GEN BUS 760: Course Project Description — *Fall 2021*

Project Summary and Grading

In this project, you will build a system to continuously score phrases and words based on their "trendiness" on Twitter. This accounts for 100 points, with partial points for each milestone.

Collaboration and Code Reuse

You will complete this project in groups (created by me on Canvas). You may consult and reuse code found online as long as the specific source code (i.e. links, filenames, and line numbers) is cited exactly where it is reused. You may not share code with other groups.

Fair Division of Work

You will create a private Github repository for your project and give me (emaadmanzoor) access to it. The first lab session will include a tutorial on using git and Github.

All members of the group must contribute to the project: this could be via coding, schema design, architecture design, perusing external literature and documentation, etc.. Evidence of these contributions must be present as commits on Github.

For example, if you are responsible for summarizing the relevant documentation of a specific system, you must commit a file to your Github repository (like docs/KAFKA_USAGE.md, for example) with your summary and findings. If you are responsible for coding a part of your project, you must commit that code to the Github repository yourself. These commits (quality, not quantity) will be used to quantify each group member's contribution to the project, and non-contributing members will be penalized.

Project Feedback

There will be a midterm report to be submitted according to the course schedule, and a final project presentation at the end of the course. You are encouraged to seek regular and specific feedback from me on your project by scheduling an appointment with me via email. You are also encouraged to seek help with lower level programming and debugging issues from the teaching assistant via email.

Linux Virtual Machine

The project must be developed on a Linux (Ubuntu 20.04) Virtual Machine provided by me at the start of the course. This ensures that I can replicate your work, that the software we will use is functional for students on any operating system, and that you do not mess up your own personal computers while working on the project (the virtual machine is fully isolated from your own computer). Virtual machine usage instructions will be shared on the course website. The first lab session will include a tutorial on the virtual machine and Linux.

"Trendiness Score" Formula

The trendiness of a phrase p at time t is computed as follows:

$$\text{Trendiness}(p, t) = \log10[\ \text{Prob}(p \mid \text{current minute at } t)\ ] - \log10[\ \text{Prob}(p \mid \text{minute prior to } t)\ ]$$

where *log10[x]* is the logarithm of x to the base 10. Prob(p | current minute at t) is the probability of seeing phrase p in current minute at t. For example, at t=2:45:30PM, the current minute at t starts at 2:45:30PM and ends at 2:45:59PM.

Prob(p | minute prior to t) is the probability of seeing phrase p in the minute prior to t. For example, at t=2:45:30PM, the minute prior to t starts at 2:44:00PM and ends at 2:44:59PM.

The probability of seeing a phrase is calculated slightly unusually as follows:

$$\text{Prob}(p \mid \text{current minute at } t) = \frac{1 + \text{no. of times } p \text{ was seen in the current minute at } t}{V + \text{total no. of phrases seen in the current minute at } t}$$

$$\text{Prob}(p \mid \text{minute prior to } t) = \frac{1 + \text{no. of times } p \text{ was seen in the minute prior to } t}{V + \text{total no. of phrases seen in the minute prior to } t}$$

where V is the number of **unique** phrases seen in the current minute or minute prior to t, respectively. This is add-one smoothing. Add-one smoothing ensures that the probability of a phrase that was *never* seen at all is not zero (but still very small). We can take the logarithm of this small probability without any issues, but the logarithm of zero is undefined.

To understand the intuition behind the trendiness score, note that it can be re-written as:

$$\log10\left(\frac{\text{Prob}(p \mid \text{current minute at } t)}{\text{Prob}(p \mid \text{minute prior to } t)}\right)$$

The logarithm here simply takes care of the fact that each probability in the numerator and denominator can be a very small number, and computers are inaccurate when performing arithmetic on very small numbers. Let's ignore the logarithm for a bit and look at this ratio:

$$\frac{\text{Prob}(p \mid \text{ current minute at } t)}{\text{Prob}(p \mid \text{ minute prior to } t)}$$

This ratio is the probability of seeing phrase p in the current minute at t *relative* to the probability of seeing the same phrase in the minute prior to t.

If this phrase had a sudden increase in popularity in the current 60 seconds, this ratio would be a large and greater than 1 (since the numerator would be greater than denominator). Hence, the phrase's trendiness score (the logarithm of this ratio) will also be large and positive. If the popularity of this phrase was almost constant over time, this ratio would be close to 1. The phrase's trendiness score (the logarithm of this ratio) would be close to zero.

Milestone I [25 points]

For this milestone, your code will simply read tweets from the Twitter API or a file and write them to a file on disk. You can think of this file on disk as your "data lake".

A. Read tweets from the Twitter API or a file, write tweets to a file [5 points]

Write Python code in **server.py** to read tweets formatted as JSON and store them on disk.

The code should be able to read from the Twitter API sampled stream or from a file containing the JSON-formatted tweets (i.e. provide an optional --filename flag to server.py using the **argparse** Python library that; if this flag is not present, then read from the Twitter API). You may assume that the tweets in the file will be in the same format as those from the Twitter API.

The code in server.py must parse out just the text and timestamp of each tweet and discard all other information. It must write then write each tweet's text and timestamp to **tweets.txt** in the following format:

```
tweet timestamp in YYYY-MM-DD-HH-MM-SS format, tweet text
```

The code must run continuously after it has been started (i.e. it should not stop when the file has been read, or when the API has stopped providing tweets), until it is halted by the user.

You may consult this tutorial from Twitter and the code therein to read from the Twitter API sampled stream using Python.

B. Compute frequencies of words and phrases [5 points]

Write Python code in **word_count.py** to take as input a word or multi-word phrase (use the *argparse* Python library and the --word flag to consume this input). Compute and print the frequency of that word or phrase in all the tweets stored in tweets.txt.

C. Compute the number of unique words (i.e. the vocabulary size) [5 points]

Write Python code in **vocabulary_size.py** to compute and print the number of unique words used in all the tweets stored in tweets.txt.

D. Document points of failure in server.py, implement code to warn and gracefully recover from such failures [5 points]

Document the points of failure and the recovery logic in **FAILURE.md**. Implement the recovery logic in the Python code you have written so far. Some examples of failure points are: losing access to the Internet, and running out of disk space.

E. Tag and release the code on Github [5 points]

Create a Github tag and release for the code on completion of this milestone. The snapshot of the code at this tag and release will be used to award the points for this milestone. Document how to run your code in **README.md**.

Evaluation Protocol: I will follow your README.md to run the code at this tag/release on my own virtual machine with a file of tweets I have created. I will compare the true word counts and vocabulary size in my file with those computed by your code. I will also test how your code crashes and/or recovers with different types of failures.

Milestone II [35 points]

For this milestone, you will transition your code to use a data warehouse (a PostgreSQL database) instead of a data lake. You will continue to read tweets from the Twitter API or a file, but will now write them to your database using Python.

A. PostgreSQL schema design and implementation [10 points]: Design a warehouse to store the information required to compute the most current trendiness score for any word or phrase (remember that the trendiness score of a phrase depends on the current time t).

Document the schema design in **SCHEMA.md**, with a short summary of the logic behind it. Write the SQL code needed to construct the tables under this schema in **schema_postgres.sql**. Include any additional details needed to recreate the database tables in README.md.

B. Read tweets from the Twitter API or a file, write the information needed to compute the trendiness score to PostgreSQL [5 points]

Write Python code in **server_postgres.py**, repeating subtask A of milestone I, but this time write the information required to compute the trendiness score to the PostgreSQL database you designed in the previous subtask.

As in subtask A, code should be able to read from the Twitter API sampled stream or from a file containing the JSON-formatted tweets (i.e. provide an optional --filename flag to server.py using the **argparse** Python library that; if this flag is not present, then read from the Twitter API). You may assume that the tweets in the file will be in the same format as those from the Twitter API.

The code must run continuously after it has been started (i.e. it should not stop when the file has been read, or when the API has stopped providing tweets), until it is halted by the user.

You may consult this tutorial from Twitter and the code therein to read from the Twitter API sampled stream using Python.

You may use the *psycopg2-binary* Python library to read/write from/to PostgreSQL.

C. Compute frequencies of words and phrases in the current minute [5 points]

Write Python code in **word_count_postgres.py** to take as input a word or multi-word phrase (use the *argparse* Python library and the --word flag to consume this input). Compute and print the frequency of that word or phrase in the tweets posted in the current minute. For example, if the time the script is run is 2:45:30PM, compute the frequency of the word/phrase in the tweets posted between 2:45:00PM and 2:45:30PM.

D. Compute the number of unique words in the current minute [5 points]

Write Python code in **vocabulary_size_postgres.py** to compute and print the number of unique words used in the tweets posted in the current minute.

E. Compute the trendiness score [5 points]

Write Python code in **trendiness_postgres.py** to take as input a word or multi-word phrase (use the *argparse* Python library and the --word flag to consume this input). Compute and print its most up-to-date trendiness score using the formula provided earlier in this document.

F. Tag and release code on Github [5 points]

Create a Github tag and release for the code on completion of this milestone. The snapshot of the code at this tag and release will be used to award the points for this milestone. Document how to run your code in **README.md**.

Evaluation Protocol: I will follow your README.md to first create the PostgreSQL tables using schema.sql, and then run the code at this tag/release on my own virtual machine with a file of tweets I have created. I will compare the true word counts, vocabulary size, and trendiness scores of a few words in my file with those computed by your code. I will also examine the commits from each group member to ensure fair division of work.

Midterm Report [10 points]

A [5 points]. Share a short writeup of your progress on the milestones so far (i.e. the status of each milestone and submilestone), the contributions of each team member, and note any technical and non-technical challenges you faced along the way. You may also provide suggestions to improve the project for future iterations of this course.

B [5 points]. Discuss your thoughts on using the "data lake" versus the "data warehouse" to produce up-to-date trendiness scores for any given phrase: which one would you prefer to deploy in a company interested in Twitter trends, and why?

Milestone III [30 points]

For this milestone, you will extend your code to use a streaming message queue (Kafka).

A. Read tweets from the Twitter API or a file, write tweets to a Kafka queue [5 points]

Write Python code in **server_to_kafka.py** to read tweets formatted as JSON and write them to a Kafka queue. Include any steps to setup the Kafka server/queue in README.md.

The code should be able to read from the Twitter API sampled stream or from a file containing the JSON-formatted tweets (i.e. provide an optional --filename flag to server.py using the argparse Python library that; if this flag is not present, then read from the Twitter API). You may assume that the tweets in the file will be in the same format as those from the Twitter API.

The code in server.py must parse out just the text and timestamp of each tweet and discard all other information. It must write then write each tweet's text and timestamp to Kafka.

The code must run continuously after it has been started (i.e. it should not stop when the file has been read, or when the API has stopped providing tweets), until it is halted by the user.

You may consult this tutorial from Twitter and the code therein to read from the Twitter API sampled stream using Python.

You may use the *kafka* Python library to read/write from/to Kafka.

B. Read tweets from Kafka, write the information needed to compute the trendiness score to PostgreSQL [5 points]

Write Python code in **server_from_kafka.py** to read tweets from Kafka and store the required information for the trendiness score computation in the PostgreSQL database you created for the earlier milestone. This code must also run continuously.

The code must run continuously after it has been started (i.e. it should not stop when the file has been read, or when the API has stopped providing tweets), until it is halted by the user.

You may use the *psycopg2-binary* Python library to read/write from/to PostgreSQL.

C. Continuously compute the most up-to-date trendiness score of a word/phrase [5 points]

Write Python code in **trendiness_kafka.py** to take as input a word or multi-word phrase (use the *argparse* Python library and the --word flag to consume this input). Compute and print its most up-to-date trendiness score using the formula provided earlier in this document. As long as this code runs, at each new minute, it should print the most up-to-date trendiness score.

D. Presentation on system architecture, failure resilience, live demonstration [15 points]

Present your overall system architecture and operational logic, what its different points of failure are, how it will react to each of these failures, and a live demonstration of the continuous trendiness score computation for any one word or phrase.

Evaluation Protocol: I will follow your README.md to replicate your live demonstration. I will also cross-examine each group member during the presentation to better understand the proposed system and each member's contribution.