Data Processing Algorithms: Twitter Download

Fatema Bhayani
6-18-2020

1 Table of Contents

2 Introduction

The algorithms developed in this project can be categorized into the following categories:

1. Twitter Download Algorithms: These algorithms download social network data from Twitter using the Twitter API.
2. Tweet Processing Algorithms: These algorithms process the tweets by removing stop words, hashtags, tweets in foreign languages etc.
3. Clustering Algorithms: These algorithms cluster users based on their similarity and their connections in the social network. Moreover, these algorithms also analyze the word frequency of the user's tweets and determine what communities they belong to.

This report will focus on Twitter Download Algorithms.

Twitter Downloader has been split up into two classes, one for a specific domain of Twitter download. We can imagine adding new classes of Twitter download as the need arises.

```
"""

Download Tweets for use in future algorithms.

"""

class TwitterTweetDownloader(Process):
```

```
"""

Download Twitter Friends for use in future algorithms.

"""

class TwitterFriendDownloader(Process):
```

3 Twitter API Interface

The Twitter API allows developers to follow, search and get users and tweets. More specifically, the Twitter API makes it easier to get a list of followers and friends of a user. The friends of a user are the Twitter users that a specific user follows. On the other hand, the followers of a user refers to the Twitter users that follow a specific user. Twitter API can also be used to get tweets of a specific user.

The Twitter API exposes dozens of HTTP endpoints that can be used to retrieve, create and delete tweets, retweets and likes. It provides direct access to rich and real-time tweet data, but requires having to deal with a lot of low level details. Tweepy is an open source python library package that allows you to bypass a lot of those low-level details. We use Tweepy to access the Twitter API since it allows us to interact with the Twitter API easily.

4 Twitter Download Functions

The Twitter Download Functions available can be summarized by the following table below.

Function Name	Description
get_tweets_by_timeframe_user	Return *num_tweets* tweets made by user with (username or user id) *id* between *start_date* and *end_date*.
get_tweets_by_user	Return *num_tweets* tweets made by user with (username or user id) *id*
get_random_tweets	Return *num_tweets* random tweets
get_friends_by_screen_name	Return an array of screen names of the user's friends with screen name *screen_name*.
get_friends_by_id	Return an array of ids of friends of user with id *id*.
get_following_by_screen_name	Return an array of screen names of the user's followers with screen name *screen_name*.
get_following_by_id	Return an array of ids of the followers of the user with id *id*.

Moreover, we have a UserListProcessor class with the following method,

def user_list_parser(self, user_list_file_path):

This takes in a file containing a list of users, where we have a different user on each line. It then returns a list of these users.

What we can do after is run,

def download_function_by_user_list(self, download_function, user_list, *argv) -> List[List]:

This is a higher order function that takes in any of the above methods (download function) and is able to run said functions on each user in the user list.

For example, we can do,

download_function_by_user_list(self,

twitter_downloader.get_tweets_by_timeframe_user, user_list, start_date, end_date, num_tweets)

And this runs,

twitter_downloader.get_tweets_by_timeframe_user(id, start_date, end_date: datetime, num_tweets)

where *id* is an element of *user_list* for every element in *user_list*.

This takes advantage of functional programming features present in Python that allow us to have higher order functions, as well as work with an arbitrary number of arguments so that they can be bound to different types of functions.

This way, we don't need functions such as,

- get_followers_by_user_list
- get_following_by_user_list
- get_timeline_by_user_list
- get_timeline_by_timeframe_user_list

That would be a mess and too many functions. Instead, we have one function that generalizes these operations.

5 Tutorial and Examples

5.1 Getting Twitter API Authentication

Apply for a developer account on https://developer.twitter.com/en/apply-for-access. It will take a couple of days for your application to be approved. Once your application is approved, you will get a consumer key, consumer secret, access token, and access token secret.

Run the code below to verify your authentication.

```
import tweepy

# Authenticate to Twitter
auth = tweepy.OAuthHandler("CONSUMER_KEY", "CONSUMER_SECRET")
auth.set_access_token("ACCESS_TOKEN","ACCESS_TOKEN_SECRET")api =
tweepy.API(auth)# test authentication
try:
    api.verify_credentials()
    print("Authentication OK")
except:
    print("Error during authentication")
```

The code is currently hosted on GitHub as an organization public repository. The following is a link to the repo: https://github.com/Social-Network-Algorithms/

Clone the repo onto your local device. In the main directory, make a new file “credentials.py”. In it, add your credentials for accessing the Tweepy API in the following format:

```
ACCESS_TOKEN = <stuff here>
ACCESS_TOKEN_SECRET = <stuff here>
CONSUMER_KEY = <stuff here>
CONSUMER_SECRET = <stuff here>
```

5.2 Download Data from Twitter

To download data from Twitter, we first need a Datastore Configuration File (DSConfig). To create a datastore configuration file for download Tweets, the file would look something like this,

```
input-datastores:

  input-ds1:
```

```
    project-name: "TwitterDownload"
    datastore-name: ""
    collection-name: ""
    type: "Tweepy"
    location: ""



output-datastores:
  output-ds1:
    project-name: "TwitterDownload"
    datastore-name: "TwitterDownload"
    collection-name: "Tweets"
    type: "MongoDB"
    location: "mongodb://localhost:27018"
```

We are downloading data from Twitter, so our **input** datastore is Tweepy. We are using MongoDB to store our downloaded data, which in this case are Tweets. Thus, we want to **output** the downloaded data to the Tweets collection inside the TwitterDownload database, whose location can be accessed at the specified path. Note that, for MongoDB paths, we access a given database by specifying an IP and port, as opposed to giving an actual physical path.

We have at this point a datastore initial configuration file for downloading Tweets. The next step is to instantiate a TwitterTweetDownloader object.

```
twitterDownloader = TwitterTweetDownloader("ds-init-config.yaml")
```

And just like that, we are able to download Tweets.

```
twitterDownloader.get_tweets(user_name, start_date, end_date, num_tweets)
```

The downloaded data is stored in the output collection and database, as specified by “ds-init-config.yaml.”

If we want to download a list of a user's friends, then we could first modify the output datastore section for our datastore config file.

```
output-datastores:

  output-ds1:

    project-name: "TwitterDownload"

    datastore-name: "TwitterDownload"

    collection-name: "Friends"

    type: "MongoDB"

    location: "mongodb://localhost:27018"
```

Notice here that we changed the collection name to Friends, since we want different types of downloaded data to be stored in different collections.

Then, just like before, we instantiate a Downloader object and call the respective method to download.

```
twitterDownloader = TwitterFriendsDownloader("ds-init-config.yaml")

twitterDownloader.get_friends_by_screen_name(user_name, num_friends_to_get)
```

5.2.1 Download Random Tweets

In order to automate the daily download of raw tweets, we have created a script called raw_tweet_scheduler_daemon.

To use this script, simply call it like so,

```
python3 raw_tweet_scheduler_daemon.py --num-tweets-to-download <num_tweets> --
time-to-download <time> --options <option-value>
```

Notice that we provide two arguments,

1) number of tweets to download
2) time each day we want to download our tweets

"options" currently serves as a placeholder for additional configuration options that this program will provide that are not yet implemented (such as how a user would like to receive monitoring information). Furthermore, if num-tweets-to-download is not specified, the program will assume

that the user wants to download at the full rate limit. If time-to-download is not specified, the program will download at a default set time.

There are three aspects of this script.

1) Schedule daily.
2) Download raw tweets.
3) Run as a daemon in the background.

To address the issue of scheduling, we use a Python library called schedule. This allows us to run the raw Tweet download function at a specified time daily.

To address the issue of running in the background, we use another Python library to run the script as a daemon. This library handles many issues associated with running a daemon, such as not printing to stdout.

Addressing the issue of downloading raw tweets, we are using the random Tweet download method,

```
get_random_tweets(self, num_tweets: int, output_collection_name: str)
```

The way this works is that we have the following function,

```
def download_daily_tweets():
```

Our scheduler calls this function daily. In download_daily_tweets, we generate a collection name, which is based on the date. We pass in the given collection date to get_random_tweets.

In turn, get_random_tweets creates a new collection with the specified name and where documents (elements) in this collection are Tweet objects (contain information like text, tweeter, location).