Web Science Assessed Exercise Report

Rishi Vinod (2331751v)

Introduction

For this exercise, I attempt to develop a data crawler to complete network based social media analytics. This crawler uses the Twitter Streaming API to collect data from tweets in real time. Upon collecting data using the streaming API, I stored it in a local database and create clusters of information such as hashtags, usernames and keywords. I can then analyse all the information stored in the database and from the trends observed through clustering. To begin this task, we require access to the Twitter API which is obtained upon gaining Twitter Developer status. Once this was done, I was given access keys and consumer API tokens which allowed me to develop my crawling app locally. I could then access the information which was being tweeted at that time. I installed mongoDB Community, pymongo and Compass to visualise the database and my data collection. With the help of the Twitter API documentation, I understood how to use the Twitter Streaming API. The Tweepy API allowed me to access all of Twitter's RESTful API methods which meant that I could build a very efficient Twitter crawler. In order to find the trending keywords in the UK, I created a method to use the tweepy API to retrieve those keywords with the WOE (Where on Earth) ID of UK. The location was filtered with the United Kingdom co-ordinates.

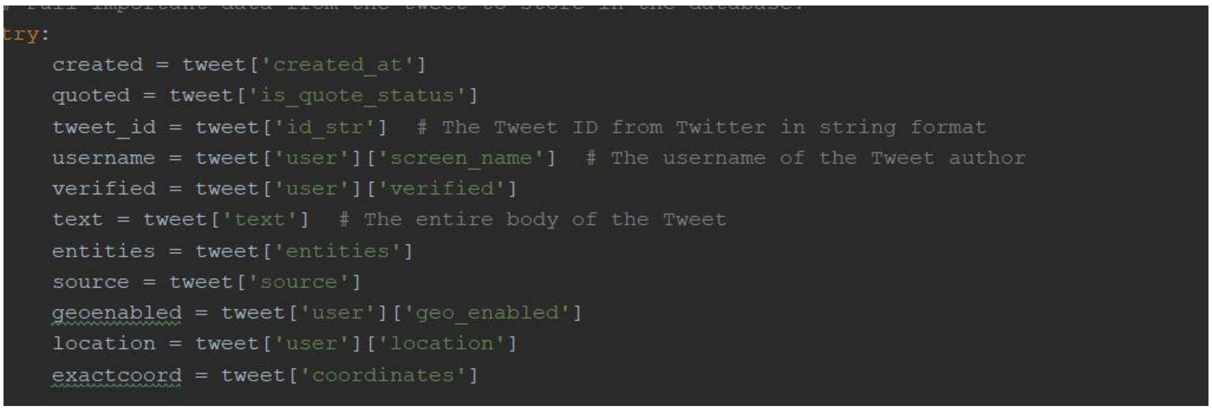
Data Crawl

The data collected using the streaming API on 22/03/2021 between 18:10 and 18:40. This data was then stored within the tweet collection in my database using MongoDB. As we were to collect around 1% of tweets, I decided to only run my data crawler for around 30 minutes collecting 13,917 tweets (including retweets). This is notably less than the recommended crawling time of 1 hours however a smaller dataset had to be used in order to feasibly run the analytics functions, as my laptop was having issues in the processing power for a large data set. Despite the relatively small amount of data, it was still enough to demonstrate the functioning of the software.

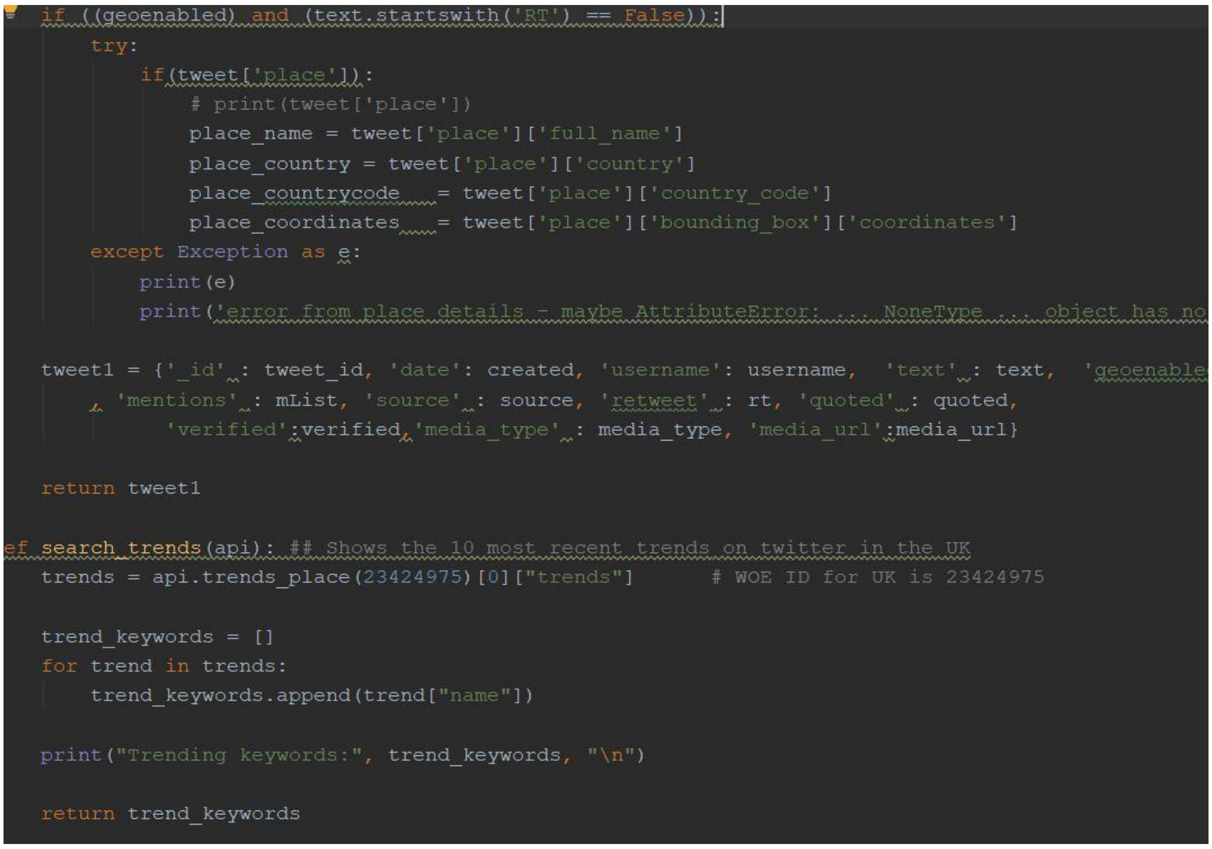
The Tweepy library was used to crawl and gather data for the software through the "streamCrawler.py" script. I scrape the current most trending keywords in the UK through tweepy, and then uses them to filter tweets gathered through the Streaming API. These keywords will aid in gathering tweets that are linguistically homogeneous and coming from a wide range of sources across the Twitter community.

For both Streaming and REST crawling, the tweets are gathered in extended mode in order to collect the entire tweet, the text is cleaned up by removing URLs, broken symbols, emojis and new lines. The tweet information which is gathered includes:

- Tweet ID
- Time posted
- Username
- Tweet text
- Geoenabled Status
- Location
- Place name, country, coordinates, and country code
- Tweet Source
- Retweet Status
- Quoted Status
- Hashtags and Mentions List
- Verified Status
- Media Type and URL

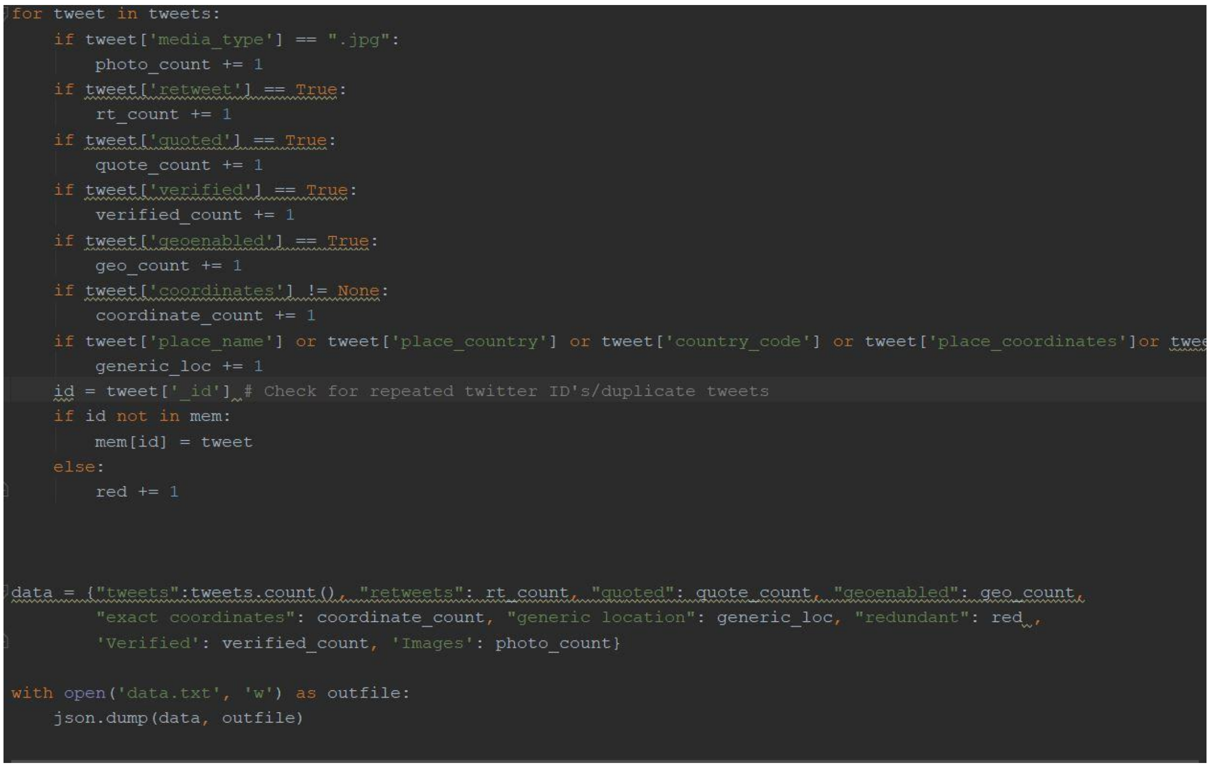


```
# Pull important data from the tweet to store in the database.
try:
    created = tweet['created_at']
    quoted = tweet['is_quote_status']
    tweet_id = tweet['id_str']  # The Tweet ID from Twitter in string format
    username = tweet['user']['screen_name']  # The username of the Tweet author
    verified = tweet['user']['verified']
    text = tweet['text']  # The entire body of the Tweet
    entities = tweet['entities']
    source = tweet['source']
    geoenabled = tweet['user']['geo_enabled']
    location = tweet['user']['location']
    exactcoord = tweet['coordinates']
```



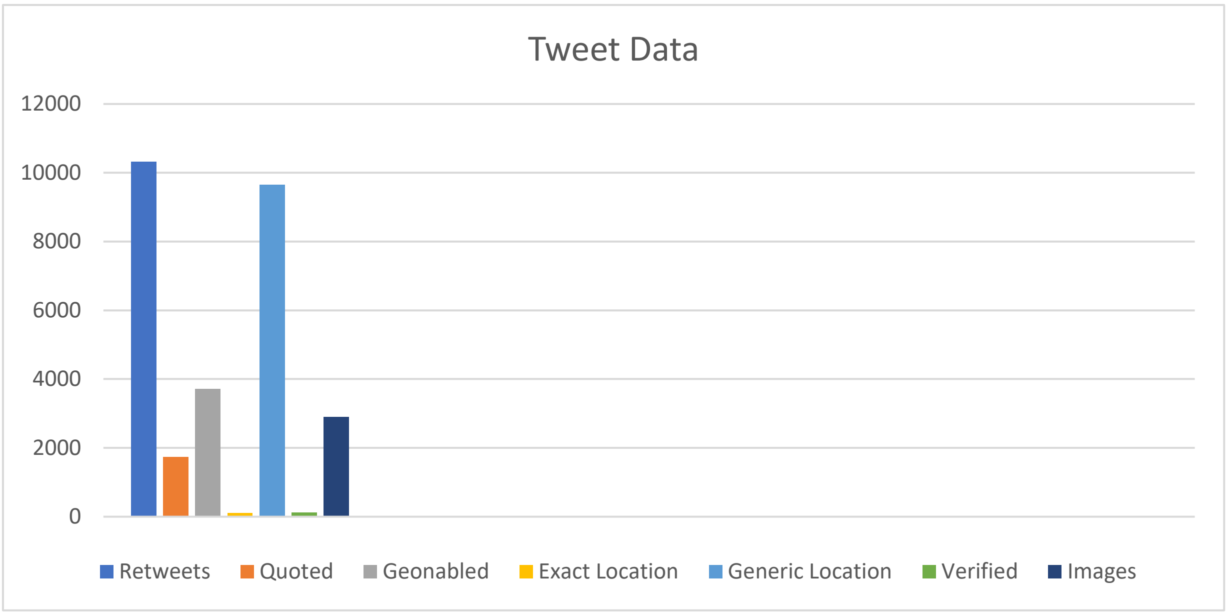
```
    if ((geoenabled) and (text.startswith('RT') == False)):
        try:
            if(tweet['place']):
                # print(tweet['place'])
                place_name = tweet['place']['full_name']
                place_country = tweet['place']['country']
                place_countrycode   = tweet['place']['country_code']
                place_coordinates   = tweet['place']['bounding_box']['coordinates']
        except Exception as e:
            print(e)
            print('error from place details - maybe AttributeError: ... NoneType ... object has no

    tweet1 = {'_id' : tweet_id, 'date': created, 'username': username,  'text' : text,  'geoenable
        , 'mentions' : mList, 'source' : source, 'retweet' : rt, 'quoted' : quoted,
            'verified':verified,'media_type' : media_type, 'media_url':media_url}

    return tweet1

ef search_trends(api): ## Shows the 10 most recent trends on twitter in the UK
    trends = api.trends_place(23424975)[0]["trends"]      # WOE ID for UK is 23424975

    trend_keywords = []
    for trend in trends:
        trend_keywords.append(trend["name"])

    print("Trending keywords:", trend_keywords, "\n")

    return trend_keywords
```

The gathered tweets are then saved to MongoDB, indexed by the Tweet ID before being grouped. In the “data_output.py” file I read the database to count the occurrences of certain fields for analysis purposes and write it to data.txt in JSON format.



```
for tweet in tweets:
    if tweet['media_type'] == ".jpg":
        photo_count += 1
    if tweet['retweet'] == True:
        rt_count += 1
    if tweet['quoted'] == True:
        quote_count += 1
    if tweet['verified'] == True:
        verified_count += 1
    if tweet['geoenabled'] == True:
        geo_count += 1
    if tweet['coordinates'] != None:
        coordinate_count += 1
    if tweet['place_name'] or tweet['place_country'] or tweet['country_code'] or tweet['place_coordinates']or twee
        generic_loc += 1
    id = tweet['_id'] # Check for repeated twitter ID's/duplicate tweets
    if id not in mem:
        mem[id] = tweet
    else:
        red += 1



data = {"tweets":tweets.count(), "retweets": rt_count, "quoted": quote_count, "geoenabled": geo_count,
        "exact coordinates": coordinate_count, "generic location": generic_loc, "redundant": red ,
        'Verified': verified_count, 'Images': photo_count}

with open('data.txt', 'w') as outfile:
    json.dump(data, outfile)
```

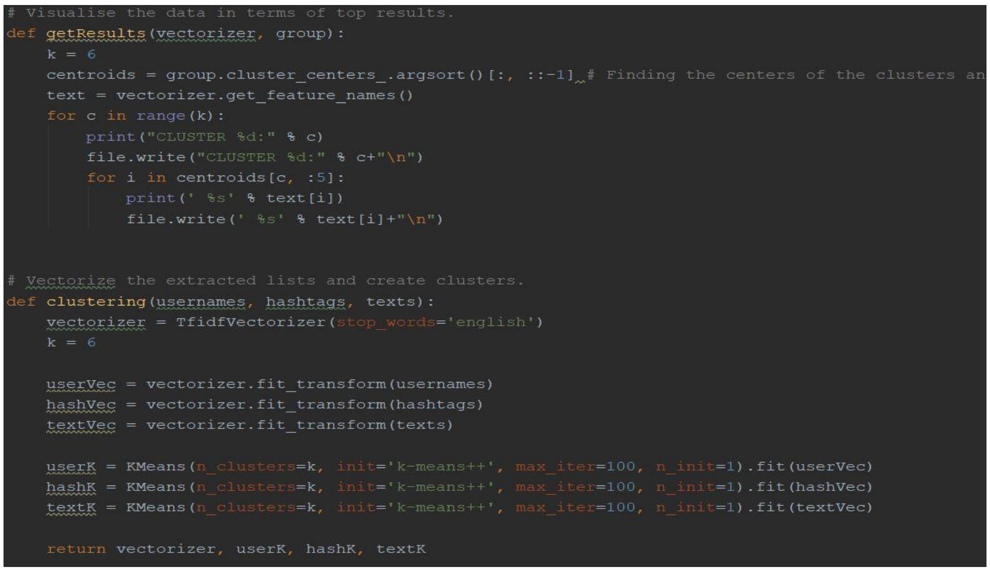
The data found is as follows:



Tweets	Retweets	Quoted	Geoenabled	Exact Coordinates	Generic Location	Redundant	Verified	Images
13917	10328	1733	3719	106	9652	0	118	2894

A majority of tweets retrieved in this time period were retweets, and as expected the number of users with generic location data was significantly higher than those with precise coordinates. A minority of users were found to be verified, which is also expected considering that status is given to accounts that have top tier popularity/following.

Once I had collected all the tweet metadata into my database, (the dataset used for this report's data is saved into the work space folder as "TweetColl.csv"), then "tweetClusters.py" is used to read from the database and the tweets are grouped using K-Means clustering, using functions from the SKLearn library; the vector quantization allowed me to produce groups of hashtags, usernames and text so that I could analyse it. The objective of K-means is to simply group similar data points together and discover underlying patterns. To do this, K-means looks for a fixed number *k* (the number of centroids you need in the dataset) of clusters in a dataset. The output in my file is 6 clusters per group (k=6), making 18 clusters, with 5 elements per cluster, and they are written to "cluster .txt".



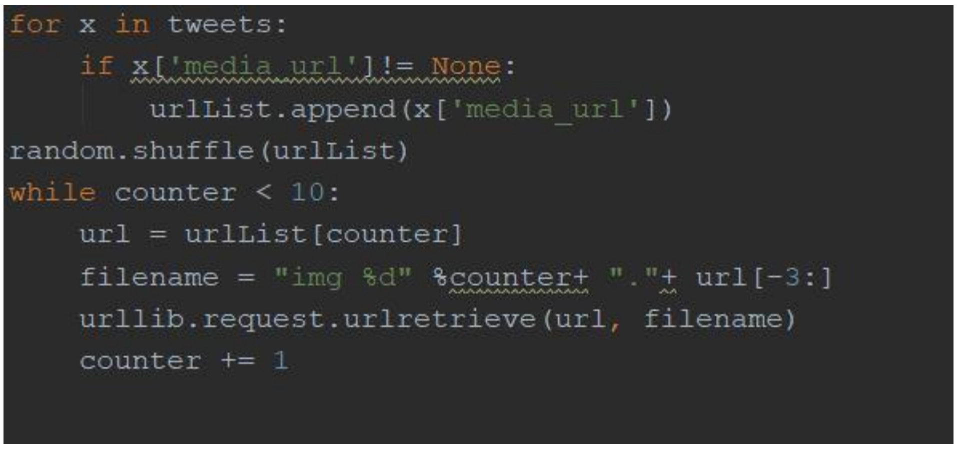
```
# Visualise the data in terms of top results.
def getResults(vectorizer, group):
    k = 6
    centroids = group.cluster_centers_.argsort()[:, ::-1] # Finding the centers of the clusters an
    text = vectorizer.get_feature_names()
    for c in range(k):
        print("CLUSTER %d:" % c)
        file.write("CLUSTER %d:" % c+"\n")
        for i in centroids[c, :5]:
            print(' %s' % text[i])
            file.write(' %s' % text[i]+"\n")


# Vectorize the extracted lists and create clusters.
def clustering(usernames, hashtags, texts):
    vectorizer = TfidfVectorizer(stop_words='english')
    k = 6

    userVec = vectorizer.fit_transform(usernames)
    hashVec = vectorizer.fit_transform(hashtags)
    textVec = vectorizer.fit_transform(texts)

    userK = KMeans(n_clusters=k, init='k-means++', max_iter=100, n_init=1).fit(userVec)
    hashK = KMeans(n_clusters=k, init='k-means++', max_iter=100, n_init=1).fit(hashVec)
    textK = KMeans(n_clusters=k, init='k-means++', max_iter=100, n_init=1).fit(textVec)

    return vectorizer, userK, hashK, textK
```

The findings are as shown below, and because the crawling process focuses mainly on collecting tweets related to particular trending keywords and thus the tweets are more likely to be relatively similar and require less clusters. The clustering program works quicker on smaller datasets which serves our purpose because the amount of data retrieved in the 30 minutes was enough to cluster swiftly.

	TOP USERNAMES	TOP HASHTAGS	TOP TEXT
CLUSTER 0	olin orangeskiies inguinal follow internet	aaaaaaaaaaaa admired 20 advocating adik	mingi yunho called say mingilights
CLUSTER 1	cards cumpleaños curse currently current	18th afraid admired 38_degrees affects	water rt worldwaterday yunho kreme
CLUSTER 2	chewed plus cushion curse currently	100 affected 18th afraid affects	happyrenjunday renjun ouruniverserenjunday 런쥔이란_영화의_스물두번째_장면 birthday
CLUSTER 3	20 plus curb curse currently	aesgf6 9gag 38 aaaaaaaaaaaa 87	happy birthday renjun ouruniverserenjunday happyrenjunday

CLUSTER 4	example plus curated current curious	adik ada afternoon 3333 33	celebrateforyuyu 티니들의행복_윤호야_생일_축하해 ateezofficial yunho ateez
CLUSTER 5	participating plus curb currently current	4jaemiin aesgf6 9gag affinityspotlight 38	falling doyoung cover harry styles

The “media_downloader.py” file, in the media folder, is used to read all the available media url’s from the database and download 10 random media files. The reason for downloading just 10 is to reduce storage issues. The “urllib” library is used to retrieve the url’s and download them. It was found that most media files were image files of either .jpg or .png extension, this may be due to the limited amount of data retrieved from the streaming API, and the time of retrieval. I found it almost impossible to retrieve any tweets having video files / .mp4 files.



```
for x in tweets:
    if x['media_url']!= None:
        urlList.append(x['media_url'])
random.shuffle(urlList)
while counter < 10:
    url = urlList[counter]
    filename = "img %d" %counter+ "."+ url[-3:]
    urllib.request.urlretrieve(url, filename)
    counter += 1
```

For the Search API you can make 180 calls per 15 minutes, so the counter is set to a while loop under 180 iterations, and with sleep duration of 15 minutes to meet the Twitter 15 minute restriction. I set the timer to start in the on_connect method (with time limit set to 300 seconds) and let the on_data method add tweet data from the streaming API to my collection until the time limit is reached. The REST and Streaming API run simultaneously (‘is_async’ = true in the stream filter.)

Bibliography

1. https://developer.twitter.com/en/docs

2. http://docs.tweepy.org/en/latest/

3. https://developer.twitter.com/en/docs/tweets/search/overview/standard

4. https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1