205 - Assignment 3 – Kasane Utsumi

List of Items to turn in:

1. A link to your S3 bucket that holds the backups documented in your README.md file. Make sure to make it publicly accessible.

   Link: https://moonlightbucket.s3.amazonaws.com/. Please examine files stored under "db_tweets" and "db_streamT" folders.

2. Your python codes.

3. The plot of your lexical diversity in task 2.2 and the result of the sentiment analysis in task 2.4 if you complete the bonus par

   Please look in this folder: 2_2/ Lex_div_histo.png

Documentation

Note: Python file and other relevant files are in corresponding folder with same number as a problem number (i.e. Please look in "1_1" folder for 1.1)

1.1

1. 1_1_store_tweets_into_db_streamT.py

This code retrieves tweets by search term ""#microsoft OR #mojang" specified by date range and dumps them into db_streamT%StartDate%(replace '%StartDate%' with start date that was passed as a command arg) collection. Type below to run the code.

python 1_1_store_tweets_into_db_streamT.py "startdate" "enddate"

As I explained in the previous assignment, since I had numerous issues (i.e. "KILLED" error) with retrieving entire week worth of tweets in one execution, I had to introduce a flexibility in my program to retrieve tweets one day at a time. The commands had to be ran 7 times to retrieve the tweets between "2015-03-03" and "2015-03-09"

2. 1_1_concatenate_into_streamT.py

This file concatenates contents of db_streamT%StartDate% (%StartDate% is specified by command argument) into db_streamT. It takes seven arguments.

python 1_1_concatenate_into_streamT.py “2015-03-03” “2015-03-04” “2015-03-05” “2015-03-06” “2015-03-07” “2015-03-08” “2015-03-09”

That said though, a better design would have been simply have ”1_1_store_tweets_into_db_streamT.py” keep appending tweets in json to db_streamT as new tweets are retrieved. Then, concatenation would have been unnecessary.

1.2

1_2_fill_db_tweets.py

This code iterates through all tweet jsons from db_streamT and stores tweet text ONLY into db_tweets collection.

2.1

1. 2_1_get_top_30_retweet.py

This file first iterates through db_tweets collection to find any tweet that starts with “RT”, indicating that the tweet is probably a retweet. Then, it will look for corresponding tweet json in db_streamT. If a corresponding tweet json includes a node called “retweeted_status”, which verifies that the tweet is a retweet according to Twitter API documentation. Then, “retweeted_status” node is inserted into the “db_retweets” collection (no duplicate is allowed in this collection). At the same time, “id” attribute within retweet_status node is stored as a key in the “retweetDict” dictionary. Value in “retweetDict” dictionary is how often the tweet has been retweeted. Then, “retweetDict” is converted to a tuple so it can be sorted by the dictionary value (the frequency) in descending order (from top down). For the first 30 entries in the tuple, retweeted text and the user information (name and location) are retrieved from db_retweets and are stored in “final_top30_retweets.txt”

That said, this particular requirement could have been fulfilled just by looking for retweeted_status node in the db_streamT collection. Db_tweets did not have to be referenced at all.

2.2

1. 2_2_lexical_diversity.py

   This code groups db_streamT by users first. Then, it computes number of unique words and number of words in all of tweets by each user, which is used to compute lexical diversity, which in turn is stored in db_lexical_diversity collection. The structure of db_lexical_diversity is:
   {"id": (userId), "username": (username), "lexical_diversity" : (lexicalDiversity float)}

   In calculating lexical diversity, the below words were omitted since these are necessary words to form an English sentence. Using these words often should not lower the lexical diversity of users.

“the”, “a”, “from”, “to”, “and”, “for”, and URLs (starts with https:// or http://)

2. 2_2_plot_lexical_diversity.py
   This code generates a histogram which shows frequency of lexical diversity range based on db_lexical_diversity collection.

3. Lex_div_histo.png
   This is a histogram of lexical diversity for all users. I thought histogram is more interesting than actually plotting the lexical diversity of each user, since the histogram can tell us how linguistically competent in general are these users that I have collected.

2.3

1. 2_3_get_followers_for_top_retweets.pyt
   Using Twitter API, this code retrieves a list of followers (id) for users who tweeted the top 30 retweets which were stored in top30RetweetedUsers collection in 2.1. The follower ids for each top retweeted user are stored in db_followers collection.

   Since some of the users in the top30RetweetedUsers collection had large amount of followers (with one more than 3 Million), given the time constraint I was unable to get follower ids for all 30. I was able to retrieve the follower ids for only 17 out of 30 users. In addition, there was a bug in my code in which 17 users whose followers I was able to obtain were not first 17 users in the top 30 retweeted users list (when ordered by retweet count descending). If I were to re-implement this, I will add sorting by retweet count while retrieving users from top30RetweetedUsers collection.

2. 2_3_followers_after_week.py
   This code iterates through first 10 users in db_followers and retrieves followers for the same users after a week, then stores the result in the db_followers_after_week collection. Again, as I explained in the previous paragraph, the users whose followers were retrieved were not first 10 users in the top 30 retweeted users list. Also, due to the time constraints, these followers were retrieved only two days after the first time followers were retrieved.

   If time allowed, I would have put the common functionalities between this code and 2_3_get_followers_for_top_retweets.py into a common file.

3. 2_3_get_unfollowers_after_week.py
   This code compares the list of followers for a given user between db_followers and db_followers_after_week and get ids of followers who were no longer following the user after a week. Then it calls Twitter api to retrieve information about each of the “un”followers and stores them in the db_dropped_followers collection. Then it iterates through db_dropped_followers to print out id and name of “un”followed users for each user.
4. Log_unfollowed_user_list.txt
   These are the “un”followed users that were printed out at the end of #3 above.

3.1

1. 3_1_store_backup_to_S3.py
   This code make a backup of db_tweets and db_streamT and uploads to a corresponding location on S3. The items in db_streamT are bundled into 500 per json file(value), and for db_tweets 1500 per csv file(value). TwitterSerializer.py is a file class used to facilitate bundling tweet json from db_streamT into a file.

2. 3_1_restore_backup_from_S3.py
   This code restores backup files in S3 made from db_tweets and db_streamT and restores the data into db_tweetsRestored and db_streamTRestored collections.

More Possible Design Enhancement

1. In general, I should have created a common ".conf" file where all of the authentication information (i.e. for S3 and twitter) are stored.