CS 242 Project

Crawler & Lucene (Part A)

Project Members:

Student ID	Name
sgupt068	**Sheenam Gupta**
hsang006	**Hanisha Sree Sangam**
hsrin004	**Harish Kanna Srinivasan**
pvelm001	**Ponmanikandan Velmurugan**

Collaboration Details

We as a team decided on retrieving twitter data based on a particular geographical location, since New York is the most happening place in the world for tweets so our project is based on tweets in and around 2000Km around New york.

Sheenam Gupta has worked on writing the script and collecting the data from twitter using Tweepy . Sheenam and Ponmanikandan collaboratively provided their valuable inputs in resolving the issues while crawling the twitter. Ponmanikandan, Hanisha and Harish have worked on indexing the data using Lucene. We all discussed and came up with a project map that includes an approach which involves everyone to read and collaborate on a weekly basis.

Crawling Overview

1. **Libraries Used :-**
    a. Tweepy - A python library for accessing the twitter apis. Twitter Streaming api is used from this.

b. CSV - CSV module is used to generate the output in a CSV file. It implements classes to read and write tabular data in CSV format.
c. JSON - Json module is used to decode the data retrieved from twitter and make it accessible by CSV.
d. Argparse - argparse is used to parse the user inputs (i.e location coordinates and file size)
e. Datetime - datetime is used to convert the input date from twitter into desirable format

2. **Crawler Overview :-**
    a. Tweepy is used to crawl the data from twitter. The Twitter streaming API is used to download twitter messages in real time. It is useful for obtaining a high volume of tweets, or for creating a live feed using a site stream or user stream.
    b. The streaming api is quite different from the REST api because the REST api is used to pull data from twitter but the streaming api pushes messages to a persistent session. This allows the streaming api to download more data in real time than could be done using the REST API.
    c. Python file is written and is run on a terminal which executes and starts producing output as a csv file.
    d. The "main" function in the python file is called as soon as the script is started on the terminal which calls MyStreamListener class.
    e. In Tweepy, an instance of tweepy.Stream establishes a streaming session and routes messages to a StreamListener instance. The on_data method of a stream listener receives all messages and calls functions according to the message type.
    f. In my script ,MyStreamListener class has all the functions defined in it which will be used for various options while running the file:-
        i. **Login Function :-** It handles all the auth of the tweepy by passing the consumer and secret keys and also sets a access token using access token and access token secret keys to connect to twitter streaming Api. This function is also called in main function in case the streaming connection gets disconnected.

ii. **Exception Function:-** This handles all the exceptions coming from tweepy, prints the exception and returns false to terminate the function.

iii. **Error Function :-** This handles all the errors coming from tweepy, prints them and returns false. For example error code 420, it comes when the stream is disconnected in case of a high rate limit. False is returned in this function in order to disconnect the stream in case of heavy load, code for reconnecting the stream is written in the main function in order to reconnect the stream.

iv. **Data Function :-** This is the most important function given by tweepy. All the data of the tweets retrieved by the streaming api is retrieved in this function. Data is first decoded and made it compatible to access in order to pick the various columns from the data. All the data is not picked as it is from a tweepy response as many parameters were null and empty and were not of any use. Particular parameters which will be needed in our search indexing our retrieved for instance Tweet time, tweet text,profile pic of user etc. Tweet text and profile pic are made available in user readable form by replacing extra unwanted characters with spaces while retrieving the data. Optimised checks are applied for storing empty values in case tweepy returns null or does not return a parameter.

g. File is opened in append mode at the top of the script with a name “Ouput.csv” and a newline is given for creating a new line for every tweet. CSV heading are defined and passed in a dictionary and headers are written in csv file before the “data function” is called.

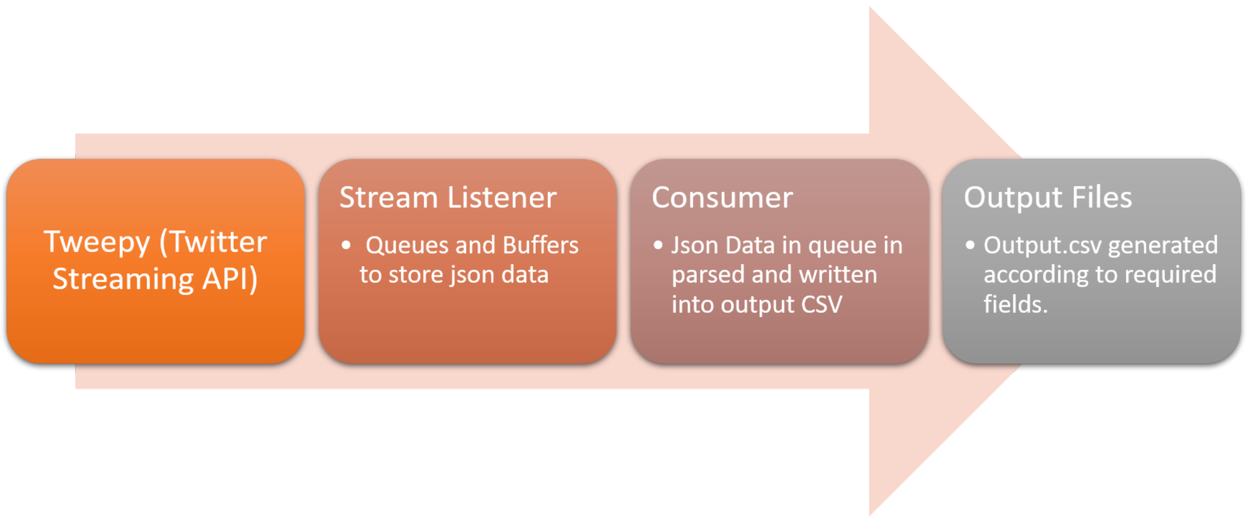
h. **Script is made by taking care of user experience, the location and the size of the file is asked from the terminal. Whichever location the user will enter and whatever file size the user will enter in the terminal that data will be retrieved, so the user does not have to make changes in file for retrieving data of different locations.**

i. File size check is applied in “data function” that means when file will reach the size as given in the input, the script will get terminated. Default file size is kept to be 65 mb, so whatever number will user enter that will be multiplied with 65 mb, so lets say user enters 4, the output file be 65*4 = 260 mb

j. Data is filtered based on location region around New york city and language english parameter is also passed, so that all the tweets are retrieved in English only.
k. “Wait on rate limit” and “Wait on rate limit notify” parameters are used so that the program runs within the rate limit defined by Twitter. This means the program will pause a few seconds every few seconds, so it's not long. By using these 2 parameters streams can run for as long as we want without being shut down by twitter.
l. **Worst cases for streaming disconnecting are handled, one by passing the parameters like “Wait on rate limit” in the streaming api and the other by disconnecting the stream in case the api returns 420 error of overload and connecting the stream again in main function.**
m. Once the stream is run the “Data function” is called to get the tweets and write in the CSV file.

3. **Crawling Strategy :-** Twitter allows only a single connection for each application. This denies the ability to create multiple connections to speed up data collection. However, due to the additional processing that needs to be done to convert stream data, steps have been taken to ensure no data is lost and each Tweet gets processed. All Tweets gathered are stored inside of queues that serve as buffers. From these queues, multiple consumers continually monitor and process data.

4. **Crawler Architecture :-**


Tweepy (Twitter Streaming API)
Stream Listener
• Queues and Buffers to store json data
Consumer
• Json Data in queue in parsed and written into output CSV
Output Files
• Output.csv generated according to required fields.

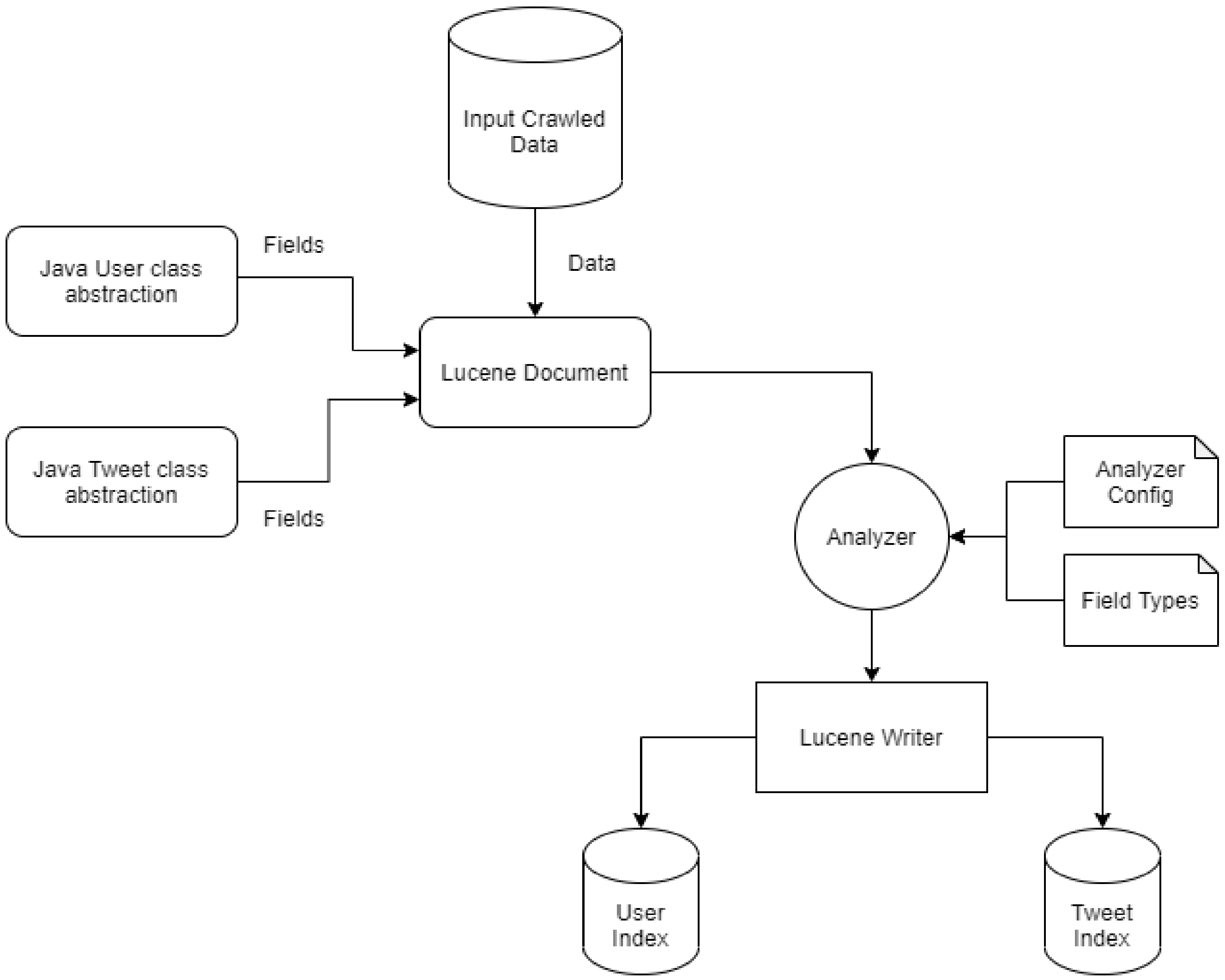
Lucene Overview

Lucene is an open source full text search suite provided by apache. We used IntelliJ IDE and Java programming language for lucene implementation. The crawled data is stored and read from Excel Sheets throughout the process. Our goal is to efficiently index tweets, in order to achieve the strategy used is :

Crawled data is indexed data into User Index and Tweet Index sharing the common reference field as User Id. We observed the following:

- ❖ In the case of large dataset, This method eliminates duplicate users to be indexed.
- ❖ It provides faster retrieval methods for user related queries.
- ❖ It helps in user related question answering queries.

Process Flow of Indexing



User Fields and Tweet Fields used in the project are :

User Fields	Tweet Fields
User Id	
User Name	TweetId
User Description	Tweet Hash
User Friends	Tweet content
User Followers	Tweet URL
User IsVerified	Tweet Likes
User Location	Tweet Miscellaneous(TimeStamp,Coordinates)

We used various fields and analyzers to further optimize our indexing process.

Lucene Fields

The fields include :

String Field: A field that is indexed but not tokenized: the entire String value is indexed as a single token. String field has a limit of 255 characters.

Text Field: A field that is indexed and tokenized, without term vectors. For example this would be used on a 'body' field, that contains the bulk of a document's text. Text field has a character limit of 30,000 characters.

Long Point: An indexed long field for fast range filters. If we also need to store the value, we should add a separate StoredField instance.

NumericDocValuesField: Field that stores a per-document long value for scoring, sorting or value retrieval. If we also need to store the value, we should add a separate StoredField instance.

Fields	Indexed Variables	Justification
StringField	user_name, user_verified, user_location, tweet_hashtags, tweet_url, misc_timestamp, misc_coordinates	We are using user variables to better structure the tweet with user accounts.
TextField	user_description, tweet_content	This is the key variable used to retrieve tweets.
LongPoint	user_friends, user_followers	This is a additional variable used to make range queries about user

NumericDocValuesField	user_id, tweet_id, tweet_likes	We are using these key values to reference each tweet and user with a unique number.

Lucene Analyzer

We used a Per Field analyzer because it provided us with customization for every field. I.e, **Every field can be mapped to an analyzer that better suits it's behaviour**.

The analyzers include :

Standard Analyzer: It can recognize URLs and emails. Also, it removes stop words and lowercases the generated tokens.

English Analyzer: It consists of StandardTokenizer, StandardFilter, EnglishPossessiveFilter, LowerCaseFilter, StopFilter, and PorterStemFilter.

Keyword Analyzer: It is useful for fields like ids and zip codes.

By using Per Field Analyzer, we are mapping the variables as follows,

Analyzer	Indexed Variables	Justification
Standard Analyzer	tweet_url	Standard analyzer has the capability to recognize URL
English Analyzer	user_description, tweet_content	English analyzer can remove stop words, do stemming, and tokenizing.
Keyword Analyzer	user_id, user_name, user_friends, user_followers, user_verified, user_location, tweet_id, tweet_hashtag, misc_timestamp, misc_coordinates	Keyword analyzer can store the entire data as a single token. This is better suited for ID's and numbers.

Limitations

Steps have been taken to optimize the crawler data manipulation. Protection against program crashes has not been rigorously tested. As the "wait rate limit" parameter is used which pauses every few seconds to maintain the twitter guidelines, this ends up taking some additional time to get 250 mb output data.

While indexing, the sole limitation we observed is that, only a single Index Writer is active at a same instance of time.

Obstacles and Solutions

For Crawling :

Obstacle :- The hard part in the twitter crawler was to collect the 250 mb data, as we were running the scripts on our personal systems so it took us few days to collect the data from twitter, also streaming api response depends on the numbers of tweets in real time, so sometimes the api used to return very slow response especially at night time when people were not tweeting much.

Solution :- We made a script which was accessible by all, that means by entering the location and file size in the terminal itself, anybody can run the script on his system. There is an additional option to enter multiple locations as well in the terminal So two or three people from our group ran this script parallelly for the New York region by entering different boundary values or coordinates in a proximity of 2000 km range of New york.

For Indexing :

Obstacle : Initially, we tried to use a CSV file(comma separated files) but it was memory bound. The larger the data, higher the run time and in most of the cases OutOfMemoryBuffer Exception aroused when reading large CSV files.

Solution : Instead of a single file, we split the dataset into smaller .xlsx files which has also helped in giving faster results.

Instructions to run the crawler

Python and tweepy should be installed in the system. Download the file in a directory and in the terminal write python filename --locations -124.7771694 24.520833 -66.947028 49.384472 -164.639405 58.806859 -144.152365 71.76871 --num 1. Here locations are the boundary values for which you need to scrape the data and the “1” after number tells the size of the file you want , that means if you pass “1” it will generate a file of 65 mb , if you pass “2” after number it will generate a file of 65*2= 130 mb. In the above example I have entered 2 locations,each location needs 4 parameters which can be retrieved by bounding boxes .Script will start crawling the data in your current directory where you ran the script.

Instructions to run the Indexer

Java and IntelliJ should be installed in the system. Ensure the crawled data is stored in the runtime directory. Pass the file name of the crawled data in the LuceneIndexer.java code. Now run the main function of the LunceneIndexer code. The indexed data will be stored in two directories as user_index and tweet_index which has its respective indices.