Group 6

Soroosh Safari Loaliyan, Mehrnaz Ayazi,
Javad Saberlatibari, Manny Argueta
University of California, Riverside
CS 242, Information Retrieval and Web Search

Phase 1

Each member contributions are as follows (based on the priority):

Crawler: Mehrnaz Ayazi, Javad Saberlatibari, Soroosh Safari Loaliyan and Manny Argueta

Indexer: Soroosh Safari Loaliyan, Mehrnaz Ayazi, Javad Saberlatibari and Manny Argueta

Documentation: Javad Saber Latibari, Manny Argueta, Soroosh Safari Loaliyan and Mehrnaz Ayazi.

Introduction

Twitter has a plethora of information that can be retrieved using Twitter streaming APIs. We decided to use collect our data in the form of user tweets. To do this, we crawled Twitter using Tweepy to collect tweets in English until we reached our limit of 250MB of data. This Twitter streamer is not limited to a particular subject, but rather is allowed to collect general data starting with a specific user. The application of this streamer is versatile and may be modified later to include only desired tweets from a particular subject or user. The data, once crawled, is indexed by Lucene and will be available for further use later in the process. Lucene manages an index over a dynamic collection of documents and provides very rapid updates to the index as documents are added to and deleted from the collection. An index may store a heterogeneous set of documents, with any number of different fields that may vary by a document in arbitrary ways.

Crawler

There are a few essential functions that need to be considered for crawling Twitter. Among them, the crawler needs to be able to authenticate a user in order to crawl, the listener needs to handle tweets from an incoming stream and establish a connection to the API, and the API needs to be in place to provide methods for various parameters. These provide a basic architecture for our crawling system, and we use the API to make adjustments for our project's purposes. In order to develop the tweets crawler, we exploit from phyton.

Requirements before running code:

- Applying for the authorization token and key in developer tweeter accounts.

- Installing the required packages and libraries including:
    - Numpy: A Python library which supports large and multi-dimensional arrays and matrices and their required high-level mathematical functions. (Numpy installation with *pip*: NumPy)
    - Tweepy: A Python library for accessing the Twitter API. (Tweepy installation with *pip*: Installation — tweepy 3.10.0 documentation)
    - Pandas: A Python library which provides easy-to-use data structures and data analysis tools. (Pandas installation with *pip*: Getting started — pandas 1.2.2 documentation (pydata.org))

Implementation of our Crawler:

In the following section, we will briefly describe our crawler in each part:

- **Import:**

- **From tweepy.streaming: import StreamListener**

By importing the *StreamListener* from *tweepy*, we can handle the live stream of tweets.

- **From tweepy: import OAuthHandler and import twitter_credentials**

By importing *OAuthHandler* and *twitter_credentials* files, and reading the keys and tokens from the file which is beside the main crawler code (in the same directory), we can access to live stream of tweets.

- **Body**

- **get_friend_list definition**

In this definition, we can find and extract the friend list of the user.

- **recievTweets definition**

In this definition, we aim to access the tweets information from tweeters clients. In this part, by using the API of the tweeter, we get the live stream of the tweets. First of all, we check the *id* of each tweet, in order to find if this tweet has been crawled in the previous execution or not. Therefore, we can prevent creating duplicated tweets. If this tweet has not been crawled yet, we try to handle the extracted live stream of tweets by saving it in the text or *json* file.

- **data_frame definition**

In this part, we seek to provide a frame in order to keep the extracted tweets in the file and also to create a legible input for the next indexing step. Each frame consists of comprehensive information of the crawled tweets (tweet id, screen name, name of the user, the location of the tweet, description, number of the user followers, length of the tweet, date of the tweet, number of likes, and number of the retweets).

Example of Json output file

[{"tweets":"\"What's been stopping Gronk is Tampa Bay's need to run a certain offense in order to protect Tom Brady\"\u2026 https:\/\/t.co\/Z3C6jfXyRY","id":1357776669101195280,"screen_name":"PatMcAfeeShow","name":"Pat McAfee","location":"Indianapolis","description":"Award winning wrestler. Kicked off a SuperBowl. Tossed bowls to Red Panda. LIVE M-Friday Noon-3est on YouTube & @siriusxm. @FDSportsbook\u2019s cool.","followers_count":2015738,"len":128,"date":1612554158000,"source":"Twitter Media Studio","likes":96,"retweets":8}]

- **Our strategy of crawling**

In general, our strategy of crawling is “starting from one user and recursively crawling the user’s tweets, then going for the user’s friend list and their tweets”.

- **Multi-threading**

In order to improve the performance of our crawler, we can use from multi-threading. Multi-threading enables us to add more tweets to the queue, so the crawler having more data to extract. We added several lines of code about this matter to the end of our crawler.

Indexer

We develop our indexer by exploiting the Java-based Lucene. The input of our indexer is the *json* files which are the output of our crawler. In the following sections, we will briefly describe each part of our indexer.

- **TextFileFilter.java**

Since our indexer accepts the *json* files only as inputs, we put a filter on all the indexer inputs to determine the type of the input files in the *TextFileFilter.java*.

- **Searcher.java**

In order to easily test and evaluate the indexer, we developed *Searcher.java*. The reason is by proving “search in indexed tweets”, we are able to validate the correctness of our indexer functionality.

- **LuceneConstants.java**

The constant values which are required by our indexer to correctly execute are in *LuceneConstants.java* including file name, file path, and the maximum number of searches.

- **Indexer.Java**

The main implementation of our indexer is in *Indexer.java*. At first, it selects a field from the list of *json* files. Then in the main loop, the indexer inserts a document based on the field type, e.g., tweets, name, screen_name, etc.

Indexer uses StandardAnalyzer, Standard analyzer use StandardTokenizer, LoweCaseFilter and StopFilter (The default list of stop words for Standard analyzer is {"a", "an", "and", "are", "as", "at", "be", "but", "by", "for", "if", "in", "into", "is", "it", "no", "not", "of", "on", "or", "such", "that", "the", "their", "then", "there", "these", "they", "this", "to", "was", "will", "with"}) to parse and index documents.

- **LuceneTester.java**

Our main function that calls the Indexer and Searcher is in *LuceneTester.java*.