Computing for Medicine – Database Systems
Periklis Andritsos
Faculty of Information, University of Toronto
periklis.andritsos@utoronto.ca

CREATING A DATABASE OF TWEETS OF MEDICAL TERMS

Introduction

Twitter has become one of the most popular sources of information. More than 60% of adults are estimated to read their news from this social media platform, while at the same time there exists a large number of applications that use its data to perform analyses. Apart from the typical user timeline, Twitter offers search a search engine and search results when user keywords are specified, via its search interface at https://search.twitter.com/ .

At the same time, to facilitate development of third-party application, Twitter offers an Application Programing Interface (API), to expose (part of) its functionality to software developers. For example, using the API, we can give specific keywords through a Python program and get as a result all the tweets related to the keywords.

In this project, you will be asked to create a database of tweets related to medical terms and perform different queries. The following sections will describe the different steps that you need to follow to accomplish the above.

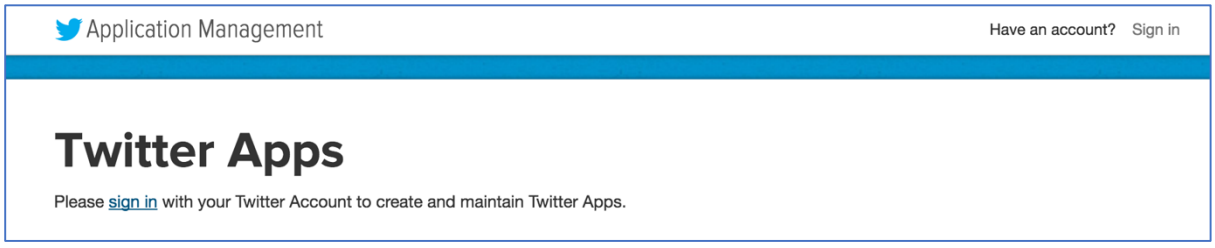
Readings:
http://www.journalism.org/2016/05/26/news-use-across-social-media-platforms-2016/
https://dev.twitter.com/rest/public

Twitter and its REST API

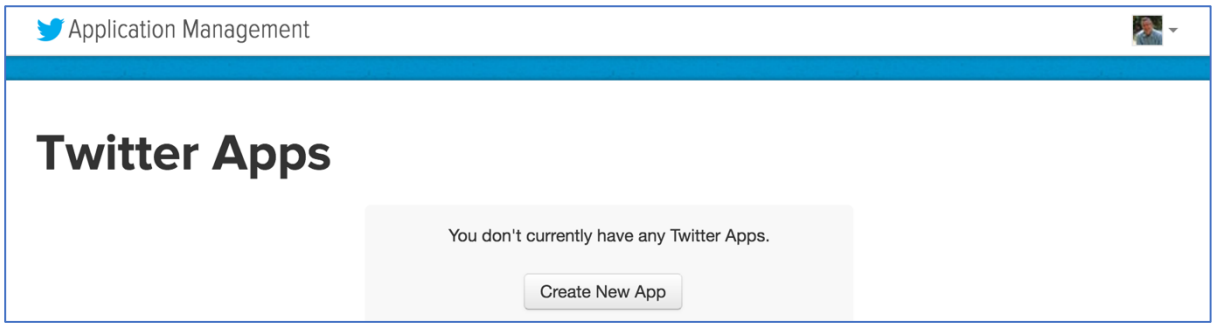
In order to programmatically access Twitter, you will need to use its REST API. The following subsections describe the initial steps that you will have to take.

Become a Twitter Developer

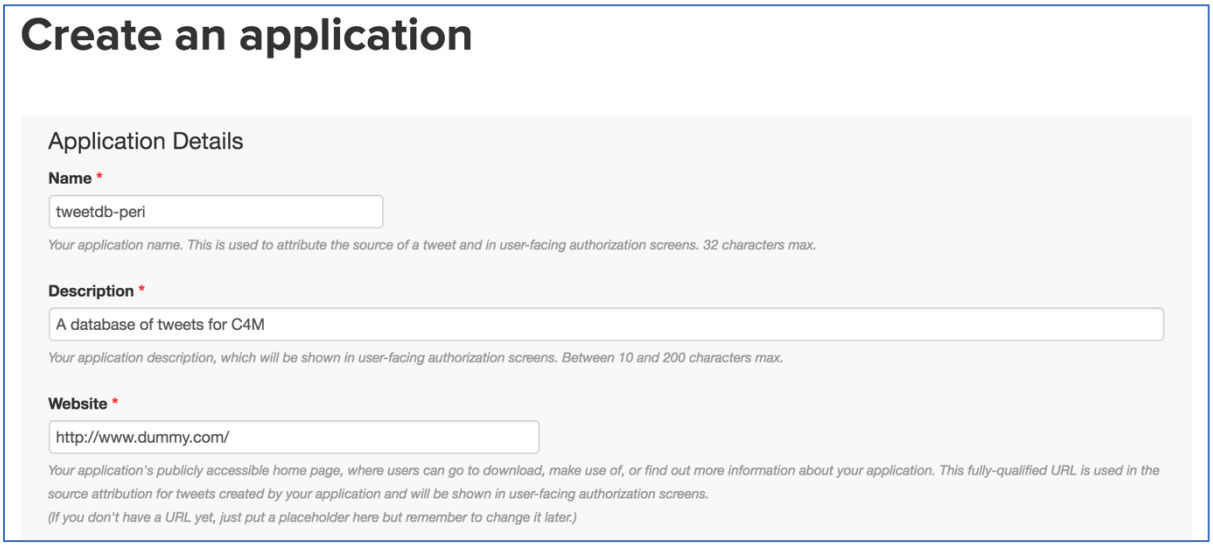
1. If you do not have a Twitter account, create one (mine is "`periklis13`")
2. Go to https://apps.twitter.com/ and log in with your username and password. This page will give us the necessary credentials to access the API.



3. Click on “Create New App”



4. In the next page, fill in the required fields (marked by *). If you do not have a Website, just fill in with a dummy URL.



5. In the following page, click on the Tab “Keys and Access Tokens”. At the bottom of this page, click “Create my access token”.
6. You will need to copy the following values: “API key”, “API secret”, “Access token” , ”Access token secret”. These four values are needed so that Twitter can authenticate you when you are accessing it via your program.
7. You will store these values in special variables:
    a. In the `examples` directory, you see them in `config.py`.
    b. In the `project` directory, you see them in `settings.py`.

Using ‘tweepy’ to get tweets

Accessing twitter can be a small nightmare since it does not allow frequent “hits” from user applications (see: https://dev.twitter.com/rest/reference/get/search/tweets for more information). Luckily, `tweepy`, a special python library, takes care of all that and permits easy access to the Twitter stream, i.e., the tweets that get posted in its public timeline.

You are going to use `tweepy` for authentication and for accessing public tweets that match your list of keywords. The directory examples contain a small tutorial file called `tutorial_tweepy_stream_api.py`.

In order to run the example, store your twitter credentials in the provided `config.py` file. The python program:

1. Authenticates the user.
2. Uses a `search_term` variable that includes a list of terms to be matched with the incoming tweets. In the example, `search_term=['flu']`.
3. Starts the `Stream` tweets using the `search_term` as parameter.

The `class StdOutListener(StreamListener)`, includes two methods.

1. Method: `on_status`: it stores all information about the tweets in the `status` structure. The different fields of the structure that are returned by the REST API, can be found here: https://dev.twitter.com/rest/reference/get/search/tweets
2. Method: `on_error`: it returns a flag (can be customized for error handling) in case of error.

When you run the above, you will see some of the contents of `status` printed in stdout.

The full `tweepy` documentation can be found here: http://docs.tweepy.org/en/v3.5.0/
Moreover, an interesting introduction to collecting twitter data can be found here: https://marcobonzanini.com/2015/03/02/mining-twitter-data-with-python-part-1/

Using `sqlite3` to create & query your database.

`SQlite` (https://www.sqlite.org/) is a popular, lightweight Database Management System that is very easy to integrate with python. The module that includes all necessary functionality is called `sqlite3`. To use the `sqlite3` module, you must first create a connection object that represents the database.

Here is an example:

```
        import sqlite3 as db
        conn = db.connect('test.db')
        print("Opened database successfully")
```

If you do not want to create a database on disk (e.g. `'test.db'` above), give the special name `':memory:'` and the database will be created in main memory.

To create a table for a company, you can write:

```
import sqlite3 as db
conn = db.connect('test.db')
print("Opened database successfully")
conn.execute('''CREATE TABLE COMPANY
       (ID INT PRIMARY KEY     NOT NULL ON CONFLICT IGNORE,
        NAME           TEXT    NOT NULL,
        AGE            INT     NOT NULL,
        ADDRESS        CHAR(50),
        SALARY         REAL);''')
Print("Table created successfully")
conn.close()
```

The connection (`conn`) should be closed when you are done.

Notice the PRIMARY KEY constraint declaration and the NOT NULL keywords used when NULLs are not allowed. The special keywords "`ON CONFLICT INGORE`" specify that if there are duplicates, the whole tuple will be ignored.

The database 'test.db' appears as a binary file in the directory from within which you run the script.

Upon creation, the new database is ready to store values. The following statements insert four new tuples to the COMPANY table. Notice the absence of single quotes when you insert numbers and their presence when there are strings.

```
import sqlite3 as db

conn = db.connect('test.db')
print("Opened database successfully")
conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (1, 'Paul', 32, 'California', 20000.00 )");

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (2, 'Allen', 25, 'Texas', 15000.00 )");

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (3, 'Teddy', 23, 'Norway', 20000.00 )");

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (4, 'Mark', 25, 'Rich-Mond ', 65000.00 )");

conn.commit()
print("Records created successfully")
conn.close()
```

Notice that after the INSERT statements, a `commit` command is given. It is VERY IMPORANT to be performed after any updates (inserts here) are performed, as it forces them to take effect, i.e. the data is updated on the disk.

Once the records are in the database, you can execute queries and get the results back into our Python program:

```
import sqlite3

conn = sqlite3.connect('test.db')
print(Opened database successfully)

cursor = conn.execute("SELECT id, name, address, salary  from COMPANY")
for row in cursor:
   print "ID = ", row[0]
   print "NAME = ", row[1]
   print "ADDRESS = ", row[2]
   print "SALARY = ", row[3], "\n"

print(Operation done successfully)
conn.close()
```

In order to get back the results, we use cursors (see '`cursor = conn.execute(…)`' in the example). A cursor facilitates subsequent processing, in conjunction with the traversal, such as retrieval, addition and removal of database records. Cursors are used to process individual rows returned by SQLite queries.

When the above is executed, we get:

```
Opened database successfully
ID =  1
NAME =  Paul
ADDRESS =  California
SALARY =  20000.0

ID =  2
NAME =  Allen
ADDRESS =  Texas
SALARY =  15000.0

ID =  3
NAME =  Teddy
ADDRESS =  Norway
SALARY =  20000.0

ID =  4
NAME =  Mark
ADDRESS =  Rich-Mond
SALARY =  65000.0

Operation done successfully
```

Inside the `examples` directory, you will find an example called `tutorial_pysqlite.py`, which showcases the above.

Putting it all together

You now have the tools to (a) access tweets, and (b) create and query databases. For the project of this C4M Workshop, you will be asked to create a Python program that takes as input several medical keywords, creates a database of tweets based on these keywords and outputs the results of several database queries. The queries will comprise simple selections, joins, aggregate functions, GROUP BYs and the use of special keywords, such as the LIKE keyword to match substrings inside textual values.

See some more example queries in SQLite here:
https://sqlite.org/
https://www.tutorialspoint.com/sqlite/

An outline of your final code is given inside the project directory.

→ Setting up

This concerns file 'settings.py'. You will need to specify:

- The authentication variables that you got from twitter when you followed the instructions at the beginning of this handout.
- The terms to be searched. The given file contains variables that need to be replaced with your own medical terms. You can search for the top medical terms used on twitter and use these.
- The Maximum number of tweets to be fetched from the stream.
- The name of you Database.
- The names of the two tables.

→ File '`project-sol.py`'

This is the skeleton of your solution. Inside this file, you will find several lines that start with

`## TODO`

Using the Python examples given in the `examples` directory you can learn all the preliminaries of authenticating and connecting to the database (in `__main__`). Inside function `initialize_tables()` your first task is to create the tables, whose schemas are given in the comments.

Upon creation of the tables the program will open the twitter stream and look for your keywords in the tweets that are being posted. It will download `MAX_TWEETS` and store them in a structure called `status`. This structure is rich in contents. You can visit: https://dev.twitter.com/rest/reference/get/search/tweets to get the full list of what information we can get through the twitter REST API. You have to set the variables that store different features of `users` and `tweets`, matching the two database tables that you just created in your database. We will use a cool function that can give us the sentiment of each tweet. Sentiment is a measure of whether a tweet is positive or negative. To do so, we convert the `t_text` variable (the actual tweet) into a Binary Large Object (or BLOB) and use the `sentiment.polarity` function that returns the polarity of each tweet.

Your next task is to write the INSERT statements (see the examples provided) in order to add each set of features into your tables. VERY IMPORANT, do not forget to perform a commit after each INSERT. NOTE: if you want to fetch more features, you are welcome to do so by changing the structure of the database tables as well as the INSERT statements.

Once the database is ready, we are giving you several SQL queries that we want to be answered and their results to be printed in `stdout` using Python. The definition of each query is given in the TODO comment of each function corresponding to a query.