# Distributed Tweets Harvesting

## Get data from Twitter API

In the implementation of this system, a python library called ‘Tweepy’ is adopted. It provides access to the entire twitter RESTful API and Streaming API.

### Authentication

As Twitter require all requests for data must be authenticated, only those who is authenticated via OAuth service can harvest Twitter data via Twitter API. To get authentication, the developer has to create Twitter application in Twitter apps website and get consumer key, consumer secret, access token and access token secret. In view of the rate limit for single application, several applications were created to improve the efficiency of harvesting and the access tokens of them are stored as JSON file. When the twitter harvesting program starts, this information is specified in the parameters of command line.

### RESTful API

RESTful API is a common application program interface that uses HTTP requests to GET, PUT, POST and DELETE data from the API provider. The Twitter RESTful API accepts parameters like query, geocode, language. In this scenarios, the geocode is made fully use of because the geographic information of the twitter is the main concern. The API returns tweets by users located within the particular geographic area, which is specified with given latitude, longitude and radius. However, the tweets returned are not always valuable. The preference of API is the location geotagged by the users, but it will fall back to the location information in users’ profile. Obviously, it is not direct relationship between users’ location and the position they post their tweets. Thus, the tweets that matter are only those provide coordinates position.

In utilization of Twitter RESTful API, pagination is exploited a lot to iterate through twitter messages, users’ timelines, etc. Performing pagination requires a page parameter with each requests, which leads a problem that a mass of code is needed to handle the pagination loop. Nevertheless, as Tweepy is adopted, the Cursor class enables pagination to be implemented easier with less code.

Yet, the RESTful API has its own restriction, as the area accepted geocode represents a circle area using latitude, longitude and radius. In a distributed Twitter harvesting program, it is challenging and counterintuitive to make arrangement so that there is no overlapping or omission among the searching areas of all nodes. Therefore, Streaming API is also crucial to develop an efficient and accurate Twitter harvesting system.

### Streaming API

Twitter Streaming API is provided to monitor tweets of particular users, tweets of particular topics or tweets in particular areas in real time. While RESTful API is used to pull data from twitter, the Streaming API pushes twitter message to a persistent session. It is supportive when obtaining a high volume of tweets or real-time feedback is needed.

The Tweepy library provides a stream listener class to receive messages from streaming session. In this system, a subclass inheriting from Tweepy StreamListener is created. The class override the on-data method, which enables listener to call functions to deal with received data. In addition, filter is available in Streaming API. In this case, the listener is set to listen the tweets posted in particular area. The parameters is four numbers(latitude and longitude) indicating the vertexes of a rectangle. To ensure efficiency and accuracy of the system, the monitoring areas of four nodes are designed carefully so there are no overlapping or omission.

### Parallel Tweets Harvesting

In this system, the multiple instances on Nectar Cloud work run the tweets harvesting program simultaneously. As shown in the figure below, each instance maintains a RESTful API program and a Streaming API program harvesting tweets posted on different areas at the same time. Like aforementioned, the harvesting areas are elaborately designed since any omission is not endurable. Afterwards, twitter data will be handled in several processes. These processes, which will be discussed in following sections, includes sentiment analysis, drunk detection, hashtag extraction. Subsequently, the processed data and raw twitter will be stored separately in the Couchdb.



## Get data from Other sources

To enhance the extensibility of the system, the ability of getting data from various sources are developed as well. In some scenarios, it is necessary to acquire data from other database using Curl command. To make this process more effortless, we implement a python program which request data using Curl command and process the returned JSON data, be followed by storing them into the database.

Also, the system allows data to be acquired from JSON file. As the Twitter JSON data is commonly a large file, it is assumed that the most time-consuming task is to load the JSON object from file considering the cost of reading file. To solve this issue, a rational solution is Single Instruction, Multiple Data Stream (SIMD) architecture. To obtain results more rapidly, the tasks are divided and run parallelly on multiple processes with different data. The adopted architecture is illustrated below, with some trivial details being omitted to make the figure more clear.



In this architecture, the main process calculate the starting position of processes’ reading. In this case, starting position refers to the number of bytes from the beginning to the position where the process should start reading the file. Afterward, it use MPI broadcast function to broadcast this information to all processes, including itself. Before reading the file, all processes invoke file.seek() function to locate the start position. While reading the lines from the JSON file, all process simultaneously match the Twitter data to the blocks in the grid. Once the position of the file reading pointer goes over the start position of other processes, the processes stop reading.

If this solution is to be taken, it is crucial to define an efficient way to confirm the start position of each process’s reading. Two approaches were taken in order to achieve a more effective solution.

In the first approach, the main process traversed the JSON file by lines to obtain the total number of lines as well as the starting position of each line. By simply dividing the total count of lines by number of processes, starting lines are assigned to each process. This approach seems to be logical since it fully make use of all processes to load the JSON data. However, it was shown that its performance is not ideal enough. Although the main process did not need to handle the raw text, it is still time-consuming to go through a gigabytes-size file. All other processes must wait until they receive the starting positions of lines from main process. In other words, the traversing caused a significant waste.

To enhance the performance, a better approach is to reduce the file traversing while keeping other parts of the code. As the size of JSON file can be straightforwardly accessed by the os package in python, the main process actually do not need the number of lines to averagely divide the file. Each process can get a ‘rough starting positions’ by dividing the file size by the number of processes. However, the reason why it is called ‘rough’ is that if the file is merely separated in this way, the starting positions will be more likely located at the other parts of lines rather than the beginning. To solve this, one possible way is to move the starting position to the beginning of the next line by invoking file.readline() function. Thus, the main process barely need to find the correct starting position for all processes instead of traversing the whole file. With the reduction of time spent on assigning different parts of file to processes, the program provide more ideal performance.

In JSON file reading program, since the data will be received more frequently than RESTful and Streaming API, updating database every time it get JSON will introduce significant delay, leading considerable reducing of performance. To deal with this challenge, batch updating is adopted as well.

## Error Handling

Because geographic information is represented as circular in RESTful API, the overlapping of areas is inevitable, which means different nodes on the cluster will probably get the same tweet data. In order to solve the problem of data duplication problem in database, we exploit the ID of raw tweet as the ID in CouchDB. When data is updated to CouchDB, the tweets with existing ID will be handled as a new ‘rev’ version of the original data.

In order to ensure the stability of service, Twitter have officially restricted the rate speed of harvesting tweets for single Twitter application. To cope with this restriction, we have set up waiting time for API invoking in addition to apply multiple applications for different nodes in the cluster. When the streaming listener receives 420 errors types (rate speed limit error), the program will automatically wait until the limit is lifted.

Another possible error is that the file format may not be well organized when data source is JSON file. As the JSON file is processed line by line, when the program encounters a file line that cannot be turned into JSON, it only abandons the line having wrong format and continue to read subsequent lines. So the error is handled with the minimum volume data lost.

# Twitter Analysis

For the tweets harvested, we first analyze whether it contains coordinate information and discard all the tweets without coordinate information. For the remaining tweets, we maintain two databases. One database is used to store the raw twitter data, which contains all the information of the tweets. Another database is used to store the data that analyzed and filtered, which contains only the suburb the tweet belongs to, tweet sentiment, whether it is sent under drunk and its hastags (if any). Retaining the raw data can provide us with possibilities to analysis more interesting scenarios in the future, while keeping the processed data can make map reduce more convenient and faster.

## Sentiment Analysis

Tweets sentiment analysis is to determine whether the sentiment of a tweet is positive, negative, or neutral. The method used for tweets sentiment analysis is Sentiment Analysis with Long Short Term Memory Units (LSTMs). This method is from “Perform sentiment analysis with LSTMs, using TensorFlow" (Deshpande, A., 2017). We do this task following O'Reilly tutorial and use his teaching sample code (O'Reilly, 2018).

LSTMs is a deep-learning-based method. We choose deep learning for several reasons: In the past, sentiment analysis requires a lot of domain knowledge, such as linguistic and psychological knowledge, which may need several years’ study. However, in recent years, deep learning has greatly reduced the difficulty of sentiment analysis. Instead of requiring a lot of domain knowledge, deep learning methods use general and understandable mathematical and statistical methods to process text, and then use the processed data to train the model. It greatly reduces the threshold of entry into this field and often performs well.

The data set we use is the Imdb movie review dataset. It has 25,000 labelled movie reviews, half of which have positive labels and half have negative labels. People tend to express emotions in movie reviews, so this dataset should be suitable for sentiment analysis. We divided this data set into 90% training data and 10% testing data.

The task then can be divided into 4 steps:

1)Build the word vector model and create id matrix for training data

2) Build RNN (with LSTMs)

3) Train the model

4) Compare performance with Textbolb and VADER Sentiment Analysis tools

1) Build the word vector model

The input for neural network can not be raw string, because some basic operations such as backpropagation or dot products cannot be performed on string. The input should be some scalar numbers or vectors or matrices of scalar numbers. In order to turn the raw text into the input for neural network, we need to create word embeddings, mapping words from the vocabulary to vectors of real numbers. “Word2Vec” model is useful is to this task. It creates word vectors by taking as its input a large corpus of text and produces a vector space. Word vectors are positioned in the vector space such that words that share similar contexts in the corpus are placed in close proximity to one another in the space (Mikolov, T., Chen, K., Corrado, G., & Dean, J., 2013). After process data though Word2Vec model, it will output an embedding matrix, which contains word vectors for every word in the training dataset.

For simplicity, we use GloVe pre-trained word vectors to generate a 400,000\*50 dimensional embedding matrix (Pennington, J., Socher, R., & Manning, C., 2014). Each row of the matrix is a word vector.

For the training set, we remove punctuation, parentheses, question marks, etc., and leaves only alphanumeric characters for each sentence. Then we use Tensorflow’s embedding lookup function to generate the vector representation for each sentence in the training set and create an id matrix containing these vectors as training input.

2) Build RNN (with LSTMs)

* Recurrent Neural Networks (RNNs)

The temporal information of the text is important when it comes to natural language processing, because each word in a text is very dependent on its context. In order to extract and use context information, we use RNNs instead of traditional feedforward neural network.

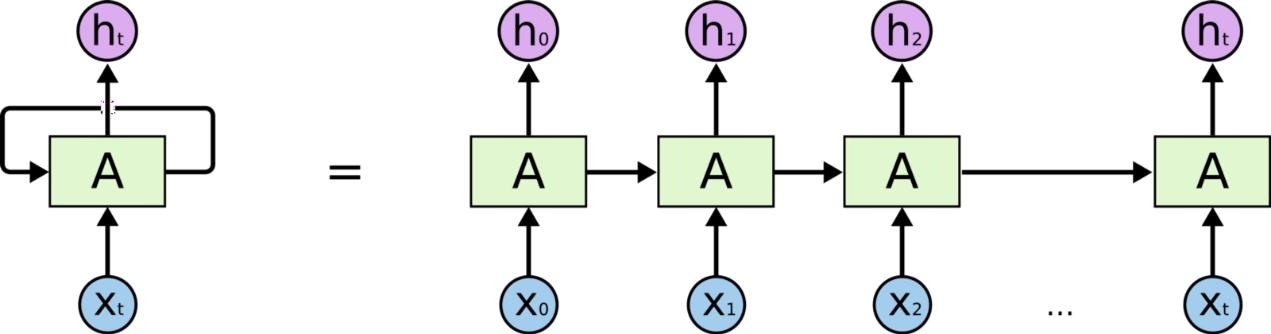
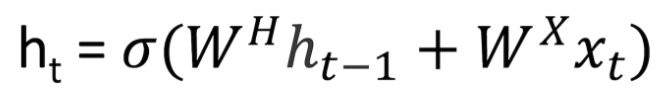


Figure-1. Sequential processing in RNN, from: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

Figure-1 is the sequential processing in RNN. represents for input word. Each is related to a time step t and each time step t is also corresponds to a hidden state . The hidden state contains the information from previous time steps. Hidden state is calculated by the following equation:



In the above equation, is the activation function. and represents the weight matrices. For all time steps, is the same, while varies for each input, so that the hidden state is affected for both current input and previous hidden state. These matrices are updated through backpropagation as time goes. Finally, a binary softmax classifier is used for the final hidden state and output values between 0 and 1, which represents the probabilities of positive and negative sentiment.

* Long Short Term Memory Units (LSTMs)

There is a problem in the traditional RNNs: when the gap between the relevant information become very large, RNNs are unable to learn to connect the information. In other words, the traditional RNNs performs bad in the long-term dependencies. To solve this problem, we add a long short term memory units into the previous RNNs.

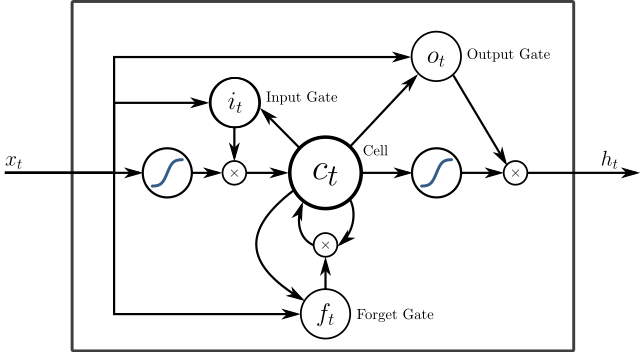


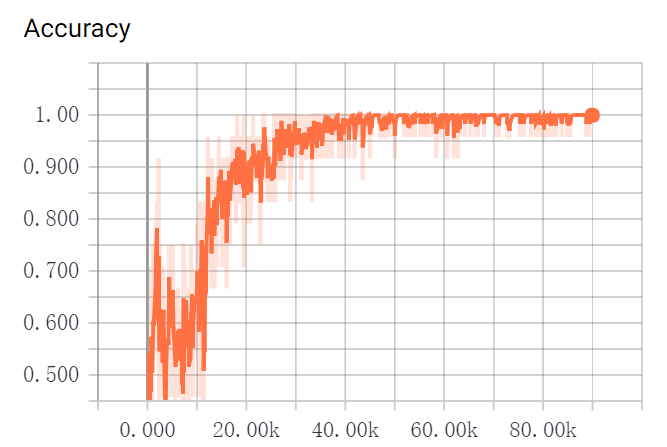
Figure-2. A peephole LSTM unit with input (i.e. i), output (i.e. o), and forget (i.e. f) gates, from <https://www.wikiwand.com/en/Long_short-term_memory>

As shown in Figure-2, instead of using a simple function discussed above to calculate hidden state vector , LSTM units use a more complex function to calculate . It introduces an input gate to decide how much should the model care about each input, a forget gate to throw away some information the model don’t needed, and an output gate to get input from the intermediate state and output the final hidden state.

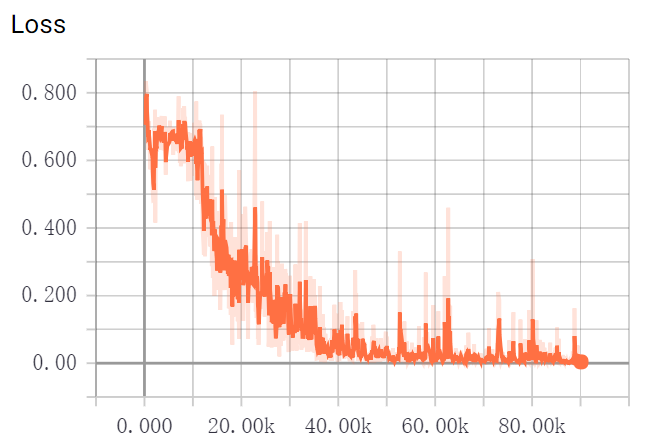
We firstly construct a LSTM cell with 64 units using tensorflow’s nn.rnn\_cell.BasicLSTMCell, then use a dropout wrapper to the LSTM cell to prevent overfitting. After that, we put both input data and the LSTM cell into the dynamic RNN then go through a dense layer to get the final output. The output contains 2 classes, positive or negative. We use standard cross entropy loss with a softmax layer for the final prediction then use Adam optimizer to update the neural network.

3) Train the model

We use Tensorboard to monitor the loss and accuracy. The following charts show the change of accuracy and loss over time. The model is run for 90,000 iterations and finally converged. However, there is possibility that the model overfits the training data.



Chat-3. The Accuracy of LSTMs model in different iterations



Chat-4. The Loss of LSTMs model in different iterations

4) Compare performance with Textbolb and VADER Sentiment Analysis tools

After training the model, we first test the performance of this model on the testing data. The accuracy is 87.5%. The result seems general acceptable. However, when it is applied to the twitter data, compared to other sentiment analysis APIs such as Textbolb and VADER Sentiment Analysis, our model performs not as well as expected.

VADER (Valence Aware Dictionary and sEntiment Reasoner) is a sentiment analysis tool. It is a lexicon and rule-based tool. It is specifically suitable for sentiments expressed in social media (Gilbert, C. H. E., 2014). It has 4 outputs for each sentiment analysis, positive value, negative value, neutral value and compound value. Compound value is a float in the range [-1.0, 1.0], where -1.0 is negative and 1.0 is positive. It combines the first three values and can be used as a total sentiment index.

TextBlob is a Python library for processing textual data. The text processed by TextBlob has a sentiment property, which can be used to sentiment analysis. The sentiment property returns a (polarity, subjectivity) tuple. The polarity is a float within [-1.0, 1.0] where -1.0 is negative and 1.0 is positive. The subjectivity is a float within [0.0, 1.0] where 0.0 is very objective and 1.0 is very subjective (Loria, S., Keen, P., Honnibal, M., Yankovsky, R., Karesh, D., & Dempsey, E., 2014).

The following examples reflects performance of our LSTMs model over the other two methods.

For the first two examples, three methods all performs well as expected. However, the remaining three examples show the problems of the LSTMs model and TextBlob.

Firstly, in the third example, the LSTMs model cannot tell neutral sentiment because itself is a binary classifier. In addition, as shown in the fourth and fifth examples, LSTMs model cannot tell the sentiment of the emoji in tweets. The reason is that when we process the training data, we only leave the tokens of words or numbers as training data. In this case, the model hadn’t been trained with any emoji data, so it cannot classify correctly the tweets containing a large number of emojis but only a small number of words. The Textbolb sentiment analysis method has the same problem. As shown in the fourth and fifth examples, Textbolb incorrectly classify the tweets to neutral sentiment. These tweets are special, in which words has neutral sentiment but there are still some emoji reflecting strong sentiments. This shows that Textbolb cannot deal with emoji well.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Tweet | LSTMs | Textblob | VADER |
| 1 | “Happy birthday! @Josh” | Positive | 1.000 | 0.6114 |
| 2 | “Beautiful Friday #smile” | Positive | 0.575 | 0.5994 |
| 3 | “It's just 10 days 'til #AllStarLanes #ShepherdsBush opens.” | Negative | 0.000 | 0.0000 |
| 4 | “😊😊😊 #Saturday” | Negative | 0.000 | 0.5614 |
| 5 | “Photoshop? 🙄😒” | Negative | 0.000 | -0.5423 |

Table-1 The example of sentiment analysis on tweets using LSTMs, Textblob and VADER sentiment analysis.

In order to ensure the accuracy of the sentiment analysis part of this task, VADER Sentiment Analysis model is finally chosen. The reason why VADER can deal with emoji well is that the VADER model incorporate numerous lexical features common to sentiment expression such as a full list of Western-style emoticons, so that it can extract sentiment information from emoji.

Future work

In the future, there are some possible improvement in the LSTMs motel to make it more practical:

1. Change the binary classifier to classifier with 3 outputs, respectively positive, negative and neutral.

2. Use larger data set. Google created 3 million word vectors. Each word vector has a dimensionality of 300. This larger word vector can produce a more general model.

3. Select emojis as features of the network to make the classifier sensitive to emoji information.

## Drunk Detection

Since analysis of Twitter has become a widespread approach for geo-spatial studies of human behavior, we have been inspired by many previous studies. In 2016, Nabil Hossain introduced a 3-SVM model to detect whether tweets are sent under the condition that the poster is drunk (Hossain, 2016). In addition, he provides a set of alcohol-related keywords, which is adopted in this system. In the data processing, tweets are filtered depending on if they included a mention of alcohol, defined by the inclusion of any one of several drinking-related keywords (e.g., “drunk”, “beer”, “party”) and their variants. After drunk detection, one value, 0 or 1, is stored for each tweet, where 0 represents this tweet has no relationship with drinking or alcohol and 1 represents that this tweet is sent under the condition that the poster is drunk.

# MapReduce

After being processed, the information we store in the database only includes tweet ID, suburb, sentiment polarity and drunk classification we get from previous process, hashtags, day of week. However, these massive and poorly organized information cannot be directly used for data visualization. In CouchDB, we need to predefine several views to enable the front-end server to access data for various scenarios. The information of these views are listed below.

|  |  |  |
| --- | --- | --- |
| Key | Value | Reduce |
| Suburb | Sentiment polarity | Average value of the sentiment polarity in the suburb |
| Suburb | Drunk classification | Average value of drunk classification in the suburb |
| Suburb, Day of Week | Sentiment polarity | Average value of the sentiment polarity in the suburb in this particular day of week. |
| Suburb, Day of Week | Drunk classification | Average value of the drunk classification in the suburb in this particular day of week. |
| Suburb, Single Hashtag | 1 | The overall count of the particular hashtag posted on the suburb. |

Cite: Alcohol-related keyword: cs.rochester.edu/ u/nhossain/icwsm-16-data.zip

Reference

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1532-1543).

Deshpande, A. (2017). Perform sentiment analysis with LSTMs, using TensorFlow.

<https://www.oreilly.com/learning/perform-sentiment-analysis-with-lstms-using-tensorflow>

Gilbert, C. H. E. (2014). Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Eighth International Conference on Weblogs and Social Media (ICWSM-14). Available at (20/04/16) http://comp. social. gatech. edu/papers/icwsm14. vader. hutto. pdf*.

Loria, S., Keen, P., Honnibal, M., Yankovsky, R., Karesh, D., & Dempsey, E. (2014). Textblob: simplified text processing. *Secondary TextBlob: Simplified Text Processing*.