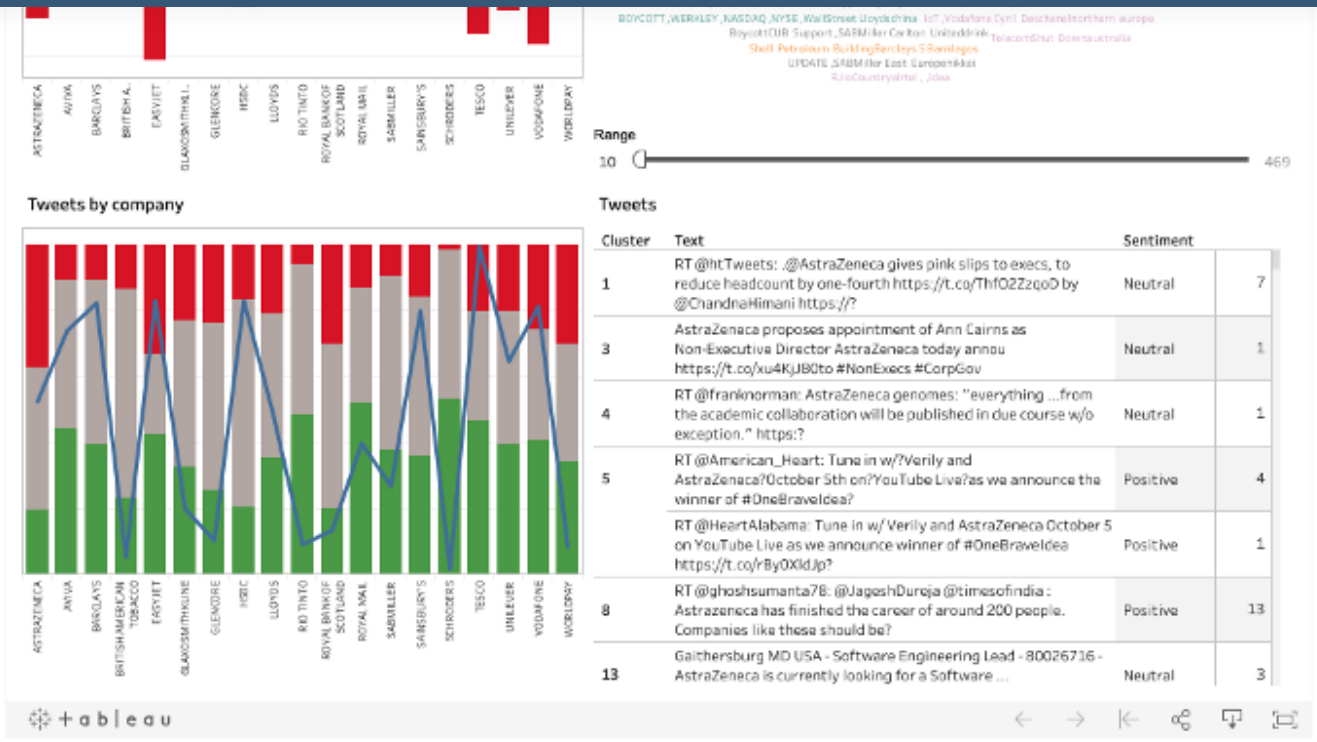
It is useful to see the proportion of positive, negative and neutral tweets for each company as shown below. This can lead to some interesting insights about the brand health of a company. For example we can see that for Schroders despite having a low weighted sentiment it has a very high proportion of positive tweets. This analysis when conducted over time can serve as an indicator of brand health.

The named entities which we have extracted using our named entity recognition algorithm can be visualised using a word cloud. In the below section I have created a word cloud in order to help with the analysis. The words are coloured by the companies and sized by the number of times it is repeated in the analysis. As default the word cloud is representing the entities which have been repeated at least 10 times. However this can be changed by moving the slider control at the bottom of the dashboard. The word cloud provide an easy way of understanding the most prominent topics being discussed in the tweets.





We can use the combined dashboard seen below to explore the topics driving the sentiment. We can see from instance from the word cloud that the Phrase “AsdaSainsburyBritish” is highlighted as the most prominent phrase. When we click on it we can see that it is due to tweets about a superbug found in the meat sold at these retailers. This can also be used to see what is driving the large negative positive sentiment in specific companies. For example by clicking on the red bar on the first chart for AstraZeneca and selecting the phrase from the word cloud we can see that the driving factor for negative sentiment here is a failed drug trial for Brilinta.



Conclusion:
In conclusion the above techniques can be used to analyse huge volumes of twitter data. Additionally the algorithms can be applied to any kind of text data such as news articles or customer service chat data.

NLP | Twitter Analytics | Data Science | Python | Sentiment Analysis

WRITTEN BY

Prajwal Shreyas

Follow

Data Scientist/ ML Engineer — Experienced in building and deploying large scale ML models to enhance business value.

Write the first response

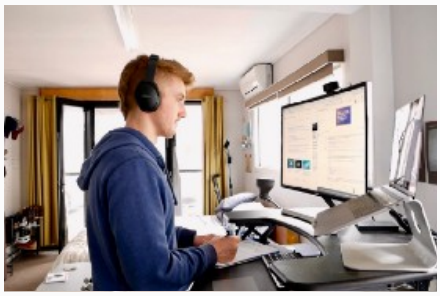
More From Medium

More from Towards Data Science

The 5 Sampling Algorithms every Data Scientist need to know

Rahul... Jul 20 ... 374

More from Towards Data Science



12 Things I Learned During My First Year as a Machine Learning Engineer

Daniel... Jul 6 · 1... 6K

More from Towards Data Science

P-values Explained By Data Scientist

Admon... Jul 13 · ... 1K