Sanchit Goel
Rohan Behera
Ishika Rakesh
Raghav Gupta
Benson Wan
CS172 - Information Retrieval
8 May 2022

CS172 Project Phase 1 Report

Collaboration Details:

- **Sanchit Goel:** Wrote a function in crawl.py to convert a tweet to a python dictionary and crawled 400 megabytes of tweets. Also developed a function to calculate the total tweets crawled and time, the tweets crawled per minute, and the number of files created to store those tweets.
- **Rohan Behera:** Crawled an estimated 400 megabytes worth of data to satisfy the storage requirement. Wrote a boolean function in our initial crawler.py file to filter out all tweets that are retweets. Wrote filter_tweet function exclude any retweets we may find while running the crawler in crawl.py and stream.py
- **Ishika Rakesh:** Crawled 400 megabytes of tweets. Worked on data analysis and analyzing trends in the JSON output. Took screenshots to ensure the code is optimal. Implemented a test case in crawl.py where there were no tweets found. Configured the bearer token and specified the number of tweets in each JSON file in the .env file.
- **Raghav Gupta:** Crawled 400 megabytes of tweets. Figured out the data collection aspect of the program to ensure that what we are collecting is relevant to what we want to build and display when it comes to the UI phase of the project. Recommended JSON as a way to store information rather than CSV as it's easier to parse and store in a database if needed, also allows for nested objects. The data collection aspect includes the tweet id, the user ID used to tweets, time and date the tweet was sent, device it was sent from, the tweet text, likes, and retweets in the crawl.py file and the tweets are stored in a collection of JSON files.
- **Benson Wan:** Crawled 400 megabytes of tweets. Contributed to figuring out the logistical aspects of what tweets are collected, what fields among those

tweets are going to be used, and the filtering aspect of said tweets. Also Studied and analyzed the stream.py class to convey the functionality of the code in this report.

System Overview:

a. Crawling Strategy:
    i. The objective of our crawler is to retrieve tweets that relate to the topic of the Marvel Cinematic Universe or Marvel Comics. This is a broad topic that spans a series of movies, TV shows, and text material.
    ii. Initially, our idea was to get the most recent tweets and then jump from one tweet to another tweet based on some criteria. However, this posed several challenges. The first and most prominent challenge was that there was no criteria which would ensure that we jump to a tweet related to the Marvel topic without having to jump through several unrelated tweets. The reason this was an issue was because of the Twitter API Tweet Cap which limited the number of tweets that can be retrieved.
    iii. Because there was no efficient way to jump between tweets without easily hitting that Tweet Cap, we opted to get real-time tweets that related to the Marvel topic. The Twitter API has an ability to define a set of "streaming rules" that can filter only retrieving real-time tweets that match the set of rules created. In essence, we could ensure that we were only getting tweets that related to Marvel with a low chance of capturing any unrelated tweets. Furthermore, we also did not have to check for duplicate tweets because we were only getting new tweets.
    iv. We also strongly avoided getting retweets because very little new content is provided by retweets, so we only focused on getting original tweets.
    v. However, this method comes with the issue of being slow since it depended on people tweeting related to the topic, so collecting data was a long process, but much more feasible than the alternatives.

b. Architecture:
    i. The objective of crawl.py is to get the most recent tweets that match a specific query and store them into a JSON file. This was the first

iteration of the crawler, but NOT the one used for data collection. Describing the architecture of this file is an aid to describe the architecture of stream.py.

Inside crawl.py, we first import tweepy which makes accessing the Twitter API with python more convenient. We used a dotenv library which will help with setting environment variables from .env files we utilize. We also imported JSON to help write our python objects into JSON files.

We created some constant variables along with a query that has different hashtags that we are surfing through twitter for. In the query, we utilize Twitter query operators like "OR" to help us match specific hashtags and keywords in a tweet. We also use some filters like "-is:retweeted" to filter any retweets and "lang:en" to only get tweets in English.

Once those are added, we validate that the MAX_TWEETS constant which dictates the number of recent tweets to query is set correctly within the range of 10 and 100 tweets (a constraint defined by the Twitter API).

After the code checks for this, we have to load the .env file using dotenv_values and which returns a dictionary with respective key-value pairs. We use the BEARER_TOKEN to authenticate the tweepy client.

To get the information we want about a tweet, we have to specify the source fields we want on a tweet which are id, text, author_id, created_at, geo, source, and public_metrics. We then search through the recent tweets that are most relevant and based on the specified tweet fields so we use a search_recent_tweets function to do so and create a variable for tweets and load data into it.

We check if no tweets were queried (which likely indicates an error within the query or request) and fast-fail out of the program.

Based on the relevant fields, we then convert a tweet to a python dictionary and we use the tweets_to_dict function to do so. We are also checking if the tweet is geo tagged and its metrics as well and then we return the converted tweet. Once this is done we filter the tweets that are excluding the re-tweets by using a filter_tweet function and making sure it excludes it by specifying it doesn't start with "RT @". Once these return true or false, we write the filtered tweets to a JSON file.

ii. To get our data, we mainly utilized stream.py which builds on top of the architecture in crawl.py.
In stream.py, we started out pretty similarly to crawl.py where we import the same libraries, but on top of these, we also imported time and threading for the sake of multithreading and efficiency within our program.

The main additional component in stream.py is the creation and utilization of streaming rules. These streaming rules are what allow us to only get real-time tweets that relate to our topic. We load the "stream_rules.json" file into memory and convert these to Tweepy stream rules. Then we fetch the current stream rules on the streaming endpoint and perform list comparisons. Based on the differences, we add and remove streaming rules to the streaming endpoint.

After the streaming rules are created, we subclass the Tweepy StreamingClient in order to custom handle how each tweet received is processed. For each tweet received, we run it through a filter to make sure that it is not a retweet, then we add it to a stored list. We check if the size of that list matches a user-defined constant of tweets in a single file, and if it does we query the creation of a new JSON file to store that list of tweets.

The streaming client runs in a separate thread and will not stop streaming until the user types in "quit" or does CTRL+C, or CTRL+Z to force end it.

After the stream ends, metrics are printed about the number of tweets in the time elapsed.

c. Data Collection Strategy:

i. We are using the Twitter Streaming API known as Tweepy which uses authentication via a bearer token generated by the Twitter developer API. Tweepy is a wrapper around the Twitter REST endpoints and allows us to get all Twitter API responses as pythonic objects.

ii. Any tweets that we collected were filtered to only store specific fields of the tweet.

iii. These tweets were stored in JSON files. Each JSON file contains 3,100 tweets which roughly amount to 1 MB of tweets per file.

iv. While we could store more tweets, the reason we chose 1 MB was because in case of hardware interruption, we would not lose a significant amount of tweets that were not saved.

v. Furthermore, because it is in a JSON file, data post-processing can be easily done to merge the tweets into larger files if necessary.

vi. The fields collected for a tweet were:

    1. *id* : Unique ID of the tweet.
    2. *user_id* : Unique ID of the author of the tweet. Thiscan be used to then query the author's display name, Twitter handle, and profile picture if necessary.
    3. *created_at* : Date/time the tweet was tweeted.
    4. *device* : Source of the tweet.

5. *text* : Text content of the tweet.
6. *likes* : Number of likes the tweet received.
7. *retweets* : Number of retweets for that specific tweet.
8. *Geo Location:* Geo location data (specifically point data) if any.

d. Data Structures Employed:

i. The data structures employed in phase 1 of our project were a Python dictionary to store the tweet ID,, what device the tweet was sent from, date/time of the tweet, geolocation of the tweet, and the content of the tweet itself. We also used lists to store the dictionary values and then stored these lists directly into JSON files.

Limitations:

One limitation of our crawler is that it takes quite a bit of time to retrieve 2 gigabytes worth of data and store them in JSON files. Another challenge we encountered was removing retweets so we wouldn't have any duplicate tweets and all of the tweets would be original. Another challenge we faced is the limit given by the elevated twitter developer accounts as we can only crawl 2 million tweets per month and we needed approximately 4.5 million tweets to satisfy the storage requirement. To get around the elevated twitter account limit, each of us applied for the twitter developer account so that way we would not have to worry about going over the allowance for the month.

Instructions:

In order to run the crawler, the system running it must have the following:

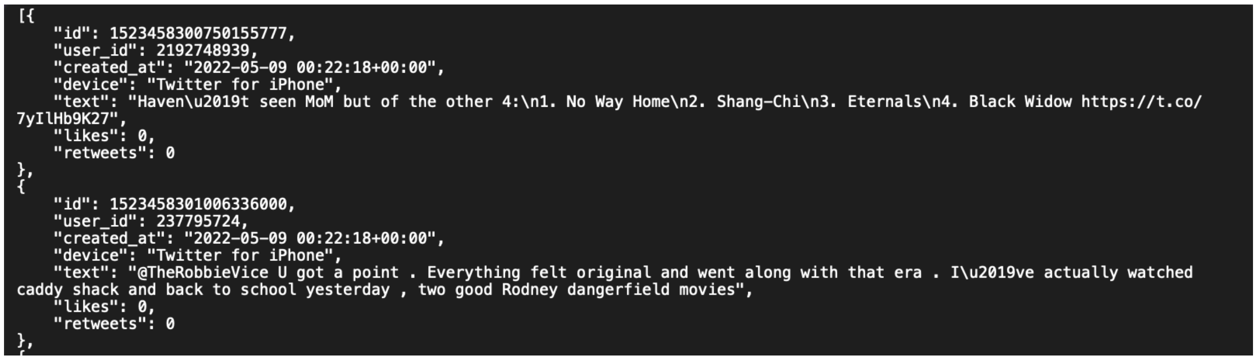
- Git
- Python 3
- Pip 3 (Python Package Installer)

The instructions to get the crawler running:

- Unzip (or use an extraction tool) to decompress the CS172 project directory from the submission.
- Open up the terminal/command prompt on your computer and run the following commands.
- cd CS172Project
- pip install -r requirements.txt

- cp .env-example .env
    - Must replace BEARER_TOKEN in the `.env` file with a valid Twitter Developer API bearer token.
- python3 stream.py
- The tweets will be stored in different JSON files.
    - You will be able to view the following information for the tweets:
        - id
        - user_id
        - created_at
        - device
        - text
        - likes
        - Retweets

What output should look like:

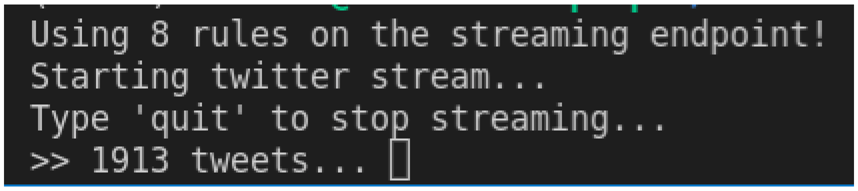


```
[{
    "id": 1523458300750155777,
    "user_id": 2192748939,
    "created_at": "2022-05-09 00:22:18+00:00",
    "device": "Twitter for iPhone",
    "text": "Haven\u2019t seen MoM but of the other 4:\n1. No Way Home\n2. Shang-Chi\n3. Eternals\n4. Black Widow https://t.co/
7yIlHb9K27",
    "likes": 0,
    "retweets": 0
},
{
    "id": 1523458301006336000,
    "user_id": 237795724,
    "created_at": "2022-05-09 00:22:18+00:00",
    "device": "Twitter for iPhone",
    "text": "@TheRobbieVice U got a point . Everything felt original and went along with that era . I\u2019ve actually watched
caddy shack and back to school yesterday , two good Rodney dangerfield movies",
    "likes": 0,
    "retweets": 0
},
```

Screenshots:

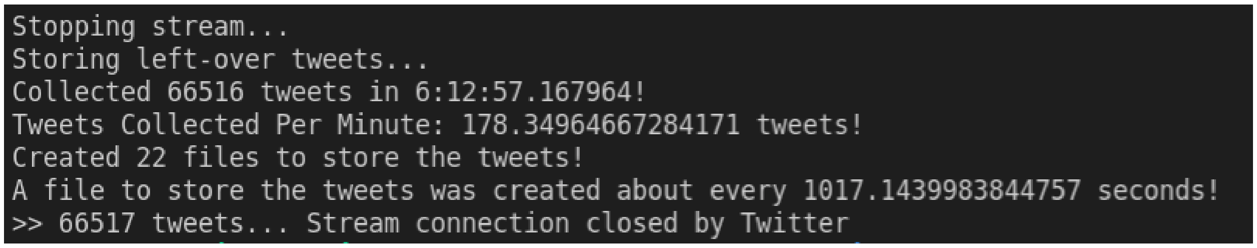
Running the crawler:

Displays an updating number of tweets retrieved.



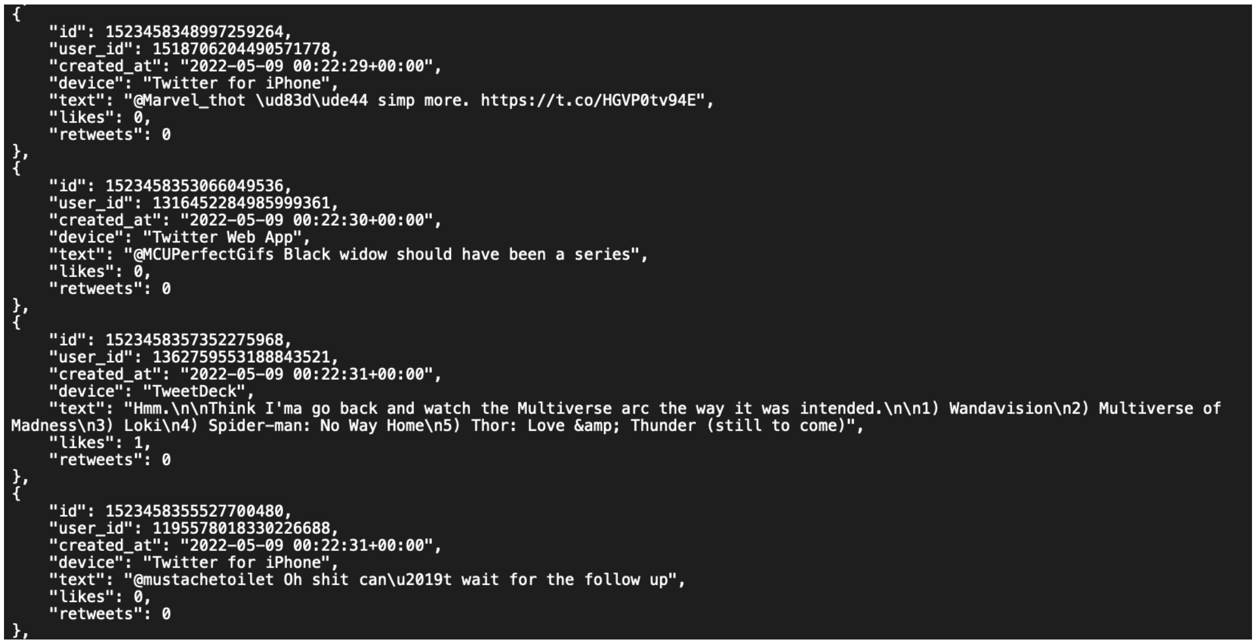
```
Using 8 rules on the streaming endpoint!
Starting twitter stream...
Type 'quit' to stop streaming...
>> 1913 tweets...
```

Stopping the stream and receiving the metrics:



```
Stopping stream...
Storing left-over tweets...
Collected 66516 tweets in 6:12:57.167964!
Tweets Collected Per Minute: 178.34964667284171 tweets!
Created 22 files to store the tweets!
A file to store the tweets was created about every 1017.1439983844757 seconds!
>> 66517 tweets... Stream connection closed by Twitter
```

Example of JSON file:



```
{
    "id": 1523458348997259264,
    "user_id": 1518706204490571778,
    "created_at": "2022-05-09 00:22:29+00:00",
    "device": "Twitter for iPhone",
    "text": "@Marvel_thot \ud83d\ude44 simp more. https://t.co/HGVP0tv94E",
    "likes": 0,
    "retweets": 0
},
{
    "id": 1523458353066049536,
    "user_id": 1316452284985999361,
    "created_at": "2022-05-09 00:22:30+00:00",
    "device": "Twitter Web App",
    "text": "@MCUPerfectGifs Black widow should have been a series",
    "likes": 0,
    "retweets": 0
},
{
    "id": 1523458357352275968,
    "user_id": 1362759553188843521,
    "created_at": "2022-05-09 00:22:31+00:00",
    "device": "TweetDeck",
    "text": "Hmm.\n\nThink I'ma go back and watch the Multiverse arc the way it was intended.\n\n1) Wandavision\n2) Multiverse of
Madness\n3) Loki\n4) Spider-man: No Way Home\n5) Thor: Love &amp; Thunder (still to come)",
    "likes": 1,
    "retweets": 0
},
{
    "id": 1523458355527700480,
    "user_id": 1195578018330226688,
    "created_at": "2022-05-09 00:22:31+00:00",
    "device": "Twitter for iPhone",
    "text": "@mustachetoilet Oh shit can\u2019t wait for the follow up",
    "likes": 0,
    "retweets": 0
},
```