Alex Smith
MIDS W205

ASSIGNMENT #2

OVERVIEW

I wrote tweetsearch.py to pull tweets over a week long period that contained two hashtags: #NBAFinals2015 and #Warriors. My analysis categorized tweets into three buckets: (1) those only containing the first hashtag, (2) those only containing the second hashtag, and (3) those containing both hashtags. The program used Twitter's REST API rather than the Twitter firehouse. I found the following distribution of tweets for my hashtags:

#NBAFinals2015 only: 72482
#Warriors only: 81659
#NBAFinals2015 & #Warriors: 3631

ARCHITECTURE & DESIGN

The program can be run by simply typing "python tweetsearch.py" into the command line. To search custom queries, the user simply has to modify the fields "QUERY1" and "QUERY2". The user can also modify range of tweets searched by changing the "START" and "END" fields. I chose to use constant variables rather than hard-coding this information into my code to make it easy for future users to modify my basic assumptions.

The program requires importing a number of modules. We list the modules here and explain their purpose in comments in the code: sys, urllib, datetime, boto, json, nltk, pickle, tweepy, signal, threading, and time.

The architecture of this program is designed around the main file, tweetsearch.py. The program calls tweetacquire.py using date and query parameters. tweetacquire.py uses the Tweepy module to download tweets and calls tweetserialize.py to write the tweets to json files. The program chunks files based on the date of the tweet and the search term. I chose this design because it allows for an intuitive grouping of tweets. Although some files are large, all can be easily and quickly opened with a simple text editor, like TextWrangler.

tweetsearch.py then calls tweetdictionary.py to create a dictionary storing all the tweets by their tweet id before calling tweetcount.py to loop through the dictionary and count the instances of tweets that fall into the three buckets specified in the overview. tweetdictionary.py also saves the dictionary as a pickle file. I chose to save the file as a pickle file to make it simple and easy to import into Python for future analysis.

tweetsearch.py then calls histogram.py that creates three CSV files, one for each bucket specified in the overview. These files contain the each word and the number of instances the word appeared in tweets. I chose a CSV file rather than a histogram of the top 30 so that users would have a more precise understanding of the tweet makeup. tweetsearch.py concludes by calling tweetS3.py to upload the pickle file to Amazon S3.

RESILIENCY

This code is designed to be resilient to exceptions from both software and the user. The code uses the threading module to ensure that if the user aborts the program, the program will still finish the tweet it is writing. This prevents the data from corrupting. The code handles Twitter's rate limits by using the Tweepy module and waiting every time the limit is reached. Often, Twitter will close a download connection if it has been running for too long (even if the rate limit has not been reached). This code handles this exception by putting the downloading of tweets in a try function and waiting for 1,000 seconds if Twitter closes the connection.

OUTPUTS

The code produces a number of outputs. First, it downloads the tweets as json files and chunks them by query and by date. Additionally, the program outputs the histogram distribution of words in tweets into three separate files, based on the buckets specified in the overview. These files are CSV files for easy analysis in spreadsheet programs, like Microsoft Excel. Finally, the program produces a file, "tweet_output.p" that is a pickle file of all tweets by tweet id. This file can also be found here:

https://s3-us-west-1.amazonaws.com/keivahn-w205-assignment2/9f8693fc49d5f1e093dc94ef7d093d3a