WEBSCIENCE COURSEWORK

Student: 2192793m

Andrew Mackay
2192793m@student.gla.ac.uk

Section 1: Introduction

Four different crawlers have been developed for this exercise. Three for Twitter and one for Flickr. The details of each crawler are specified in Section 2 (basic crawler, enhanced crawler, geo-tagged Glasgow crawler) and Section 5 (flickr crawler).

The crawlers developed for Twitter rely on the Tweepy package [1] to interface with the API. The Flickr crawler makes use of flickrapi package [2] for accessing Flickr's API. All data has been stored using MongoDB [3]. To interface with the database the package pymongo [4] was used.

Alongside the crawlers, various visualization scripts have been developed. These rely on Matplotlib [5] for producing the graphs.

For manipulating data, the package Pandas [6] combined with Numpy [7] is used.

Software for clustering and assigning locations to tweets has also been developed. To convert the corpus into a TF-IDF matrix, Sklearn's [8] TfidfVectorizer is utilized. For clustering of the tweets, Sklearn's KMeans is also used combined with Sklearn's joblib for saving of the k-means object to save expensive re-computing each time.

All software was developed using Python and to ensure the crawlers all ran for 1 hour, python's time and datetime modules were utilized.

Table 1 summarizes the time and duration the data was collected over:

Collection	Date	Start Time	End Time	Duration(hours)
Basic Crawler (1a)	14/11/2018	11:00	12:00	1
Enhanced Crawler (1b)	14/11/2018	12:00	13:00	1
Geo-tagged Crawler (1c)	14/11/2018	13:00	14:00	1
Flickr Crawler (4)	17/11/2018	11:00	12:00	1

Table 1 Specification of time and duration of data collected for section 2.

Section 2: Data Crawl

Basic Crawler (1a):

For streaming 1% of the data, twitter's "statuses/sample" API [9] was used. This returns a stream consisting of a random sample of all tweets with the ability to specify the language and this was set to English. The API does not require the input of key words to track.

Enhanced Crawler (1b):

The enhanced crawler relied on the same streaming as described in 1a. To enhance the crawler, two separate types of REST requests were made whilst the streaming API was running. To do this, the streaming method, first REST method and second REST method were all run on separate threads.

The first REST method made use of twitter's "statuses/user_timeline" API [10]. As tweets came in from the stream, the users who tweeted those tweets were added to a queue. In a separate thread the user was removed from the queue then using the "statuses/user_timeline" API, the users most recent tweets were requested.

The second method utilized both twitter's "trends/place" API [11] and "standard search" API [12]. First, the "where on Earth identifier"(WOEID) [13] for the United Kingdom was manually looked up and hard coded in. This was then used as a parameter for the "trends/place" API to obtain the top 50 trending topics in the United Kingdom. The United Kingdom was chosen to help ensure the language of the tweets was in English. For each trend, the "standard search" API was used with the trend as the search query to get a collection of tweets related to the trend topic. Again, the language of the search API was specified to be English.

Geo-tagged Crawler (1c):

To adapt the enhanced crawler from 1b to target geo-tagged tweets for Glasgow some modifications had to be made. First, the "statuses/sample" API as used in 1a and 1b was switched out to use twitter's "statuses/filter" [14] API. The filter API can take in as a parameter a bounding box for a location. To get the bounding box for Glasgow, the website "BoundingBox" [15] was used. This was looked up manually and hard coded in.

As in the original enhanced crawler, the users of the tweets retrieved by the stream were added to a queue and processed on a separate thread. It was thought that a user who had one geo-tagged tweet in Glasgow would likely have more. To retrieve the user's tweets, the same "statuses/user_timeline" API as in 1b was used. To aid this, when retrieving the tweets from the user's timeline, if more than five non-geo-tagged tweets were retrieved then the loop moved onto the next user. This ensured that the request limit was not wasted on users unlikely to return more geo-tagged tweets.

The trending method as described in 1b also needed to be modified. When using the "trends/place" API, the Glasgow WOEID was used instead of the United Kingdom. This then returned the top 50 trending topics in Glasgow. These trending topics were again used as the input to the "standard/search" API, however, the geocode for Glasgow was also passed in for the geocode parameter. To obtain the latitude and longitude for Glasgow, the website "LatitudeLongitude" [16] was used. This was done manually and hard coded in. For the radius around this lat/lng, six kilometres was chosen. The language for the search API remained set to English.

As in 1b, the filter streaming method, user-based REST method and trending-based REST method were all run on separate threads.

Discussion:

As the "statuses/filter", "standard search" and "statuses/user_timeline" API all have separate rate limits, using one does not interfere with the others and therefore guarantees a greater amount of collection in the case when the limits are maxed out. To manage what happens when reaching the limits, Tweepy allows the ability to set a "wait_on_rate_limit" flag when constructing the API instance. This was set to true and simply waited until the limit was reset before continuing the requests.

When making the REST requests there is the ability to restrict the time period. As the task required collecting the highest volume, not necessarily the most relevant tweets, this parameter was not restricted.

The key to collecting as much data as possible was ensuring that the rate limits for each API were being maxed out. To monitor if this was happening, in the main thread, requests to twitter's "application/rate_limit_status" [17] API were made and printed out to the terminal.

Section 3: Basic Data Analytics

Table 2 shows the summary counts for the data collected by each crawler. Table 3 displays the combined total tweets collected and the combined number of Geo-tagged tweets collected by all crawlers.

Collection	Total Tweets Collected	Geo-tagged Tweets	Overlap with 1%	Duplicates	Retweets	Quotes
Basic Crawler 1a	32782	442 (1.35%)	N/A	58	20195	3595
Enhanced Crawler 1b	178486	4816 (2.70%)	N/A	837	109855	14765
Geo-tagged 1c	591	591 (100%)	0	56	0	33

Table 2 Summary counts for the data collected using the various crawlers described in section 2.

Total Tweets Collected	Total Geo-tagged
211859	262 (0.22%)

Table 3 Combined analytical counts for all data collected.

The following graphs (figures 1-9) represent the data collected over the hour. Tweets retrieved by REST requests that were tweeted before the hour period are not shown.

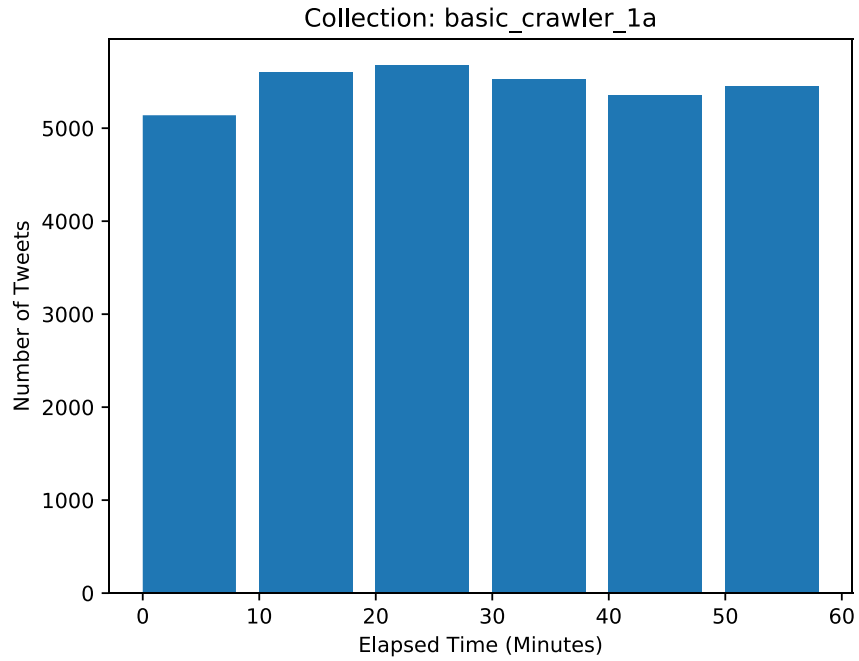

Figure 1 Total number of tweets collected for the basic crawler developed for 1a over 1 hour of crawling.

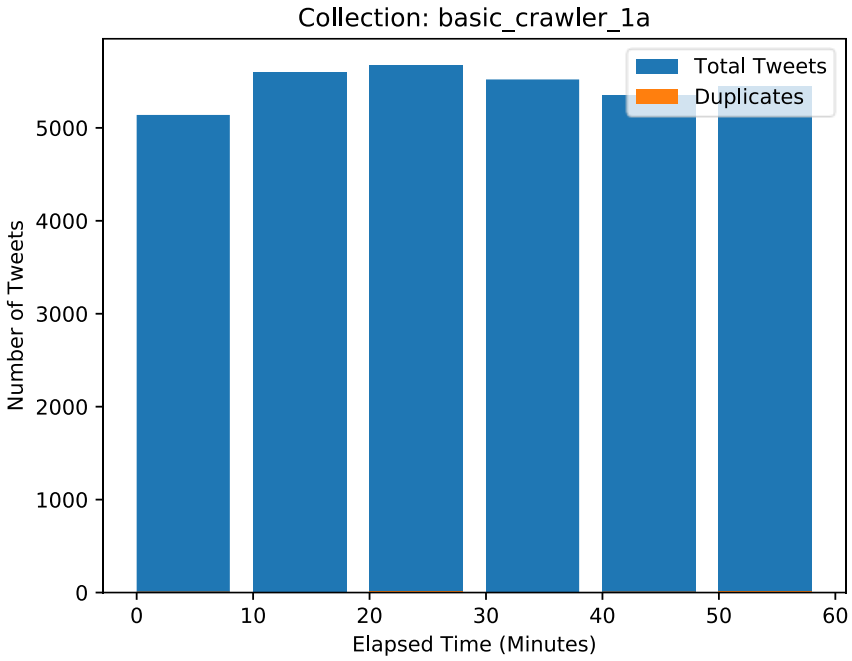

Figure 2 Total number of tweets and number of duplicates collected for the basic crawler developed for 1a over 1 hour of crawling (very few duplicates).

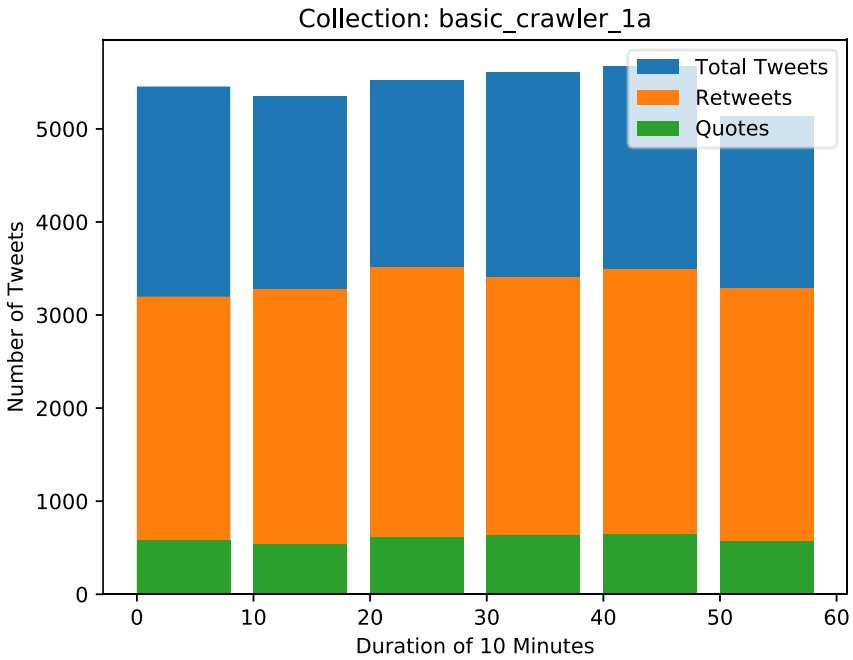

Figure 3 Breakdown of the tweets collected over the 1 hour period for crawler 1a. This shows the number of retweets and quotes that make up the total volume (All bars are measured from the bottom).

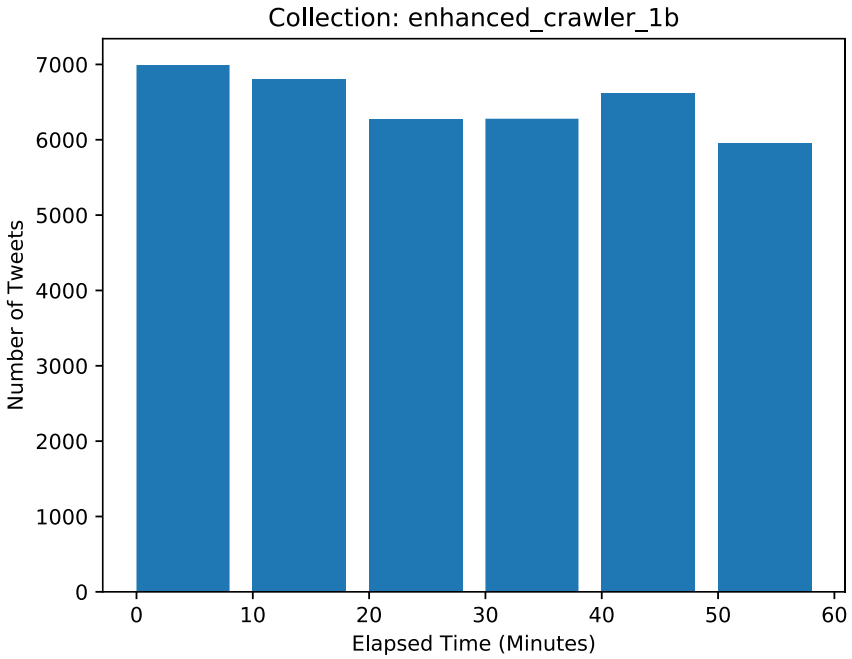

Figure 4 Total number of tweets collected for the enhanced crawler developed for 1b over 1 hour of crawling.

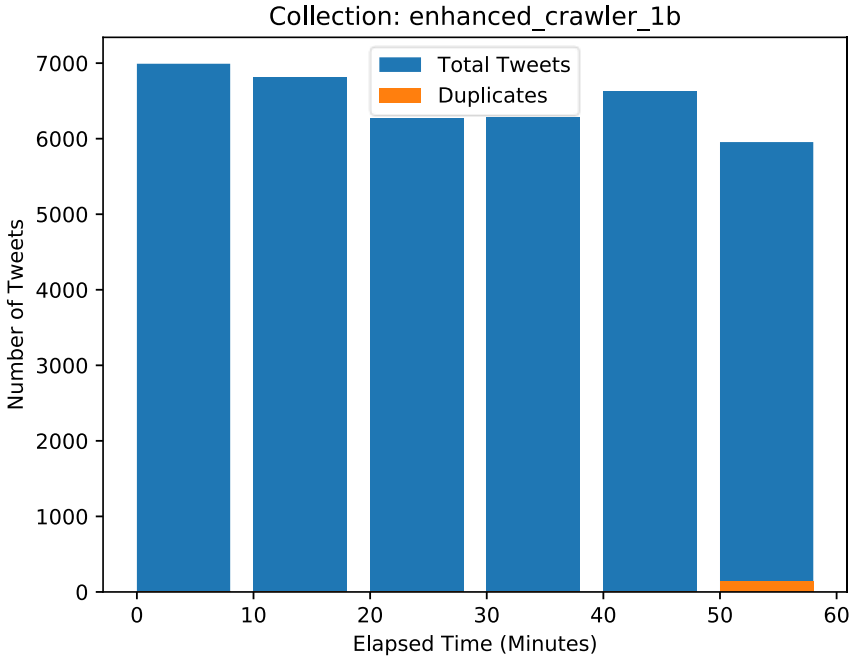

Figure 5 Total number of tweets and number of duplicates collected for the enhanced crawler developed for 1b over 1 hour of crawling.

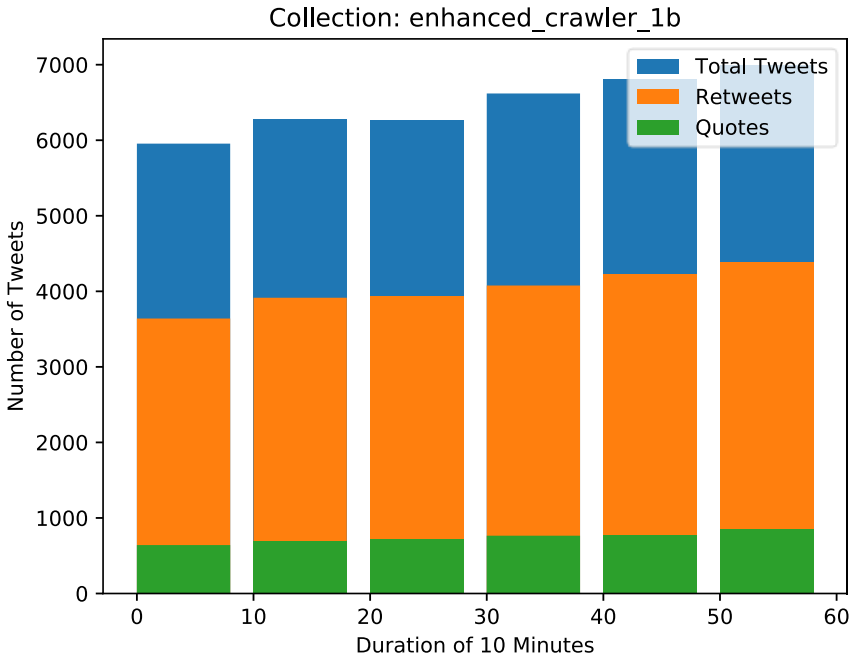

Figure 6 Breakdown of the tweets collected over the 1 hour period for crawler 1b. This shows the number of retweets and quotes that make up the total volume (All bars are measured from the bottom).

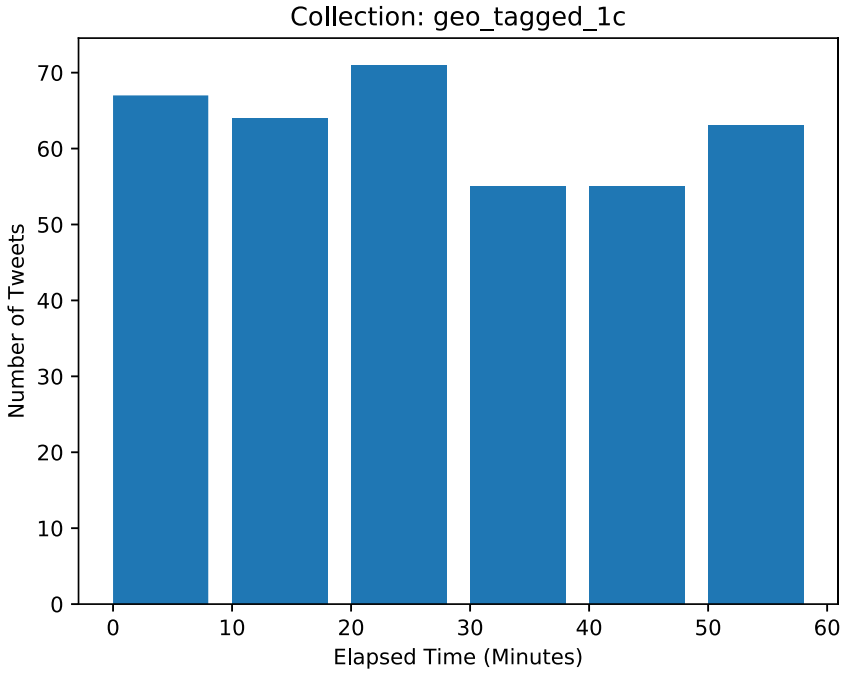

Figure 7 Total number of tweets collected for the Glasgow geo-tagged crawler developed for 1c over 1 hour of crawling.

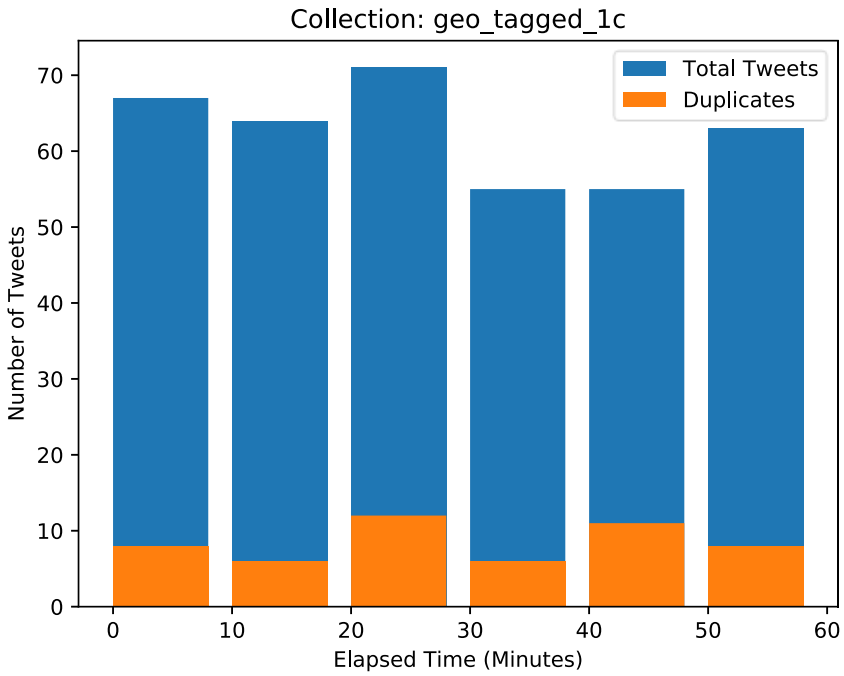

Figure 8 Total number of tweets and number of duplicates collected for the Glasgow geo-tagged crawler developed for 1c over 1 hour of crawling.

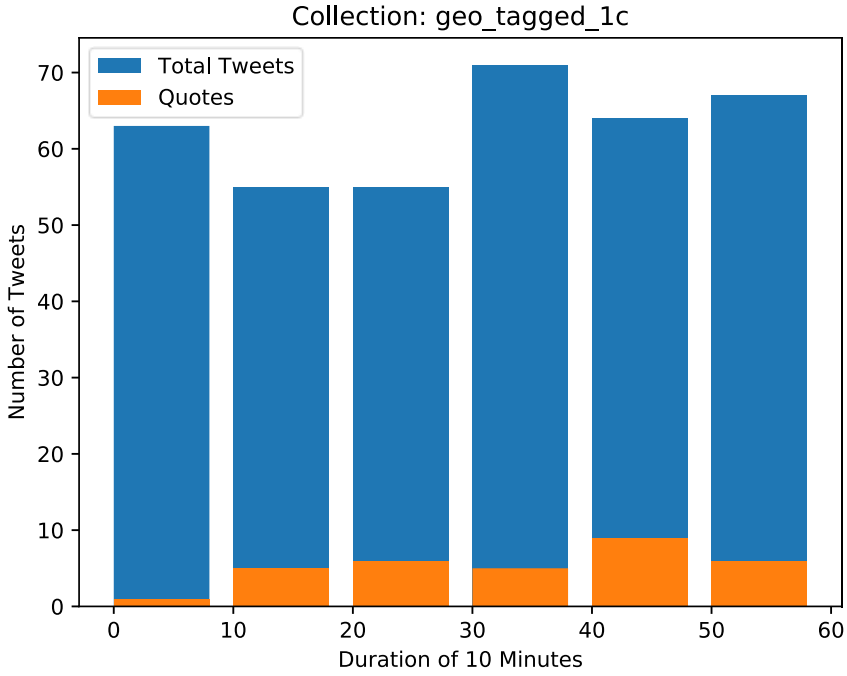

Figure 9 Breakdown of the tweets collected over the 1 hour period for crawler 1b. This shows the number of retweets and quotes that make up the total volume. From the graph you can see that no retweets were collected.

Section 4: Enhancing the Geo-tagged Data

Enhancement of the Geo-tagged data was applied on the collection of data generated from the enhanced crawler developed for task 1b. This set was chosen as it contained the largest amount of

data collected and consisted of a mix of geo-tagged and non-geo-tagged tweets unlike the collection from 1c which would only contain geo-tagged data. The method used to group the tweets relied on a TF-IDF vectorizer and k-means clustering.

To cluster the tweets, they required vectorization. This would allow the clustering algorithm to be applied. The vectorizer chosen for this task utilized the Term Frequency-Inverse Document Frequency (TF-IDF) of the corpus. In this case, the corpus was made up of the text fields of the tweets. Two implementations of the vectorizer were tested, one which used stop words and one that didn't. Adding stop words to the vectorizer did improve the evaluation score by 1% but at the cost of much less evenly distributed cluster groupings. It was decided to use the version without stop words as despite its lower accuracy on the test set, the more even distribution is expected to generalize to other collections better than the stop word version.

Once the TF-IDF matrix was produced, k-means clustering with k = 20 was applied to the matrix to group the tweets.

Figure 10 shows the resultant distribution from the clustering. The figure also shows the amount of geo-tagged tweets per cluster, the number of tweets whose authors had set a profile location and the number of tweets that were assigned a new location by the location assigning method described next.

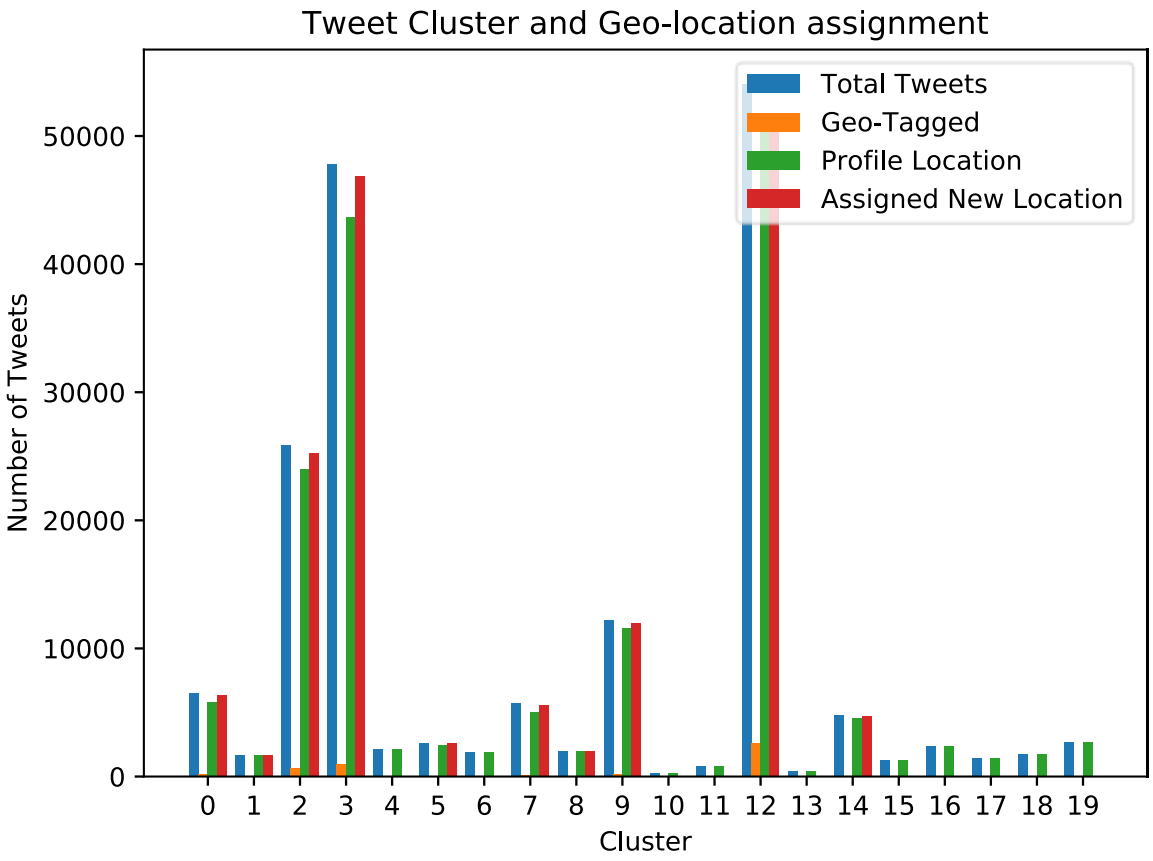

Figure 10 Analysis of tweet clustering combined with geo-location assignment.

For assigning a geo-location for non-geo-tagged tweet the following method was used. For each cluster the most common tweet "place" was calculated. This was then was applied to all tweets in

the cluster that did not already have a location. If the cluster contained no geo-tagged tweets, then no place could be assigned.

Table 4 shows where the geo-assignment by clustering failed. Of the 20 clusters, 10 clusters contained no geo-tagged tweets. However, this only translated to 15185 tweets not being assigned a geo-location as the majority of the tweets were distributed between clusters 2, 3, 9 and 12 which all contained geo-tagged tweets.

Total Clusters Used	Clusters with no geolocation	Total tweets not geo assigned
20	4, 6, 10, 11, 13, 15, 16, 17, 18, 19	15185

Table 4 Summary of the shortcomings of the geo-assignment via clusters.

To evaluate the accuracy of this geo-assignment technique the following process was used. Before vectorizing and clustering, every second geo-tagged tweet had its location set to None. The true location was recorded in a separate dictionary. After the geo-assignment process had finished, the newly assigned tweet location for the originally selected tweets was compared with their actual location. Table 5 shows the results from this evaluation.

Correctly Assigned	Total	Percentage
817	2408	33.9%

Table 5 Evaluation of geo-assignment based on tweet clustering.

Of the 2408 tweets that had their location set to None, 817 had their location correctly assigned by the algorithm.

Section 5: Crawling Data from Flickr

For task 4, a crawler was developed for the social media site Flickr [18].

At first, the crawler attempted to make use of Flickr's Panda streaming API [19]. However, after some investigation it was found that this API was no longer functioning [20]. Therefore, the crawler would have to solely rely on REST requests.

First, using Flickr's "photos.getRecent" API [21] a list of recently uploaded public photos was returned. For each of these photos, the user who uploaded the photo was added to a queue.

Next, using Flickr's "tags.getHotList" API [22] a list of 200 "hot" tags were requested. Then, for each tag, using Flickr's "photos.search" API [23] a list of photos related to the tag was requested. As in the first step, each user who uploaded the photos was added to the queue.

The final stage of the crawler then processed each user that had been added to the queue. For each user, using Flickr's "people.getPublicPhotos" API [24], the users public photos were requested.

Flickr has a very strict restriction on the number of requests that can be made. Flickr only allows 1 request per second and if this is exceeded the API keys can be made invalid [25]. To ensure that the crawler never exceeded this limit, a wait of 1 second was added after each request made. This could potentially be reduced if the user would be willing to risk invalidating their API keys.

The crawler was originally run for 1 hour between 08:00 and 09:00 on the 17/11/2018. As table 6 shows, the crawler produced a very high number of duplicates. After some investigation it was found that Flickr only gives unique photos for the first 4000 images of a search [26]. The number of pages was therefore limited, and the crawler rerun between 11:00 and 12:00 on the same day. Table 7 shows the final results.

Total Photos	Geo-tagged	Duplicates
469673	42493	432540

Table 6 First crawler results with high number of duplicates.

Total Photos	Geo-tagged Photos	Duplicates	Total Views	Average Views per photo	Most Views for an individual photo
318372	88314 (27.74%)	57168	238193258	0.0013	300396

Table 7 Final crawler results with significantly lower number of duplicates.

Figures 11 and 12 show a graphical analysis of the collection. Figure 11 represents the distribution of the photos as collected over the 1 hour period. As the bars are relatively similar in height it suggests that the queue never runs out of requests to make. Figure 12 shows the distribution of the photos by year based on the date they were uploaded. The significantly higher volume for 2018 is likely due to the use of the "getRecent" API. The graph also shows the number of unique photos collected and the number that contain geo information.

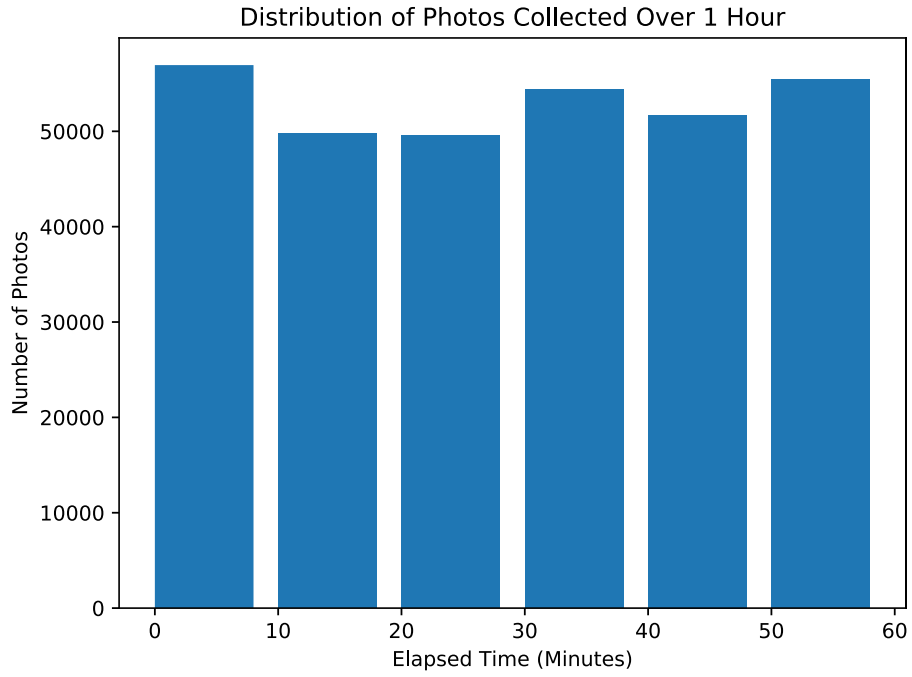

Figure 11 Distribution of photos collected over 1 hour period.

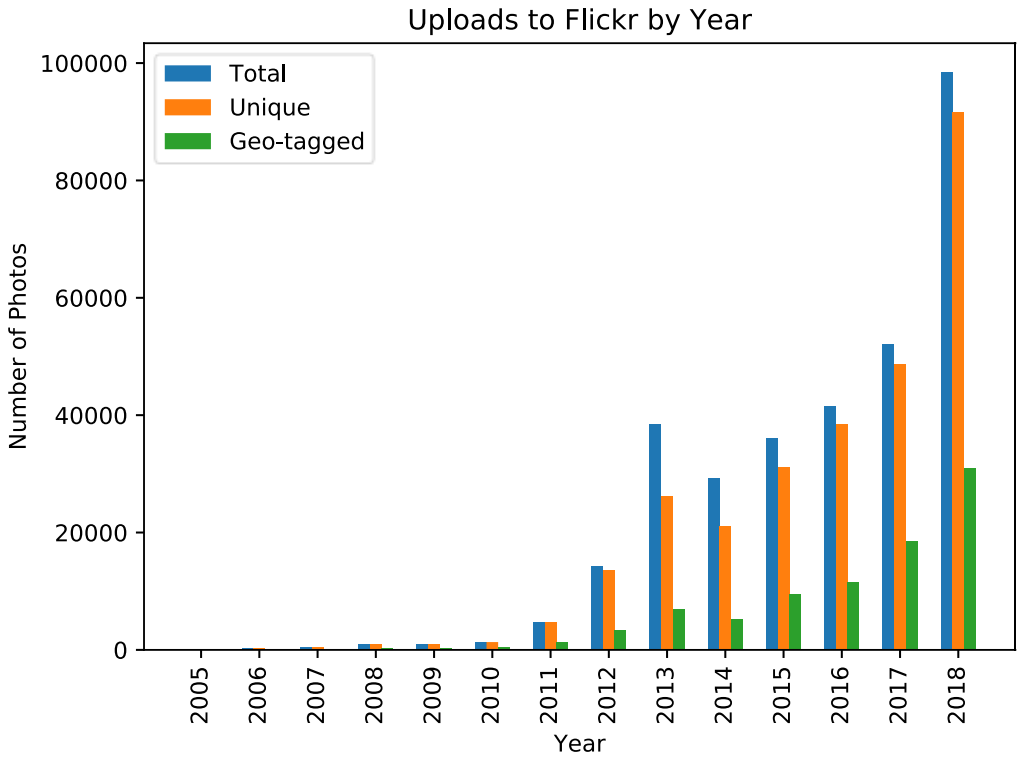

Figure 12 Distribution of photos collected by year combined with unique and geo-tagged metrics.

References

[1] Tweepy, "Tweepy," Joshua Roesslein, [Online]. Available: https://tweepy.readthedocs.io/en/v3.5.0/. [Accessed 18 November 2018].

[2] sybrenstuvel, "Github sybrenstuvel flickerapi," [Online]. Available: https://github.com/sybrenstuvel/flickrapi. [Accessed 18 November 2018].

[3] MongoDB, "MongoDB," MongoDB, Inc., [Online]. Available: https://www.mongodb.com/. [Accessed 18 November 2018].

[4] MongoDB, "PyMongo," MongoDB, Inc., [Online]. Available: https://api.mongodb.com/python/current/. [Accessed 18 November 2018].

[5] "Matplotlib," [Online]. Available: https://matplotlib.org/. [Accessed 18 November 2018].

[6] "Pandas Pydata," [Online]. Available: https://pandas.pydata.org/. [Accessed 18 November 2018].

[7] "NumPy," [Online]. Available: http://www.numpy.org/. [Accessed 18 November 2018].

[8] "Scikit learn," [Online]. Available: https://scikit-learn.org/stable/. [Accessed 18 November 2018].

[9] Twitter, “Twitter Developer,” Twitter, Inc., [Online]. Available: https://developer.twitter.com/en/docs/tweets/sample-realtime/overview/GET_statuse_sample.html. [Accessed 17 November 2018].

[10] Twitter, “Twitter Developer,” Twitter, Inc., [Online]. Available: https://developer.twitter.com/en/docs/tweets/timelines/api-reference/get-statuses-user_timeline.html. [Accessed 18 November 2018].

[11] Twitter, “Twitter Developer,” Twitter, Inc., [Online]. Available: https://developer.twitter.com/en/docs/trends/trends-for-location/api-reference/get-trends-place.html. [Accessed 18 November 2018].

[12] Twitter, “Twitter Developer,” Twitter, Inc., [Online]. Available: https://developer.twitter.com/en/docs/tweets/search/api-reference/get-search-tweets.html. [Accessed 18 November 2018].

[13] R. Krikorian, “Twitter Engineering,” Twitter, Inc., 4 February 2010. [Online]. Available: https://blog.twitter.com/engineering/en_us/a/2010/woeids-in-twitters-trends.html. [Accessed 18 November 2018].

[14] Twitter, “Twitter Developer,” Twitter, Inc., [Online]. Available: https://developer.twitter.com/en/docs/tweets/filter-realtime/api-reference/post-statuses-filter.html. [Accessed 18 11 2018].

[15] Klokan Technologies, “Bounding Box Tool,” Klokan Technologies, [Online]. Available: https://boundingbox.klokantech.com/. [Accessed 18 November 2018].

[16] LatitudeLongitude.org, “Latitude Longitude Glasgow,” LatitudeLongitude.org, [Online]. Available: http://latitudelongitude.org/gb/glasgow/. [Accessed 18 November 2018].

[17] Twitter, “Twitter Developer,” Twitter, Inc., [Online]. Available: https://developer.twitter.com/en/docs/developer-utilities/rate-limit-status/api-reference/get-application-rate_limit_status.html. [Accessed 19 November 2018].

[18] Flickr, “Flickr,” Flickr, Inc., [Online]. Available: https://www.flickr.com. [Accessed 18 November 2018].

[19] Flickr, “Flickr API,” Flickr, Inc., [Online]. Available: https://www.flickr.com/services/api/flickr.panda.getPhotos.html. [Accessed 18 November 2018].

[20] S. Judson, “Flickr API Discussions,” Flickr, Inc., [Online]. Available: https://www.flickr.com/groups/51035612836@N01/discuss/72157652283122791/. [Accessed 18 November 2018].

[21] Flickr, “Flickr App Garden,” Flickr, Inc., [Online]. Available: https://www.flickr.com/services/api/flickr.photos.getRecent.html. [Accessed 18 November 2018].

[22] Flickr, “Flickr App Garden,” Flickr, Inc., [Online]. Available: https://www.flickr.com/services/api/flickr.tags.getHotList.html. [Accessed 18 November 2018].

[23] Flickr, “Flickr App Garden,” Flickr, Inc., [Online]. Available: https://www.flickr.com/services/api/flickr.photos.search.html. [Accessed 18 November 2018].

[24] Flickr, “Flickr App Garden,” Flickr, Inc., [Online]. Available: https://www.flickr.com/services/api/flickr.people.getPublicPhotos.html. [Accessed 18 November 2018].

[25] Flickr, “Flickr Developer Guide: API,” Flickr, Inc., [Online]. Available: https://www.flickr.com/services/developer/api/. [Accessed 18 November 2018].

[26] d2kagw, “Stack Overflow,” Stack Exchange Inc., [Online]. Available: https://stackoverflow.com/a/1996378. [Accessed 18 November 2018].