

# CS 410: Final Project Documentation

## Brand Sentiment on Twitter using Sentiment Analysis

Team Stanley (Fall 2021)

### 1. Introduction

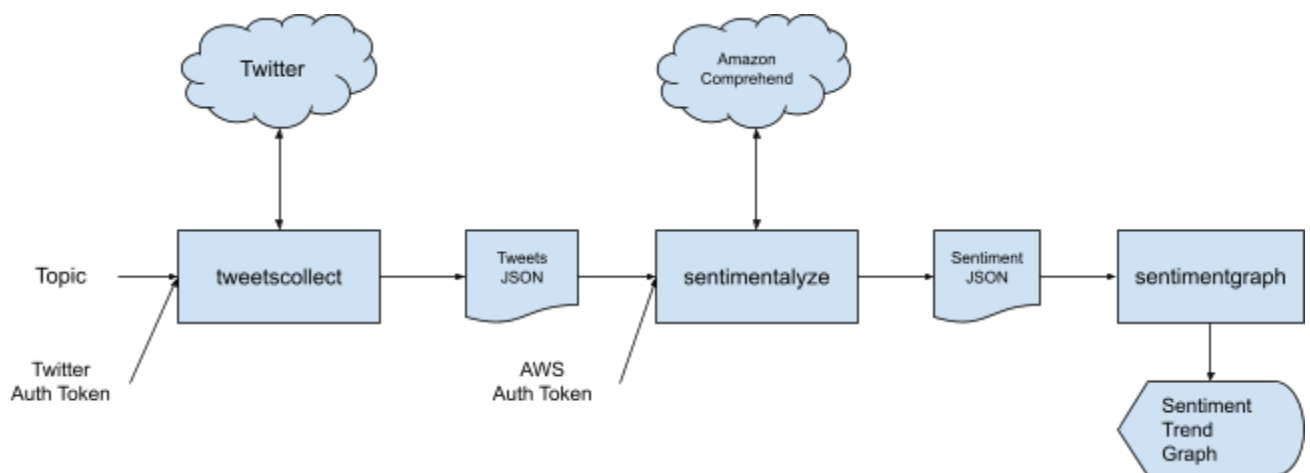
*Sentiment analysis* can capture the market or customer sentiment towards a brand. Companies can use this information to better understand their audiences' reactions to the brand's news or marketing campaigns, and to further enhance the brand. Investors or traders can also leverage this information to determine whether they should long or short their positions in the stock behind the brand.

This project is to perform *sentiment analysis* on the Twitter tweets related to a given brand over a period of time, and create a sentiment trend graph to visualize the sentiment towards the brand.

There are several tools developed for this project:

1. ***tweetscollect*** for collecting the tweets for a topic from *Twitter* for the past 7 days into a dataset.
2. ***sentimentalyze*** for performing *sentiment analysis* on the dataset.
3. ***sentimentgraph*** for plotting the *Sentiment Trend Graph* based on the *sentiment analysis*.

These tools are designed to work together as follows:



First, ***tweetscollect*** is used to collect the tweets for a given topic from Twitter and write the tweets into a file (i.e. tweets.json). Next, ***sentimentalyze*** takes the tweets returned from ***tweetscollect***, and performs *sentiment analysis* on these tweets using *Amazon Comprehend* and output the result into another file (i.e. sentiment.json). Afterwards, ***sentimentgraph*** takes the results from ***sentimentalyze*** to create a *Sentiment Trend Graph* for visualization.

## 2. tweetscollect

### 2.1. Overview

*tweetscollect* is a tool for collecting the tweets for a topic from *Twitter* for the past 7 days into a dataset. The topic could be one or more words in English. If the word contains special characters, e.g. “\$”, the character must be escaped accordingly, i.e. “\\$”.

### 2.2. Implementation

*tweetscollect* uses [Twitter's standard search API v1](#) to query against a mixture of the recent and popular tweets for the past 7 days for a given topic. Each API call returns a limited number of tweets, and multiple paginations are involved in order to collect all the tweets across 7 days. After all the tweets are collected, the tweets are further reduced to a minimum which includes only the date, text, language, favorite count, and retweeted count. At the end, the resulting tweets are written out to a file in json format. Notice that there were several considerations for the implementation:

1. *Twitter API v1.1 vs v2*. When we developed *tweetscollect*, Twitter's API v2 was in early access, and it lacked some of the functionalities we needed, hence we decided to use the Twitter API v1.1 instead.
2. *Query abilities and rate limiting*. Using the Twitter's standard developer account to collect tweets has several restrictions:
  - a. For the standard developer account, *Twitter* focuses only on relevance and not completeness in search results. This means that some tweets could be missing from search results.
  - b. There is also a limit which only allows querying the tweets in the past 7 days.
  - c. Each query request could return up to 100 tweets, but there is a rate limit allowing only 180 requests (with user authentication) in a 15-min window.
  - d. There is also a monthly tweet cap usage which allows only 500,000 tweets to be pulled.

As a result, *tweetscollect* focuses on collecting a mixture of recent and popular tweets over the past 7 days for a given topic, and its focus is on relevance and not completeness of the tweets.

3. *Standard mode vs extended mode.* Twitter originally supported only 140 characters in tweets, and later extended the support to 280 characters for certain languages. However, the Twitter API returns the tweet by default in compatibility mode which truncates the text. Hence, our implementation explicitly requests the text to be returned from Twitter in extended mode in order to overcome this issue.
4. *Tweets in different languages.* Twitter is a global service available in over 200 countries, and the tweets are written by the users all over the world in various languages. To analyze the tweets for a given topic, this presents a challenge as the entire tweet might not be written in a language which sentiment analysis would understand. For simplicity, our implementation only collects the tweets written in English.
5. *Cost of sentiment analysis.* It turns out that it is cost prohibitive to perform sentiment analysis on large volumes of text with public cloud's NLP service, and please see Section 3.2 below for more details. There are 500 millions tweets put up on average every day on Twitter. Even if 0.01% of the tweets are relevant to our topic for sentiment analysis, that's 50,000 tweets per day, and the cost could add up quickly. The total number of recent + popular relevant tweets is several orders of magnitude less than the total number of complete relevant tweets, and this is another motivation for our implementation to collect tweets focusing on relevance instead of completeness to make this project affordable.

## 2.3. Usage

To run *tweetscollect*, here are the prerequisites:

1. You must have [Go 1.17](#) installed.
2. You must have a *Twitter developer account* in order to create a *Bearer Token* to be used by *tweetscollect* for authentication. **If you are a reviewer who wants to test out the tool, please contact the author [smho2@illinois.edu](mailto:smho2@illinois.edu) directly to get the *Bearer Token*.**

Notice that running *tweetscollect* to collect tweets for the past 7-days involves retrieving tens of thousands of tweets from *Twitter*, and it will take a few minutes for the tool to run to completion. Please be patient!

Also notice that the *Twitter developer account* has a rate limit on the maximum number of requests allowed in a 15-minutes time window, and collecting the tweets for one topic alone might get very close to the limit. Hence, in order to use the tool successfully, please run the tool at most once in a 15-minutes time window.

```
# Build tweetscollect into an executable.
```

```
#
$ go build ./cmd/tweetscollect/...

# Show tweetscollect usage.
#
$ ./tweetscollect
Usage:
  tweetscollect -b <bearer-token> -t <topic> [-o <output-file>] [flags]

Flags:
  -b, --bearer-token string    Bearer token
  -h, --help                   help for tweetscollect
  -o, --output-file string     Output file (default "tweets.json")
  -t, --topic string           Topic, e.g. Facebook
  -v, --verbose count          Increase verbosity. May be given multiple times.

# Run tweetscollect to collect tweets for a topic, by using the bearer token
# from your Twitter developer account, and write the collected tweets to file.
#
$ ./tweetscollect -b "<bearer-token>" -t "<topic>"
Collecting tweets from Twitter on topic "<topic>" ...

{ "date": "2021-11-16T00:46:05Z", "text": "JUST IN: Ohio Attorney General sues
Facebook (Meta) for securities fraud.", "lang": "en", "favorite": 1077, "retweet":
328 }
...

Writing collected tweets to tweets.json ...
Done.
```

## 3. sentimentalyze

### 3.1. Overview

*sentimentalyze* is a tool for performing *sentiment analysis* over the dataset with tweets which *tweetscollect* has collected.

### 3.2. Implementation

*sentimentalyze* first normalizes all the tweets from the dataset, as there are many retweets and dedupling the tweets could significantly reduce the unique number of tweets for sentiment analysis. Afterwards, *sentimentalyze* sends the unique tweets to *Amazon Comprehend* in batches to determine the sentiment. After all the unique tweets have been analyzed, *sentimentalyze* would reprocess each of the original tweets from the dataset, identify its

associated unique tweet and sentiment, and eventually write out the tweets along with their sentiment to a file in json format. Notice that there were several considerations for the implementation:

1. *Retweet*. Many *Twitter* users don't write their own tweets, and they often favorited or retweeted what others had written instead. This means many tweets returned from *Twitter* had almost identical text, as the duplication is not ideal. Hence, our implementation normalizes the tweets by deduplication, and only performs sentiment analysis on the unique tweets. After the analysis, our implementation would reprocess all the tweets again and associate the sentiment result with each tweet accordingly.
2. Cost of *Amazon Comprehend* for sentiment analysis. It turns out that it is cost prohibitive to perform sentiment analysis on large volumes of text with *Amazon Comprehend*. For example, Amazon charges \$0.0001 per unit (up to 10M units) which each unit is 100 characters. Simply analyzing 10,000 tweets each with 200 characters alone would cost \$2. This is another motivation for our implementation to normalize the tweets to reduce duplication before performing sentiment analysis.

### 3.3. Usage

To run *sentimentalyze*, here are the prerequisites:

1. You must have [Go 1.17](#) installed.
2. You must have an AWS account in order to create an access key to be used by *sentimentalyze* for authentication. **If you are a reviewer who wants to test out the tool, please contact the author [smho2@illinois.edu](mailto:smho2@illinois.edu) directly to get a temporary AWS access key.**

Notice that running *sentimentalyze* to perform sentiment analysis involves sending all the tweets to *Amazon Comprehend* in multiple batches to process, and it will take 15 to 20 minutes for the tool to run to completion. Please be patient!

Please be aware that since *sentimentalyze* will use *Amazon Comprehend* from the AWS account, the AWS account will be charged for usage. On average, each run involves between 40,000 to 60,000 tweets, and that's approximately 15,000 to 25,000 unique tweets which costs \$1.5 to \$2.5 to perform a single run of *sentiment analysis*.

```
# Build sentimentalyze into an executable.
#
$ go build ./cmd/sentimentalyze/...

# Show sentimentalyze usage.
#
```

```

$ ./sentimentalyze
Usage:
  sentimentalyze -i <input-file> -o <output-file> -a <access-key-id> -s
<secret-access-key> [-r <region>] [-l <lang>] [flags]

Flags:
  -a, --access-key-id string      Access key ID
  -h, --help                      help for sentimentalyze
  -i, --input-file string         Input file (default "tweets.json")
  -o, --output-file string        Output file (default "tweets-sentiment.json")
  -r, --region string             Region (default "us-east-1")
  -s, --secret-access-key string  Secret access key
  -v, --verbose count            Increase verbosity. May be given multiple times.

# Run sentimentalyze to perform sentiment analysis on the tweets in the
# input file, using the access key ID and secret access key from the
# AWS account.
#
$ ./sentimentalyze -i <input-file> -o <output-file> -a <access-key-id> -s
<secret-access-key> [-r <region>]
Reading 40655 tweets from tweets.json ...
Normalizing 40655 tweets into 13089 unique tweets ...
Performing sentiment analysis on the unique tweets ...
{ "text": "Insight into the big rebrand.. 🕶️\nvia \n#Facebook #AR #VR #meta
#Metaverse #virtualworlds\n\nhttps://t.co/bMQ4hhKlTW", "sentiment": "NEUTRAL" }
{ "text": "Facebook owner Meta has opened up more about the amount of bullying and
harassment on its platforms amid pressure to increase transparency\n\n👉:",
"sentiment": "NEUTRAL" }
...
Writing tweets with analyzed sentiment to sentiment.json ...
Done.

```

## 4. sentimentgraph

### 4.1. Overview

*sentimentgraph* is a tool for creating *Sentiment Trend Graph* based on the *sentiment analysis* which *sentimentalyze* has performed.

## 4.2. Implementation

*sentimentgraph* simply reads the result of the *sentiment analysis* from a file, and uses `matplotlib.pyplot` library to create a *Sentiment Trend Graph*. The sentiment returned from the *sentiment analysis* for each tweet is either POSITIVE, NEGATIVE, NEUTRAL, or MIXED. To plot the graph, we map POSITIVE into a sentiment value of 1, NEGATIVE into -1, and NEUTRAL/MIXED into 0 respectively.

## 4.3. Usage

To run *sentimentgraph*, here are the prerequisites:

1. You must have [Python 3.10](#) installed.
2. You must also have [matplotlib library](#) installed.

Notice that running *sentimentgraph* involves plotting a *Sentiment Trend Graph* with tens of thousands of data points, and it will take a few minutes to display the visualization. Please be patient!

```
# Go to sentimentgraph directory.
#
$ cd ./cmd/sentimentgraph/

# Show sentimentgraph.py usage.
#
$ ./sentimentgraph.py
Usage:
  sentimentalyze.py <sentiment-file>

# Run sentimentgraph.py to create the Sentiment Trend Graph.
#
$ ./sentimentgraph.py sentiment.json
Loading sentiment data from sentiment.json, it will take a minute or two ...
Loaded 2000 sentiment data ...
Loaded 4000 sentiment data ...
Loaded 6000 sentiment data ...
Loaded 8000 sentiment data ...
Loaded 10000 sentiment data ...
Loaded 12000 sentiment data ...
Loaded 14000 sentiment data ...
Loaded 16000 sentiment data ...
Loaded 18000 sentiment data ...
Loaded 20000 sentiment data ...
Loaded 22000 sentiment data ...
Loaded 24000 sentiment data ...
```

```
Loaded 26000 sentiment data ...  
Loaded 28000 sentiment data ...  
Loaded 30000 sentiment data ...  
Loaded 32000 sentiment data ...  
Loaded 34000 sentiment data ...  
Loaded 36000 sentiment data ...  
Finished loading 37742 sentiment data!  
Plotting sentiment trend graph ...
```