# Crawler Side

## 1 Description

The crawler part in this project is in charge of obtaining data from various data sources—to be specific—Twitter and AURIN APIs. In order to support the research topic sound and well, the crawler itself should be able to retrieve data within a certain range of specified keywords and location coordinates, which guarantee a higher effectiveness of data usage and sentimental analysis afterwards. In addition, it is critical to accumulate a large amount of raw data, which lessens or eliminates the possibilities towards wrong conclusions during analysing process, and contributes the final goal of this project.

## 2 Crawling Tweets

Tweeter is researcher-friendly and provides a number of APIs for retrieving tweets from its database. But still there are lots of problems and challenges having been met during the crawler’s development.

### 2.1 Challenges & Solutions

#### 2.1.1 Outdated Documents

Although the Twitter gives out a range of APIs to researchers, the Documents of them, largely, do not contain the newest features about. Thus for the usage of Twitter APIs in the crawler side in the early stage, only the most basic and simple functions are put into, which only provides poor efficiency in both data retrieving and possibility of analysing afterwards.

To solve the challenge above, a great range of projects on GitHub and information in various forum have been viewed, for extra information of Twitter APIs new features. And finally, one of the API: *tweepy.api.cursor()* has been found, which covers the most of functions of previous attempts, enhancing the fetching performance to a great extent at the same time.

#### 2.1.2 Limitation of Data Requests

To constraint the total data throughputs, Twitter limits the rate of requests invoked by API users—180 times per 15 minutes (1 request every 5s) by default, which makes the process of accumulating tweets problematic.

Solution to this problem should lie in not only breaking the limitation of requests rate, but also improving the percentage of valid tweets retrieved in each reply from Twitter.

Possible solutions to these issues are:

* 1. Get tweets as many as possible within one request invoked in crawler and sent to Twitter.
  2. Describe the searching request to Twitter as explicit as possible, to ensure the majority of tweets retrieved in each request are valid and ready to be used.
  3. Register more Twitter authentication keys that allows sending tweets requests simultaneously to Twitter server, which would multiply the efficiency of data retrieving process.

In this project, altogether 3 Twitter Application keys are applied and used, and with the help of *tweepy.api.cursor()*, which provides sufficient arguments for describing each query request in detail and will package up to 500 tweets in each reply, in total the tweets crawler would output a throughput nearly 100 valid tweets per second on average.

The theoretical final throughput should be 1 req / 5s \* 500 tweets per req \*3 keys = 300 tweets/s, the duplicated tweets and data constraints lessen this number to just below 100, but still a great improvements when compared to 1- key crawling.

#### 2.1.3 Duplicated tweets

Duplicated tweets, apart from distinct tweets, would do harm to the sentimental analysis afterwards. To be specific, it would amplify the positive or negative emotions to the analysing process, which affects the final threshold of judgement. Therefore, there is a necessity for the removal of duplicated tweets.

Situation of duplication in tweets is common, especially when fewer constraints are defined while querying with Twitter APIs. On analysing the replies from Twitter APIs after each request, it is obvious to point out that there are two main types of duplication:

1. Duplication in both user *screenname* and message *text*
2. Duplication only in message *text*

For the cases above, (a) is regarded as the duplicated tweets that would do harm to the whole system and must be removed. To filter the tweets that contains same *screen\_name* and message *text*, a new *identifier* = *screen\_name + text* is defined for each tweet, which helps to fix the issue above. On defining this *identifier*, only the tweets with same *screen\_name* and *text* can be filtered out, but not the ones with same *screen\_name* only or *text* only.

At this stage, it is not a sound filter method, but a primary one, to give the first treatment to duplicated tweets, and some more complicated filter methods is implemented in the database and analysis sides, which are relevant to context and sentimental level. The meaning of the primary filter is to leverage the analysis afterwards, because the duplication here is syntax-relevant and easy to handle, which saves calculation resources for following parts for other analysing needs.

#### 2.1.4 Search Exactly

In the beginning stage, the search could only base on the keyword—all the returned results are only based on the given keyword. With this searching strategy, there is no problem in the number of returned tweets, but the percentage of tweets’ validity—only a small range of tweets contains the coordinate, within which only a tiny group of tweets are inside Australia. Although it can provide tens of thousands tweets every half of a day, but the efficiency of each request is too low to accept, which is a waste of resources and give a large pressure to the database and relevant parts.

To solve this, *geocode* is found to be helpful, which defines an area by giving the coordinate of the centre of a circle—(latitude, longitude), and the radius of the searching circle, in *km* or *mi*. After this method, the replies of Twitter would give tweets that contain keyword and in exact circle of areas.

#### 2.1.5 Scalability

### 2.2 Trim Raw Data

## 3 Crawling AURIN

## 4 Write to CouchDB