Web Science Report

Executive Summary

The project is based on the company -- Transport for London (TfL), aiming to create an alert system to explore the useful tweets about the London's buses, London Underground, Docklands Light Railway, London Overground and London Trams. The system is supposed to extract enough data to analysis transport related problems.

Task 1 is dividing the given region into 1km*1km grids, and analysis the distribution of the location of the given geo-tagged tweets in the geoLondonJan file. The tweets concentrated on a small grid, and the further away from that grid, the less tweets it contains. The part I gives the answer of Task 1.

Task 2 is to create a newsworthy scoring method to classify the tweets based on the given high-quality file(highFileFeb) and low-quality file(lowFileFeb). The method of calculating the score of tweets contains a hyper-parameter – threshold. Hence, the high-quality file and low-quality file are divided into training dataset and testing dataset. Using Recall and F1 score to analysis the quality of the model with different threshold values. The part II is the answer of Task 2.

Task 3 is to apply the methods from Task 1 and Task 2, and focus on analyzing the high-quality tweets with the use of the model trained in Task 2 (threshold = 1.52). The part III presents the details of Task 3.

Part I. Location Grids

UML and Sequence Diagram

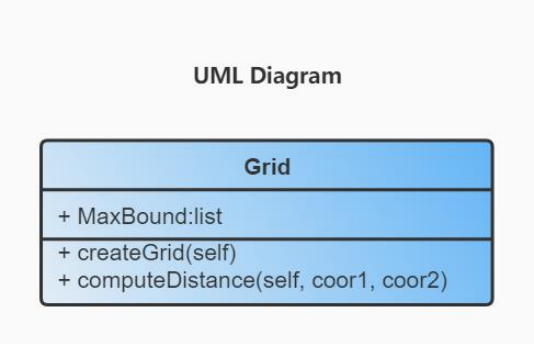

Figure 1: the UML Diagram of Task1

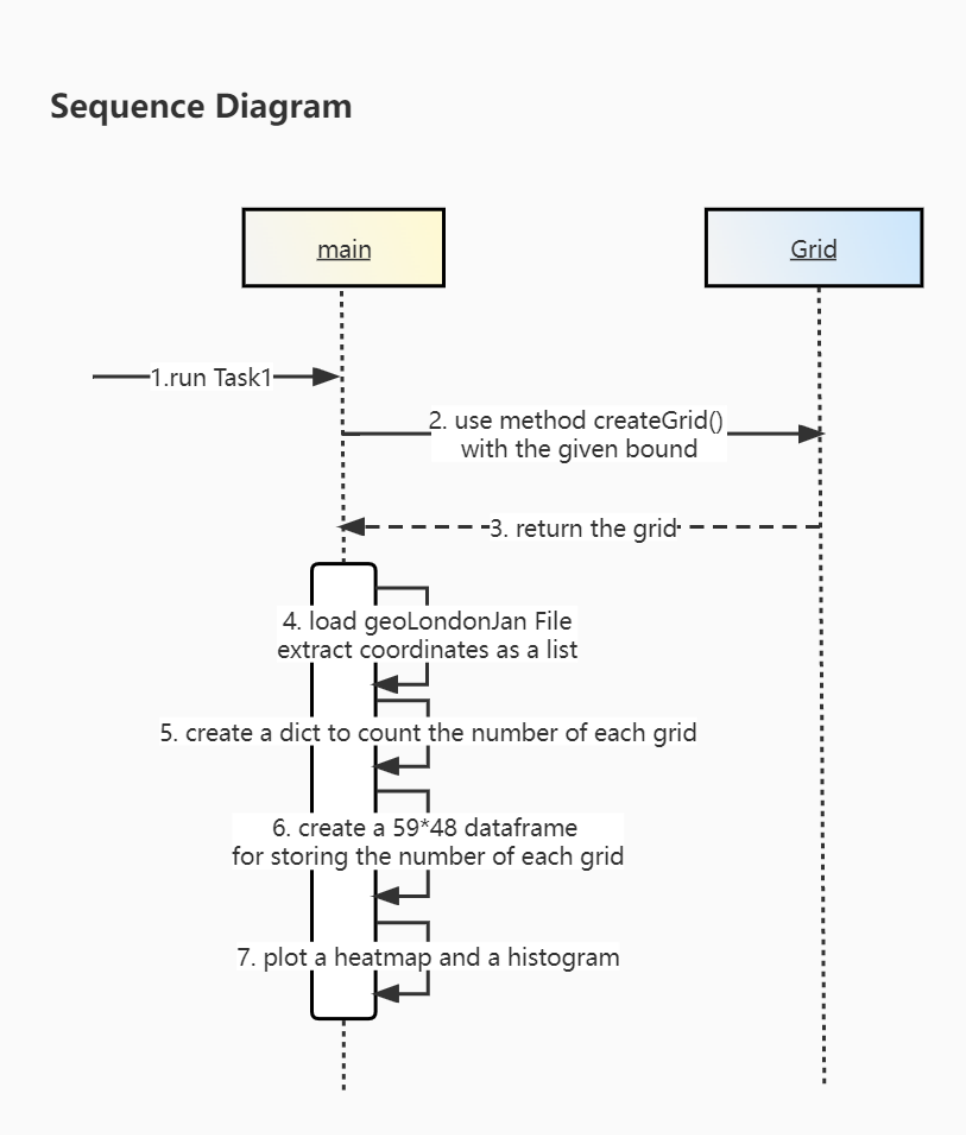

Figure 2: the Sequence Diagram of Task1

Algorithm Description

Task 1 contains one class – Grid. The class Grid contains one attribute – MaxBound. And it includes two methods – createGrid() and computeDistance(coor1, coor2).

Pseudo-code

The pseudo-code of method in class Grid:

Algorithm: computeDistance(coor1, coor2)
Input: two coordinates -- coor1(list), coor2(list) Output: the distances(d) between the two coordinates in meters with the use of Haversine Formula
1. lat1 ← coor1[0]; long1 ← coor1[1]; lat2 ← coor2[0]; long2 ← coor2[1] 2. R ← 6371.0 // the earth's mean radius in metres 3. Phi1 ← lat1 * (π / 180); phi2 ← lat2 * (π / 180)

4. delta1 ← (lat1 - lat2) * (π / 180)
5. delta2 ← (long1 - long2) * (π / 180)
6. a ← $sin(delta1 \ / \ 2)^2$ + cos(phi1) * cos(phi2) *$sin(delta2 \ / \ 2)^2$
7. c ← 2 * arctan($\sqrt{\frac{a}{1-a}}$)
8. d ← R * c // the distance between the two coordinates in meters
9. return d

Algorithm: createGrid(Maxbound)

Input: The boundary coordinates list (Maxbound): [topleft, bottomleft, topright, bottomright]
Output:
1) colPoints(list): the list of x-coordinates of the grid
2) rowPoints(list): the list of y-coordinates of the grid
3) latOffset(float): the difference of latitude between each x-coordinates
4) lonOffset (float): the difference of longitude between each y-coordinates

1. Initialize Maxbound ← [topleft, bottomleft, topright, bottomright]
2. rows←⌈max(ComputeDistance(topleft, bottomleft), ComputeDistance(topright, bottomright))⌉
3. columns←⌈max(ComputeDistance(topleft, topright), ComputeDistance(bottomleft, bottomright))⌉
4. numofGrids ← rows * columns; print numofGrids // the number of grids
5. lonOffset ← (longitude of left boundary- longitude of right boundary)/ columns
6. latOffset ← (latitude of top boundary- latitude of bottom boundary)/ rows
7. initialize rowPoints and colPoints as empty lists
8. for i = 0 to rows do
9. point ← 51.261318 + i* latOffset
10. add point to rowPoints
11. end for
12. for j = 0 to columns do
13. point ← -0.563 + j* lonOffset
14. add point to colPoints
15. end for
16. return colPoints, rowPoints, latOffset, lonOffset

There are two important points in Algorithm createGrid to explain:

1. The second step is to calculate the distance of topleft corner and bottomleft corner of the boundary, and the topright corner and bottomright corner of the boundary, choosing the maximum distance and round to the upper integer as the number of rows.

2. The third step is to calculate the distance of topleft corner and topright corner of the boundary, and the bottomleft corner and bottomright corner of the boundary, choosing the maximum distance and round to the upper integer as the number of columns.

Statistics and Distributions

The given coordinate bound of London -- [-0.563, 51.261318, 0.28036, 51.686031] was divided

into 2832 grids, with 48 rows and 59 columns.

The distribution of the tweets locations are shown in Figure 3 and Figure 4.

According to the two heatmaps if Figure 3, the tweets locations are concentrated around the coordinate (31 30), which is corresponding to the latitude and longitude -- (51.536, -0.134), containing 1349 tweets. Compare the two heatmaps, we can see most of the locations contain tweets less than 50. One of the reasons of this phenomenon is these coordinates with large number of tweets must have some famous scenic spot so lots of people will share their location when travelling.

According to the histogram in Figure 4, the number of tweets in most grids is smaller than 100, while only no more than 10 grids contain number of tweets which is larger than 100.

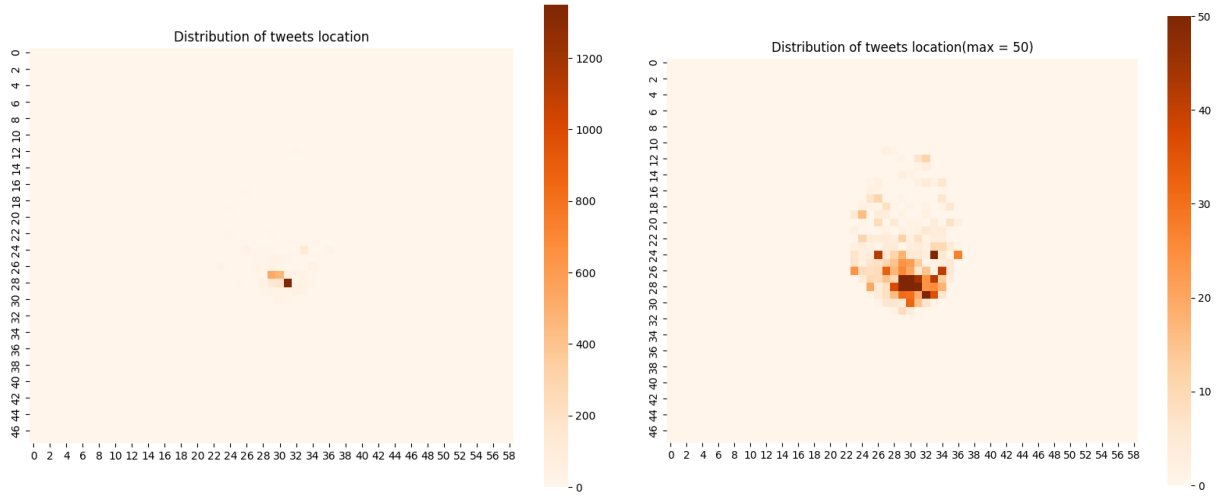

Figure 3: The distribution of tweets location

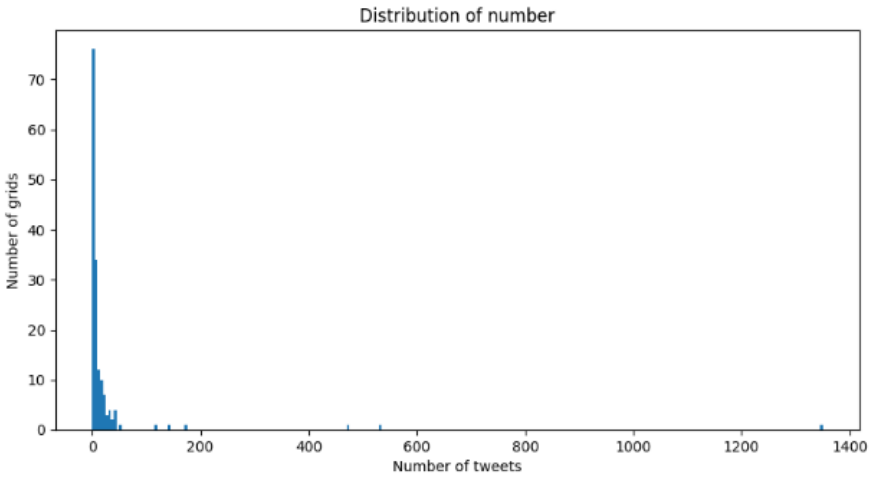

Figure 4: The Distribution of number of grids by number of tweets

Part II. Newsworthy Scoring

UML and Sequence Diagram

The figure 5 shows the major process of task 2. The figure 6 presents the classes and their attributes and methods in Task 2.

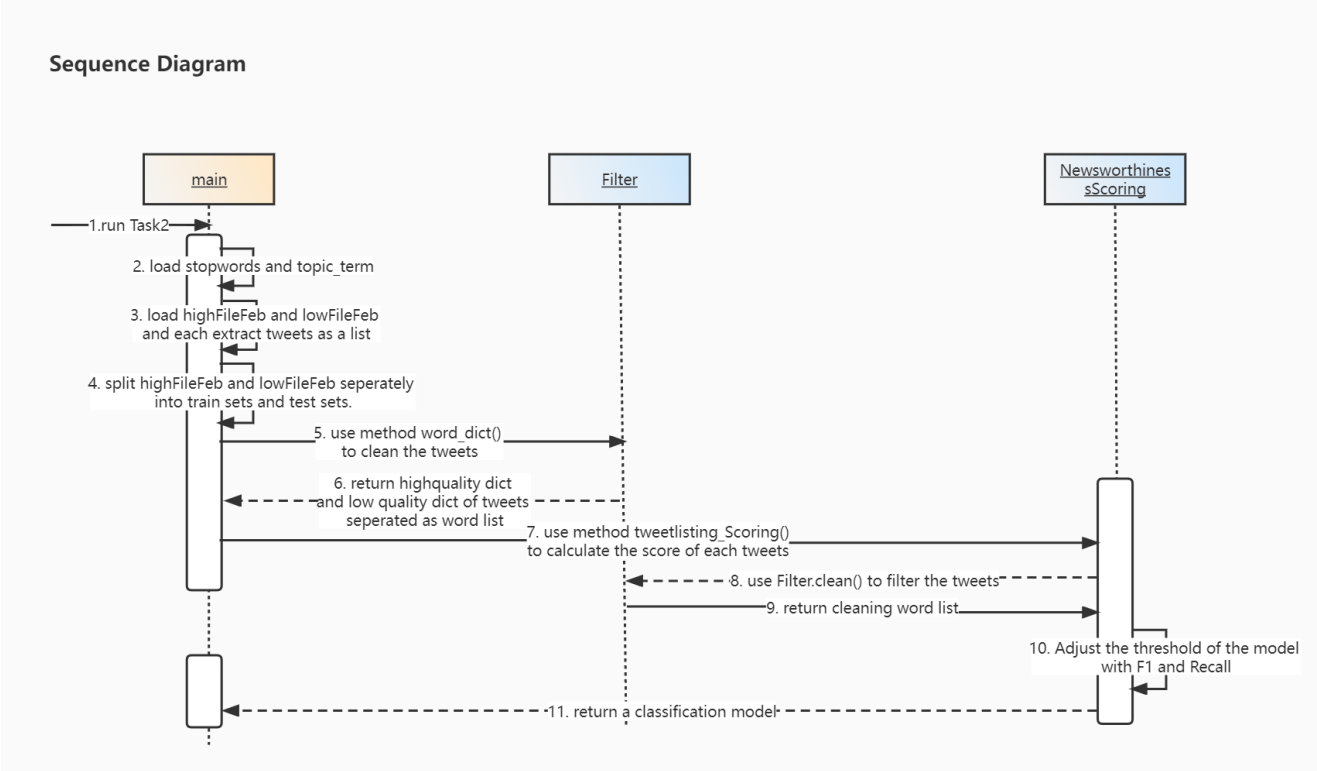

Figure 5: Sequence Diagram of Task 2

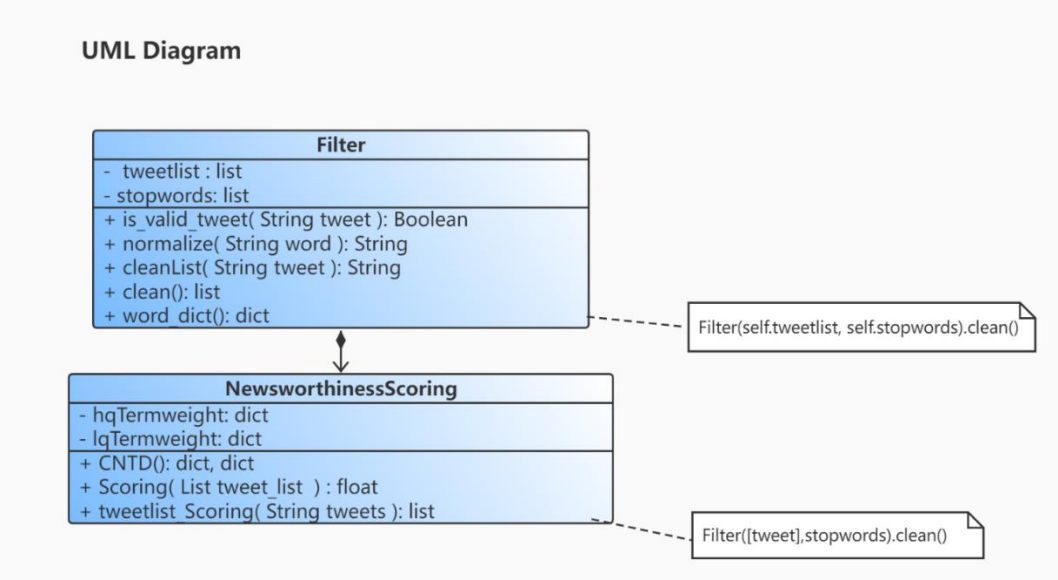

Figure 6: UML Diagram of Task 2

Algorithm Description

Task 2 section contains two class – Filter and NewsworthinessScoring.

1. Filter

a) Attributes:

- Stopwords: a stop words list with the less newsworthy words.
- Tweetlist: a list with each element is a tweet string.

b) Methods:

- is_valid_tweet(): Used to check whether the word in tweets is in stop words or equals to "&". When the words meet the conditions, the method will return False. The method is to identified the useless words.
- normalize(word): To remove symbols from the tweets and set the whole tweets as lower case.
- cleanList(tweet): remove the emoji then encode and decode the tweets.
- clean(): cut the tweets in the whole document into words and clean the words with the use of cleanList(), normalize() and is_valid_tweet(), finally return a list of words.
- word_dict(): use method clean() to clean the list of tweets and store the frequency of words into a dictionary.

2. NewsworthinessScoring

a) Attributes:

- hqTermweight: the dictionary of word frequency in highFileFeb
- lqTermweight: the dictionary of word frequency in lowFileFeb

b) Methods:

- CNTD(): calculate the score of each terms based on equation(1)-(3) with the use of the hqTermweight and lqTermweight. The threshold below is a hyper-parameter which need to be manually selected. The method finally return a word list score in high quality file and a word list score in low quality file.

$$\begin{cases} R_{HQ}(t) = \frac{P(\frac{t}{HQ})}{P(\frac{t}{BG})} = \frac{\frac{tf_{t,HQ}}{F_{HQ}}}{\frac{tf_{t,BG}}{F_{BG}}} \\ R_{LQ}(t) = \frac{P(\frac{t}{LQ})}{P(\frac{t}{BG})} = \frac{\frac{tf_{t,LQ}}{F_{LQ}}}{\frac{tf_{t,BG}}{F_{BG}}} \end{cases} \quad (1)$$

$$S_{HQ}(t) = \begin{cases} R_{HQ}(t), if\ R_{HQ}(t) \geq threshold \\ 0, otherwise \end{cases}. \quad (2)$$

$$S_{LQ}(t) = \begin{cases} R_{LQ}(t), if\ R_{LQ}(t) \leq threshold \\ 0, otherwise \end{cases}. \quad (3)$$

- Scoring(): calculate the score of each tweets which was cleaned and separated to a list based on the equation(4). Finally return a number of tweet score.

$$N_d = log_2 \frac{1+\sum_{t \in d} S_{HQ}(t)}{1+\sum_{t \in d} S_{LQ}(t)}. \quad (4)$$

- tweetlist_Scoring(): calculate each tweets in a tweet list calling the method Scoring(). Finally return a list of score corresponds to each tweet in the tweet list.

Pseudo-code

In this section, I will introduce the major algorithms in the task 2.

The major algorithms in class Filter:

Algorithm: normalize (word)
Input: the word in a tweet: word (String) Output: the word without symbols: s(String)
1. if the first character of word is “#”, “@”, “$” then 2. word ← word[1:] // remove the first character from the word 3. if the word is start with “T&amp” or “http” then 4. Return None 5. exclude ← String.punctuation // all the symbols 6. remove the characters in the exclude 7. return word

Algorithm: clean ()
Input: ● tweets stored in a list: tweetlist (list) ● stop words stored in a list: stopwords (list) Output: the tweet list stored separated words of the whole document: tweet_list (list)
1. initialize tweet_list as an empty list 2. filter ← Filter(self.tweetlist, self.stopwords) // create a Filter object 3. for tweet in tweetlist do

```
4.      tweet ← filter.cleanList(tweet) // clean each tweet using cleanList () method
5.      words ← cut the tweet into words
6.      Initialize word_list as an empty list // store the normalized word of each tweets
7.      For word in words do
8.          If word is not None and word is valid then
9.              word ←filter.normalize(word)
10.                 If the length of the word is larger than 1 then
11.                     Add word into word_list
12.     end for
13.     extend word_list to tweet_list
14. end for
15. return tweet_list
```

Algorithm: word_dict ()

Input:

- tweets stored in a list: tweetlist (list)
- stop words stored in a list: stopwords (list)

Output: the tweet dictionary stored the frequency of each words of the whole document: word_dict (dict)

```
1. initialize word_dict as an empty dict
2. tweet_list ← Filter(self.tweetlist, self.stopwords).clean()
3. for key in tweet_list do
4.      count the word in tweet_list and add into word_dict
5. End for
6. Return word_dict
```

The major algorithms in class NewsworthinessScoring:

Algorithm: CNTD ()

Input:

- the dictionary of word frequency in high quality model: hqTermweight (dict)
- the dictionary of word frequency in low quality model: lqTermweight (dict)
- the threshold of scoring: threshold (float)

Output: two lists stored the score of each tweet in high quality model and low quality model separately: S_hq (list), S_lq (list)

```
f_h ← the total number of words in hqTermweight
f_l ← the total number of words in lqTermweight
f_bg ← the total number of words in lqTermweight and lqTermweight
bgweight ← the merge dictionary of lqTermweight and lqTermweight
Initialize R_hq and R_lq as two empty dictionaries
for key, value in hqTermweight do
    tf_h ← value
    tf_bg ← bgweight.get(key, 0)
    R_hq[key] ← (tf_h * f_bg) / (tf_bg * f_h)
End for
for key, value in lqTermweight do
    tf_l ← value
    tf_bg ← bgweight.get(key, 0)
    R_lq[key] ← (tf_l * f_bg) / (tf_bg * f_l)
End for

Initialize S_hq and S_lq as two empty dictionaries
for key, value in R_hq do
    if value >= threshold
        then S_hq[key] = value
        else S_hq[key] = 0
End for
for key, value in R_lq do
    if value >= threshold
        then S_lq[key] = value
```

```
26.             else S_lq[key] = 0
27.  End for
28.  Return S_hq, S_lq
```

Algorithm: Scoring (tweet_list)

Input: words of a tweet stored in a list: tweet_list (list)

Output: score of the tweet: N (float)

```
 7.  initialize S_hq, S_lq ←NewsworthinessScoring.CNTD(self)
 8.  initialize S1, S2 as 0
 9.  for key in tweet_list do
10.      S1 ← S_hq.get(key, 0) + S1
11.      S2 ← S_lq.get(key, 0) + S2
12. End for
13. N ← math.log2((1+S1) / (1+S2))    // calculate the score with the equation(4)
14. return N
```

Algorithm: tweetlist_Scoring (tweets)

Input: a list of tweets: tweet_list (list)

Output: a list of scores correspond to each tweet in the tweets: n (list)

```
1.  initialize n as an empty list
2.  initialize model ← NewsworthinessScoring(self.hqTermweight, self.lqTermweight)
3.  for tweet in tweets do
4.      tweet_list ← clean the words in tweets and unique the word of each tweet into a list
5.      n.append(model.Scoring(tweet_list)) // scoring each tweet_list and add into n
6.  End for
7.  return n
```

Model Analysis

Stop Words and Major Terms

The stop words are loaded from an additional stopwords.txt[1]. Loading the major_term.txt with some import noun phrase which can be cut in splitting the tweets. The phrases include “transport for london”, “tfl”, “london's buses”, “london underground”, “docklands light railway”, “london overground”, “london trams”, “rail way”.

Threshold Selection

Choosing an appropriate threshold from 1.5 to 2 with an interval of 0.02. With the aim of extracting more useful information from the tweets, I prefer a high F1 score and a high Recall model.

First of all, I split the high-quality tweets from highFileFeb and low-quality tweets in lowFileFeb into training datasets and testing datasets. According to the equation (5)-(7), I calculate the Recall, Precision and F1 of the different threshold values. The explanation of variables in the equation (5)-(7) are as follows:

- TP: the number of tweets having score larger than 0 in the high-quality file;
- TN: the number of tweets having score smaller than 0 in the low-quality file;
- FN: the number of tweets having score equal to or smaller than 0 in high-quality file;
- FP: the number of tweets having score equal to or larger than 0 in low-quality file.

$$Recall = \frac{TP}{TP + FN} \tag{5}$$

$$Precision = \frac{TP}{TP + FP} \tag{6}$$

$$F1 = 2\left(\frac{Recall \times Precision}{Recall + Precision}\right) \tag{7}$$

Secondly, using the training datasets to fit the model with the different thresholds, and then I choose top 3 Recall models in the top 5 F1 score models. The values are shown in the table 1. The models of threshold equal to 1.5 and 1.52 have the same Recall (0.9967) and F1 score (0.6637), which are the highest two.

Table 1: the top 3 Recall models in the top 5 F1 score models (train set)

threshold	Recall	F1
1.5	0.9967	0.6637
1.52	0.9967	0.6637
1.54	0.9950	0.6633

Thirdly, using the testing datasets to fit the model with the three thresholds in the table 1, and the values of Recall and F1 score are shown in the table 2. The models with 1.52 and 1.54 threshold have the same Recall and F1 score.

Table 2: the Recall and F1 score of the table 1 models (test set)

threshold	Recall	F1
1.5	0.9950	0.6639
1.52	0.9975	0.6644
1.54	0.9975	0.6644

Hence, I choose the model with the highest Recall and F1 score in training set as well as in testing set(threshold = 1.52).

Word Cloud

The figure 7 and figure 8 shows the frequency of terms fitting the 1.52 threshold model in the whole high-quality file (highFileFeb) and the whole low-quality file (lowFileFeb) separately with the use of word cloud.

The figure 7 shows that the words with high frequency in high-quality content are related to the politics and the UK government, such as “boris”, “johnson”, and “pmqs”. The figure 8 presents that the words with high frequency in low-quality content are majorly associated with the economics and daily news, such as “bitcoin”, “earn” and “skynews”.

Figure 7: High quality word cloud

Figure 8: Low quality word cloud

Part III. Method Application

Sequence Diagram

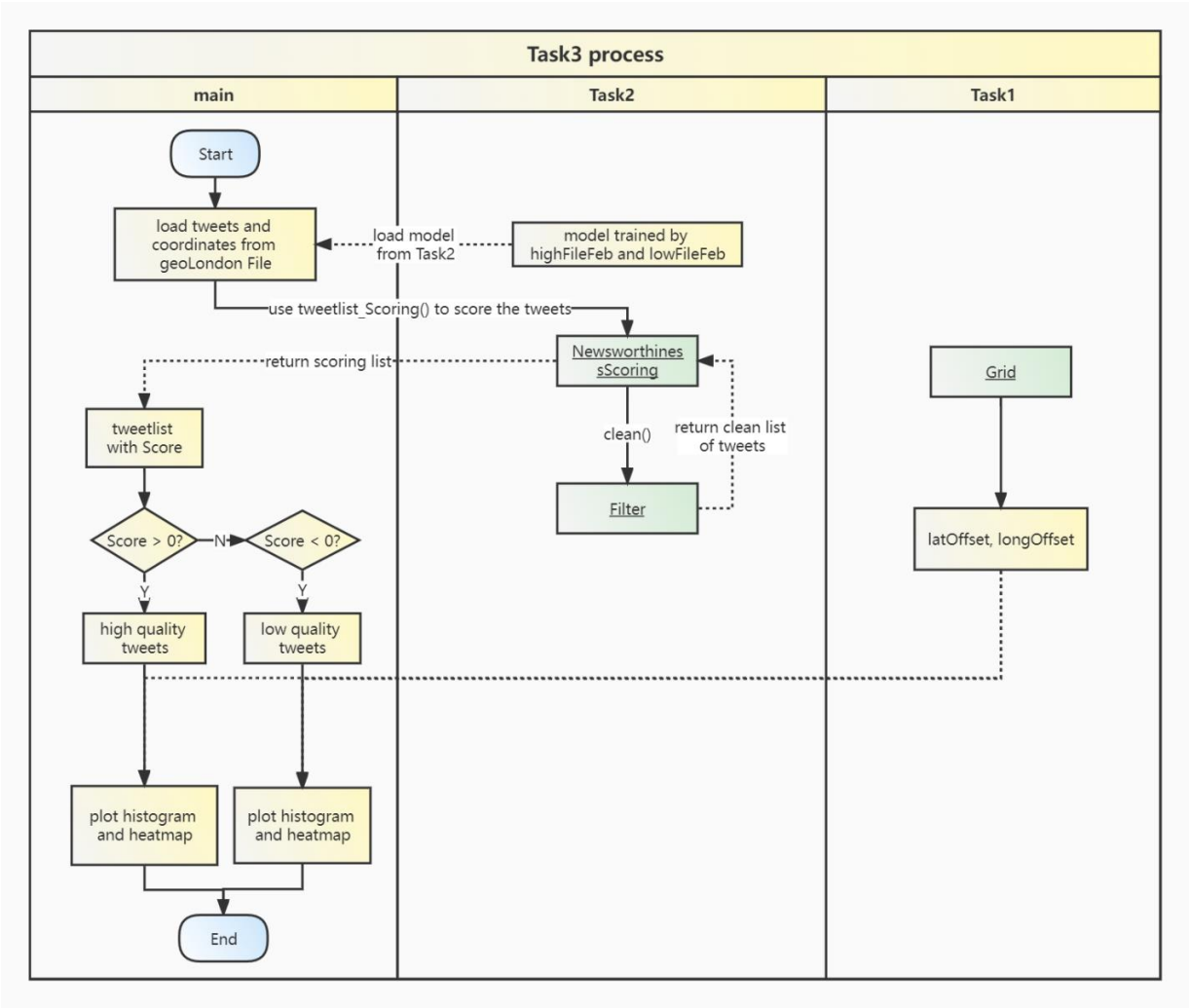

Figure 9: Sequence Diagram of Task 3

Method Application

Applying the model trained in Task 2 (threshold = 1.52) and the Grid method in Task 1 into the tweets in geoLondonJan file. The total number of tweets is 4142. After applying the method to the tweets, tweets are divided into of 2553 low-quality tweets, which have the score lower than 0 and more than twice the number of high-quality tweets (1123), scoring higher than 0. The number of no clear related tweets is 466, with 0 scores.

Statistics and Distribution

The distribution of locations of high-quality tweets are presented in Figure 10, with setting the max range of the left one and no setting range in the right one. And the distribution of locations of low-quality tweets are presented in Figure 11, with setting the max range of the left one and no setting range in the right one. The figure 12 and 13 present how many grids contain the same range of tweet numbers in high-quality file and low-quality file separately.

Comparing the graphs in Figure 10 and Figure 11, the tweets locations of high-quality and low-quality has the similar distribution, both concentrating on the coordinate (31, 30), corresponding to the latitude and longitude (51.536, -0.134). When the location is far away from the (51.536, -0.134), the number of tweets becomes smaller, no matter high-quality tweets or low-quality tweets. The number of high-quality tweets in the (51.536, -0.134) is above 350, while that of low-quality tweets is above 800. From the results we can surmise the location around (51.536, -0.134) may have lots of famous place that people prefer sharing locations when travelling there. According to Google map[2], the place about the coordinate surrounded by many schools and companies, such as Royal veterinary college and LBIC. And nearby the place, there are plenty of restaurants and famous tourist spots such as the Harry Potter Shop at Platform $9_{3/4}$. Hence most people travelling around the place prefer sharing the location and some travelling experiences in tweets.

Looking at the figure 12 and 13, most of grids contain the tweet number no more than 30 of both high-quality content and low-quality content. Only several grids which is presented and figure 10 and 11 contain most of the tweets.

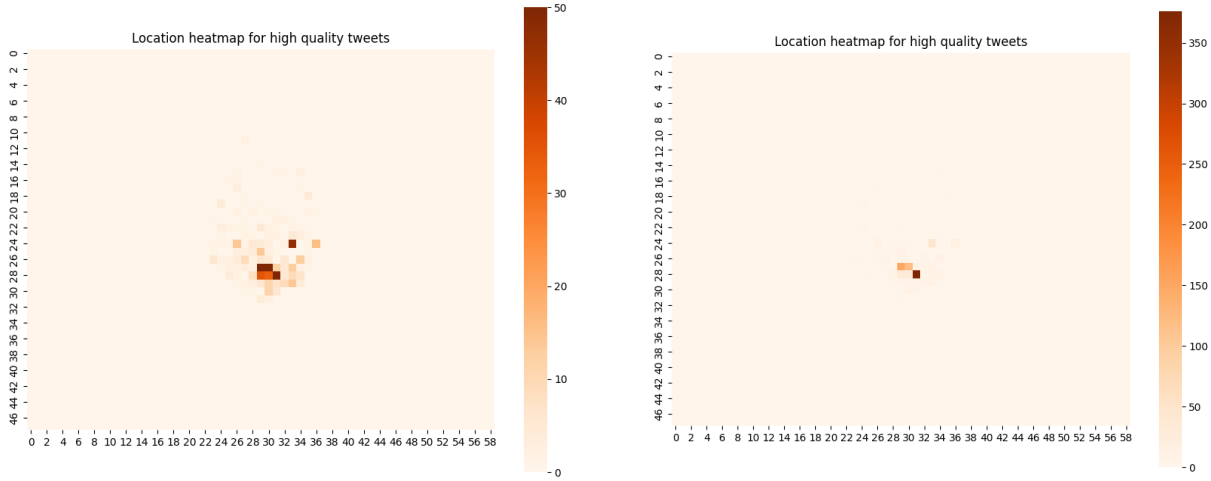

Figure 10: The distribution of high-quality tweets location

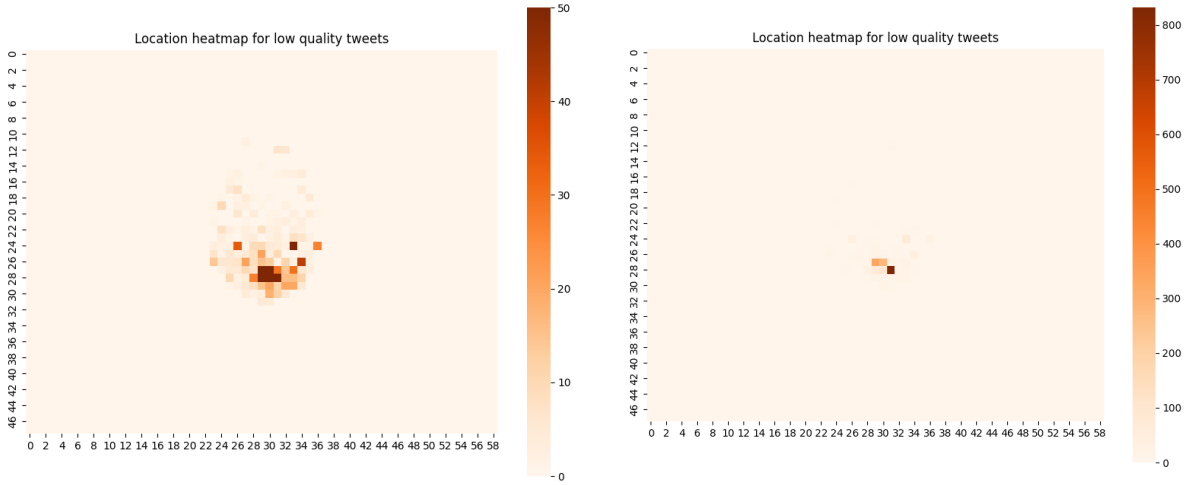

Figure 11: The distribution of low-quality tweets location

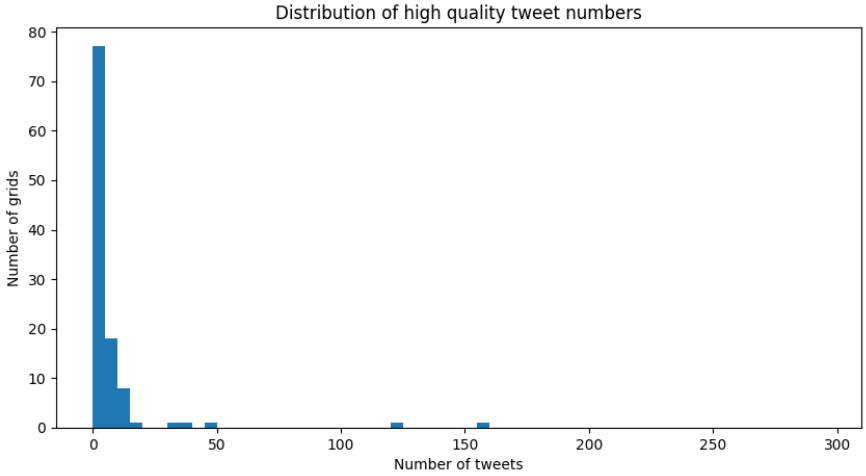

Figure 12: The Distribution of number of grids by number of high-quality tweets

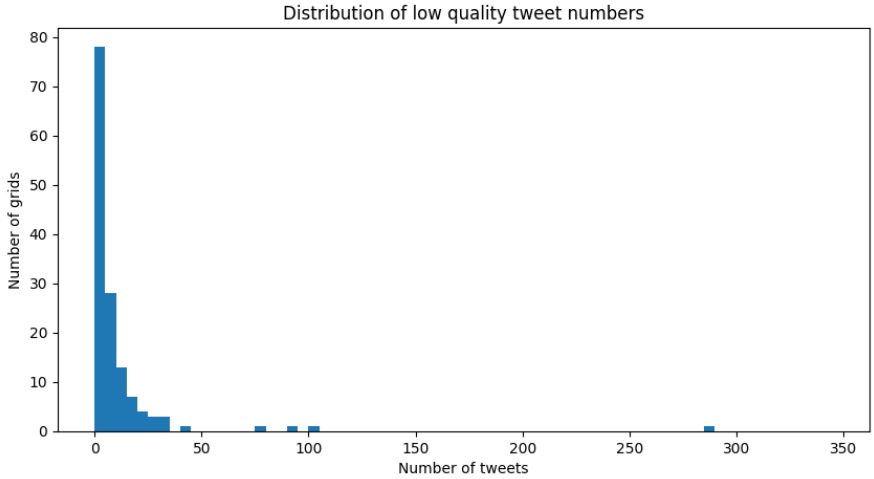

Figure 13: The Distribution of number of grids by number of low-quality tweets

Word Cloud

The figure 14 and figure 15 shows the frequency of terms in the high-quality tweets and the high-quality tweet.

Both of the high-quality and the low-quality tweets mentioning “London”, “kingdom” and “united” a great number of times. We can see that the high-quality content contains some useful words such as “tfl”, “road” and “station” in the figure 14. Looking at the figure 15, the low-quality tweets concentrate on the words such as “photo”, “drinking”, “posted” and so on.

Figure 14: High-quality content word cloud

Figure 15: Low-quality content word cloud

Conclusions

In conclusion, the distribution of high-quality tweets and low-quality tweets are similar, concentrating in the coordinate (51.536, -0.134) and getting fewer tweets when far away from the location. Using the threshold equal to 1.52 trained in task 2 and splitting the tweets into high-quality content and low-quality content. The number of tweets of high-quality content is less than half of the number of low-quality tweets. The most frequent words showing in the high-quality tweets and low-quality tweets are similar, containing “London”, “United” and so on. But the high-quality contents include some useful information although not obvious shown in the word cloud, for example, “tfl”, “road” and “bus”.

Bibliography

[1] “Stopwords-ISO/Stopwords-En: English Stopwords Collection.” Edited by Genediazjr, GitHub, 10 Oct. 2016, https://github.com/stopwords-iso/stopwords-en.

[2] “ 51°32'09.6‘N 0°08'02.4‘W.” Google Maps, Google, https://www.google.com/maps/place/51%C2%B032'09.6%22N+0%C2%B008'02.4%22W/@51.5360033,- .1361887,17z/data=!3m1!4b1!4m5!3m4!1s0x0:0x7b7b05299fde1dfc!8m2!3d51.536!4d-0.134.