Twitter Scraping Tutorial

2024

1 Introduction

Since my last video on Twitter scraping, many events have unfolded: Elon bought Twitter, rebranded it to `X`, and then did his best to shut down free scraping. In 2024, obtaining data from Twitter has become one of the most challenging tasks. In this document, I demonstrate how I managed to extract data from the website, share tips and tricks to avoid getting banned, and explain how to handle rate limit exceptions.

2 Setup and Installation

In the terminal, install the required package:

```
pip install Tkit
```

This package is used to scrape Twitter data.

Create a new file called `main.py` and include the following imports:

```
from TWkit import client % Helps interact with Twitter and handle
    ↪  rate limit errors
import too_many_requests
import time
from datetime import date, time
import CSV
from configparser import ConfigParser
from random import randint
```

3 Constants and Query Setup

Define the constants needed for the script:

- `minimum_tweets`: Set to 10 initially (this value may be increased later).
- Query: Start with a simple query like `"chat GPT"`.

4 Authentication and Credentials

4.1 Login Credentials

It is recommended to create a secondary Twitter account to use for scraping, so that if it gets banned, your main account remains unaffected. Create a file called `config.ini` with the following content:

```
[X]
username = your_username
password = your_password
email = your_email@example.com
```

4.2 Reading Credentials in Code

Read the configuration file and set the variables:

```
config = ConfigParser()
config.read("config.ini")
username = config.get("X", "username")
email = config.get("X", "email")
password = config.get("X", "password")
```

4.3 Authentication to Twitter

There are two ways to authenticate:

1. Using login credentials.
2. Using cookies (log in once, save cookies, then reuse them).

Create a client instance and log in:

```
client = client(language="en-us")
client.login(authentication_info1=username, authentication_info2=
    ↪ email, password=password)
client.save_cookies("cookies.json")
```

Running this code creates a `cookies.json` file. In subsequent runs, comment out the login lines and load the cookies:

```
client.load_cookies("cookies.json")
```

5 Fetching Tweets

To retrieve tweets, execute:

```
tweets = client.search_tweet(query="chat␣GPT", product="top")
for tweet in tweets:
    print(tweet.to_dict())
    break
```

This prints one tweet as a dictionary, containing data such as user ID, name, and more.

6 Processing Tweet Data

Instead of converting the tweet to a dictionary immediately, extract the important information:

- Tweet count
- User name
- Tweet text
- Creation date
- Number of retweets
- Number of likes

Example code:

```
tweet_count = 0
for tweet in tweets:
    tweet_count += 1
    tweet_data = [
        tweet_count,
        tweet.user.name,
        tweet.text,
        tweet.created_at,
        tweet.retweets,
        tweet.likes
    ]
    print(tweet_data)
```

Running the code prints the tweet data in the terminal.

7 Handling Multiple Batches and Rate Limits

To fetch more than one tweet at a time, use a while loop:

```
tweet_count = 0
tweets = None
while tweet_count < minimum_tweets:
    if tweets is None:
        tweets = client.search_tweet(query="chat GPT", product="
            ↪ top")
    else:
        print("Getting next tweets...")
        tweets = tweets.next()

    if not tweets:
        print("No more tweets found.")
        break

    for tweet in tweets:
        tweet_count += 1
        % Process tweet data here
```

7.1 Simulating Human Behavior with Delays

To avoid being flagged by Twitter, introduce random delays before each API call:

```
import random

def get_tweets():
    tweets = client.search_tweet(query="chat GPT", product="top")
    wait_time = random.randint(5, 10)
    print(f"Getting new tweets after waiting for {wait_time}
        ↪ seconds")
    time.sleep(wait_time)
    return tweets
```

7.2 Saving Tweets to a CSV File

Open a CSV file before authentication to store the tweet data:

```
import csv

with open("tweet.csv", "w", newline="") as file:
    writer = csv.writer(file)
    writer.writerow(["tweet_count", "username", "text", "
        ↪ created_at", "retweets", "likes"])
```

While processing tweets, write each row to the CSV:

```
with open("tweet.csv", "a", newline="") as file:
    writer = csv.writer(file)
    writer.writerow(tweet_data)
```

7.3 Handling Rate Limit Exceptions

Wrap the tweet-fetching logic in a try-except block to manage rate limits:

```
try:
    tweets = client.search_tweet(query="chat␣GPT", product="top")
except too_many_requests as e:
    rate_limit_reset = datetime.fromtimestamp(e.rate_limit_reset)
    print("Rate␣limit␣reached.␣Waiting␣until", rate_limit_reset)
    wait_time = (rate_limit_reset - datetime.now()).total_seconds
        ↪ ()
    time.sleep(wait_time)
    continue
```

8 Complex Queries

To construct a more complex query (for example, to fetch tweets from Elon Musk's account), you can use Twitter's advanced search features. For instance, the following query retrieves tweets in English from Elon Musk between January 1, 2018, and January 1, 2020:

```
from:elonmusk since:2018-01-01 until:2020-01-01 lang:en
```

Copy this query into your code and run it.

9 Conclusion

This guide shows how to scrape Twitter data for free by using login credentials, cookie management, and handling rate limits gracefully. By simulating human behavior with delays and managing exceptions, you can reduce the risk of being banned while efficiently collecting data. Happy scraping!