Table of Contents

IP REPORT 02

A static html file is available here for the better reading experience in a browser. Highly recommend.

1 Introduction

A quick recap, this project aims to build an Instagram hashtag recommender that suggests less hot alternatives based on the given hashtag. The main structure is to have 3 components: a Python web scrapper that stripes relevant info from Instagram and store the info into Database, a MySQL database, an JavaScript interface for user interaction and data retrieval from database. In this report, it will mainly focus on the progress made for the Python scrapper and the database table design.

So far, my role is to build a backend scrapper logic *without* using `selenium` package which demostrated in previous presnetaion.

2 Python Scrapper

So far the `mainlogic.py` is a fully functional command line tool to print out the top 9 posts info of the input hashtag, kindly try it out. (potential failure : I included my cookie string in the headers object for the web access, in order to stay login – FaceBook limits the number of search without login, may not work for other PC)

2.1 Basic set up

Python environment set up as below:

```
li@intel ~ % python --version
Python 3.8.5
li@intel ~ % which python
/Users/li/.pyenv/shims/python
li@intel ~ % pip --version
pip 20.3.3 from /Users/li/.pyenv/versions/3.8.5/lib/python3.8/site-packages/pip (python 3.8)
li@intel ~ % which pip
pip: aliased to /Users/li/.pyenv/versions/3.8.5/bin/pip
li@intel ~ %
```

`pyenv` is used to manage the default python version on my system, and `pip` as the package manager.

Create a pyton file in the working directory (where `.git` locates too) called `mainlogic.py`, which stores the main steps needed for the scrapper. The entry point of the program is defined as below:

```
if __name__ == "__main__":
    start_time = time.time()
    start()
    print("-- execution time: %ss --" % (time.time() - start_time))
```

The time function here is to calculate the program run time for internal use. Logic will be put into `start()` function with each step to abstracted into function as possible.

2.2 Dependencies

The packages used as below:

```
import urllib.request
import urllib.error # get webpage by URL
import re
import json
import time
from datetime import datetime, date
from bs4 import BeautifulSoup # python -m pip install bs4 # parse web, obtain data
import mysql.connector
from mysql.connector import errorcode
```

BeautifulSoup(4.9.3) and mysql-connector-python(8.0.22) are the packages that not included in Python3 by default, the rest comes with Python. In the GitHub repository, `requirement.txt` file is provided for installing the dependencies, which is generated by `pip freeze`, therefore extra packages are included unnecessarily at the moment.

To install the dependencies, `cd` to the working directory and type below command:

```
python -m pip install -r requirements.txt
```

Notice that `selenium` package is not required in this approach.

2.3 Logic

The main logic of the scrapper is comprised of below parts:

1. Get the content of the desired webpage
2. Parse the content to pull only the relevant info
3. Save the info to a DataBase

2.3.1 HOLD Get the content of the webpage

First we notice that to search a hashtag, the url is always "https://www.instagram.com/explore/tags/ *SomeHashtag*". We need to access the url and rip the HTML of it. A function `get_content_url(url)` is defined for that purpose, take url as parameter and return the whole HTML:

```
def get_content_url(url): # return page source HTML

    data = bytes(urllib.parse.urlencode({'name': 'user'}), encoding = "utf-8")

    cookie = '''csrftoken=LqWwmIYutdVdjfZs19lC21UJxj328lHe; ds_user_id=145399310; rur=PRN; urlg

    headers = {
        "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_6) AppleWebKit/537.36 (KHTM
        "Cookie": cookie
    }

    request = urllib.request.Request(url=url, data=data, headers=headers, method="POST") # fine

    html = ""

    try:
        response = urllib.request.urlopen(request, timeout = 4)
        html = response.read().decode("utf-8")
        #print(html) # For display only
```

```
    except urllib.error.URLError as e:
        if hasattr(e, "code"):
            print(e.code)
        if hasattr(e, "reason"):
            print(e.reason)

    return html
```

It basically fakes the identity as a browser to exchange the header file during the network request process, by making use of the `urllib` package's URL request function, in this case HTTP POST is used. After looking into the HTML, I found that the relevant info, of the top 9 posts, is actually embeded in an extremely long JSON string, below attach a tiny fraction of it:

```
<script type="text/javascript">window._sharedData = {"config":{"csrf_token":"LqWwmIYutdVdjfZs19
```

Which means there is parsor mechanism built in the browser, or the JavaScrit on the page to interprate it, so that the page can't be loaded with information like number of likes and comments.

So instead of the whole page of HTML, let's trim the JSON string instead, below `get_tagpage_json` wraps the `get_content_url` function, then use the `BeautifulSoup` package to parse the HTML to locate to the desired keyword argument, the content under 4th `<script type ="text/javascript">` in this case.

```
def get_tagpage_json(tag_url):
    html = get_content_url(tag_url)

    soup = BeautifulSoup(html, "html.parser") # parser object

    list_scrpit = soup.find_all("script", type="text/javascript")

    jsonstr = list_scrpit[3].string[20:-1]

    return jsonstr
```

Putting this function in the main logic body instead of `get_content_url` makes the step 1 cleaner with 1 line code.

2.3.2 Parse the JSON string

So far, we have obtained the JSON string. In this step, we need to search and take only the portion we want.

Here we utilize `re` package which enables regular expression for searching in string. For example, we only need the part where involves the info of top 9 posts, which is behind keyword "edge_hashtag_to_top_posts", I chose to use `re.search` instead of `re.findall`, because it only matches the first occurrence of a pattern, but returns a weird match object. `.group(0)` used here to access the string (hard coded to remove some unwanted characters):

```
toppost_str = re.search('"edge_hashtag_to_top_posts":.*},"edge_hashtag_to_content_advisory', js
```

By the way, the total number of likes of a hashtag is not part of top9 posts info, the number is behind keyword "edge_hashtag_to_media":

```
total_like_str = re.search('"edge_hashtag_to_media":{"count":(\d+)', jsonstr).group(0) #
tottal_like_count : int = int(total_like_str.split(':')[2]) # Milestone
```

Also to make the JSON string readable, I save the top 9 posts info into a JSON file, `json.load` and `json.dump` are from `json` package:

```
toppost_dicts = json.loads(toppost_str)
with open('raw_top.json', 'w', encoding='utf-8') as jsonfile:
        json.dump(toppost_dicts, jsonfile, indent=4, ensure_ascii=False)
```

The fraction of the file looks like this, I save the file under the same working directory, but it's for viewing only, not really essential for the program:


```
{
    "edges": [
        {
            "node": {
                "__typename": "GraphImage",
                "id": "2486504591227683678",
                "edge_media_to_caption": {
                    "edges": [
                        {
                            "node": {
                                "text": "Setup for today! \nMaking my favourite Devices Float a
                            }
                        }
                    ]
                },
                "shortcode": "CKB1sBsgQNe",
                "edge_media_to_comment": {
                    "count": 44
                },
                "taken_at_timestamp": 1610634467,
                "dimensions": {
                    "height": 1080,
                    "width": 1080
                },
                "display_url": "https://instagram.fsin1-1.fna.fbcdn.net/v/t51.2885-15/e35/s1080
                "edge_liked_by": {
                    "count": 1062
                },
                "edge_media_preview_like": {
                    "count": 1062
                },
                "owner": {
                    "id": "362783038"
                },
                "thumbnail_src": "https://instagram.fsin1-1.fna.fbcdn.net/v/t51.2885-15/sh0.08/
```


I haven't implemented to store the desired info into variables then DataBase, I print to the console as of now, those keywords are pretty self-explanatory:

```
print("PostID\t\tLikes\tComments\tDate")
for dict in toppost_dicts["edges"]:
    print(dict["node"]["shortcode"] + '\t'
        + str(dict["node"]["edge_liked_by"]["count"]) + '\t'
        + str(dict["node"]["edge_media_to_comment"]["count"]) + '\t\t'
        + datetime.utcfromtimestamp(dict["node"]["taken_at_timestamp"]).strftime('%Y-%m-%d')
    ) # for display only
```

2.3.3 STRT Database manipulation

The creation of database should be done in backend instead of frontend. So far we have create one sample database called `explore` and a table called `hashtag`. The table attributes so far are just samples, we shall discuss more in the next section.

The database operation relies on mysql-connector-python package and its manual.

1. **Create database**
   First I wrote a function to create database and catch the exception if the database exists:

```
def create_database(DB_NAME):
    try:
        newdb = mysql.connector.connect(
        host     = "localhost",
        user     = "root",
        password = "TIC3901"
        )

        newcursor = newdb.cursor()

        newcursor.execute(
            "CREATE DATABASE {};".format(DB_NAME))
        print("New Database Created!")
    except mysql.connector.Error as e:
        if e.errno == errorcode.ER_ACCESS_DENIED_ERROR:
            print("Something is wrong with your user name or password")
        else:
            print("Preparing Database...")
            pass
```

   Note that the MySQL's user is `root`, password is `TIC3901`.
2. **Use/connect database**
   Create connection and cursor in the similiar fashion.

```
conn = mysql.connector.connect(
    host      = "localhost",
    user      = "root",
    password  = "TIC3901",
    database  =  DB_NAME,
    charset   = "utf8",
    collation = "utf8_general_ci"
)

cursor = conn.cursor()
```

3. **Create table (01)**
   The table attributes design is still up to change.

```
try:
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS `hashtag` (
        `name` varchar(50) COLLATE utf8_unicode_ci NOT NULL,
        `numPost` int(12) NOT NULL,
        `top9PostId` varchar(50) COLLATE utf8_unicode_ci,
        `createDate` datetime NOT NULL DEFAULT current_timestamp()
        )
    ''')

except mysql.connector.Error as e:
    print("Table error: ")
    print(e.msg)
    pass
```

4. `TODO` **Insert sample data**

   So far, sample data is tried here to fill the table for testing purpose, seems data is able to fill in but no exception handling yet for duplicate entry error.

```
add_tag = ("INSERT INTO hashtag "
                    "(name, numPost, createDate) "
                    "VALUES (%(name)s, %(numPost)s, %(createDate)s)")

# data set for test only, TODO: data feeder implementation
data_tag = {
    'name':       tag_name,
    'numPost':    tottal_like_count,
    'createDate': date(2021, 1, 1)
}

cursor.execute(add_tag, data_tag)

cursor.execute('''
    ALTER TABLE `hashtag`
    ADD PRIMARY KEY (`name`);
''')

conn.commit()
cursor.close()
conn.close()
```

`SELECT * FROM hashtag` in terminal `mysql` shows below table:

name	numLike	top9PostId	createDate
apple	35465424	NULL	2021-01-01 00:00:00
love	2000022138	NULL	2021-01-01 00:00:00
winter	147962789	NULL	2021-01-01 00:00:00

3 `WAIT` Database design

The important experience gained from this project is that we should probaly have done the database schema design at the very beginning. Once the tables and their attributes are fixed, backend and frontend can have better seperation without worrying affect each other.

The table design is still up to discussion, but I'd propose as of now:

Table 1: hashtag

name	numLike
apple	35465424
love	2000022138
winter	147962789

`name` should be primary key as each hashtag and the search page URL are unique.

Table 2: tag-toppost

tag_name	postId
coding	`CKQLGEVAo6e`

tag_name	postId
coding	CKTPbS6A0mg
coding	CKPFztCDsUO
coding	CKRtAtvgWFO
coding	CKRZ13ygtuw
coding	CKR7KFCFD7d
coding	CKSs0yfAciw
coding	CKRLDmaAxmR
coding	CKSwFaNAsWr

Table 3: toppost

postId	numLike	numComment	Date
CKQLGFVAo6e	8892	52	2021-01-20
CKTPbS6A0mg	2799	223	2021-01-21
CKPFztCDsUO	1426	14	2021-01-19

Above are the info I can crawl at the moment, I'm confident that I'm able to parse the JSON string, to get the comment section of each post and extract all the tags tagged along with the post.

Table 4: post_contain_tag

postId	tagname
CKPFztCDsUO	NULL
CJ1d_NZgli1	peoplewhocode
CJ1d_NZgli1	100daysofcode
CJ1d_NZgli1	buildupdevs

Not 100% sure if it is the right approach to find the relation to the other possible hashtags.

4 Future planning and challenges

4.1 KILL Login and maintain the session?

As mentioned above, because of the Facebook request limitation, we can't keep performing searches without login to Instagram. I included my own cookie string to the header file, but it's unlikely to work on other machines. One way is to have a login page in frontend to input Instagram username and password, then my python file can retrive the info to login to the site. From my preliminary research, to remain the state of login, `requests` package is needed to create a session. Another idea from Hui Yuan is that we could input a list of common hastags, my python program would iterate through and feed the data to Database, so users making searches in frontend have a high chance to get the existing result directly from database, instead of running python scrapper on demand.

4.2 WAIT Database design

To discuss on above schema design idea.

4.3 TODO Nautural language dictionary?

From the beginning of the project, I have been struggling to come up with anything to tell 2 hashtags having similiar meaning. Because we want to recommend to users a set of relevant hashtags if the one they chose was too popular. Or does the close meaning of the hashtags really matter much? For example, I saw post with #coding, also tagged along with #linux, #java, #python, #javascript and more. You can't say those are of close meaning or high relevancy. We are also unlikely to understand the content of what users are going to post other than the user input hashtag, so defining the relevancy is already problematic. So shall we just ultilise the last table or other measures as the consideration to determine the relation between tags?

5 Misc reflections

5.1 Easy way to start may not be the best way

There are 6 skill courses come with Industrial Prctice, one of them is to teach us using Jupyter Notebook (content is nice and useful). The first preparation was to install a software called Anaconda, which hosts the whole environment of what we need for the course. So it is very easy and neat as we don't have to get hands dirty to install python and so on.

At the time, I already spent quite some research time to install python properly via `pyenv`, and made the then latest python 3.8.5 and pip 3 as system default. Although I must admit that I'm lack of knowledge on virtual environments. I didn't know that Anaconda comes with python and it takes over as my system's default and the `conda` is another package manager.

So after the course, at certain point of time, my python program can't run properly due to can not find the packages that I import. It took me quite a few weeks to realize what was going on.

Even though some tools are easy to use for beginners, if we don't what happen underneath, it's difficult to diagnose the reasons of abnormal behaviors of the software system.

5.2 Docs writting

In report 01, I didn't know the Indusial Prctice website only takes pdf file for uploads. So I wrote the report in markdown and planned to upload the `.md` file, I'm sure there are softwares to render markdown beauifully on prof's computer. This time, I was planning to write LaTeX as the final format has to be pdf. However, HTML may still offer more flexible layout with CSS. So this is my first practice to use Emacs Org Mode, despite of the steep learning curve, it turns out to be fantastic.