CS 172 Final Project

Risul Islam (862004316), David Swanson, Jhaymer Sabino

Introduction

Due to scheduling challenges, this project evolved into related but separate implementations of the assigned work. It includes both tasks:

- Twitter Search with Map Locator
- Web Search with URL and Document Scraping
- Both implementations use Elasticsearch with a Python client for the index and retrieval portion.

Twitter Search by Risul Islam and Jhaymer Sabino

https://github.com/CS172-UCR/finalproject-hookedonpythonics/tree/master/twitter_search

The goal of this project is to implement a search engine using different built in packages like tweeter stream collection and elastic search using Lucene. We have indexed using Lucene and showed the search result using default BM25. Also we took the search keywords in a nicely decorated UI and showed the search tweets in google map centered at riverside (mainly centered at query location in a radius). The project gave us an overall view of how search engine works.

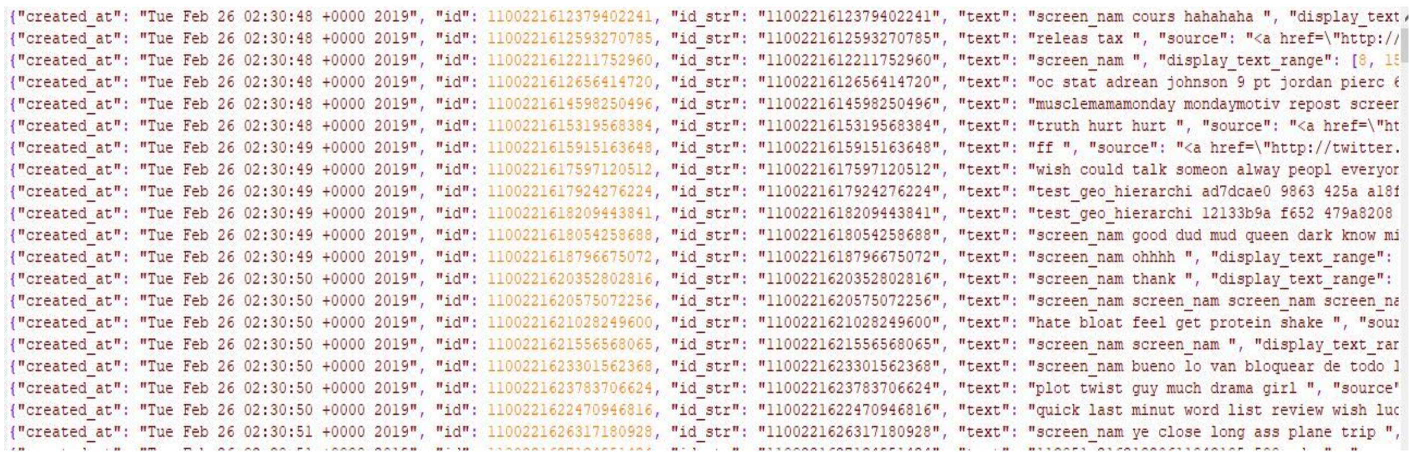
Task Completed in Twitter Search:

- Tweeter stream collection
- Preprocessing of the data
- Indexing using Lucene
- Search using lucene (Elastic search)
- GUI implementation
- Showing the result in Google map

Tweeter Stream Collection:

1. Collected tweeter stream in real time
2. Used python tweepy package
3. Implemented filtering in tweepy based on keyword and location. So, the data is composed of variety of keywords and from variety of locations.

A sample of raw data is given below:



```
{"created_at": "Tue Feb 26 02:30:48 +0000 2019", "id": 1100221612379402241, "id_str": "1100221612379402241", "text": "screen_nam cours hahahaha ", "display_text
{"created_at": "Tue Feb 26 02:30:48 +0000 2019", "id": 1100221612593270785, "id_str": "1100221612593270785", "text": "releas tax ", "source": "<a href=\"http://
{"created_at": "Tue Feb 26 02:30:48 +0000 2019", "id": 1100221612211752960, "id_str": "1100221612211752960", "text": "screen_nam ", "display_text_range": [8, 15
{"created_at": "Tue Feb 26 02:30:48 +0000 2019", "id": 1100221612656414720, "id_str": "1100221612656414720", "text": "oc stat adrean johnson 9 pt jordan pierc 6
{"created_at": "Tue Feb 26 02:30:48 +0000 2019", "id": 1100221614598250496, "id_str": "1100221614598250496", "text": "musclemamamonday mondaymotiv repost screen
{"created_at": "Tue Feb 26 02:30:48 +0000 2019", "id": 1100221615319568384, "id_str": "1100221615319568384", "text": "truth hurt hurt ", "source": "<a href=\"ht
{"created_at": "Tue Feb 26 02:30:49 +0000 2019", "id": 1100221615915163648, "id_str": "1100221615915163648", "text": "ff ", "source": "<a href=\"http://twitter.
{"created_at": "Tue Feb 26 02:30:49 +0000 2019", "id": 1100221617597120512, "id_str": "1100221617597120512", "text": "wish could talk someon alway peopl everyor
{"created_at": "Tue Feb 26 02:30:49 +0000 2019", "id": 1100221617924276224, "id_str": "1100221617924276224", "text": "test_geo_hierarchi ad7dcae0 9863 425a a18f
{"created_at": "Tue Feb 26 02:30:49 +0000 2019", "id": 1100221618209443841, "id_str": "1100221618209443841", "text": "test_geo_hierarchi 12133b9a f652 479a8208
{"created_at": "Tue Feb 26 02:30:49 +0000 2019", "id": 1100221618054258688, "id_str": "1100221618054258688", "text": "screen_nam good dud mud queen dark know mi
{"created_at": "Tue Feb 26 02:30:49 +0000 2019", "id": 1100221618796675072, "id_str": "1100221618796675072", "text": "screen_nam ohhhh ", "display_text_range":
{"created_at": "Tue Feb 26 02:30:50 +0000 2019", "id": 1100221620352802816, "id_str": "1100221620352802816", "text": "screen_nam thank ", "display_text_range":
{"created_at": "Tue Feb 26 02:30:50 +0000 2019", "id": 1100221620575072256, "id_str": "1100221620575072256", "text": "screen_nam screen_nam screen_nam screen_na
{"created_at": "Tue Feb 26 02:30:50 +0000 2019", "id": 1100221621028249600, "id_str": "1100221621028249600", "text": "hate bloat feel get protein shake ", "sour
{"created_at": "Tue Feb 26 02:30:50 +0000 2019", "id": 1100221621556568065, "id_str": "1100221621556568065", "text": "screen_nam screen_nam ", "display_text_ran
{"created_at": "Tue Feb 26 02:30:50 +0000 2019", "id": 1100221623301562368, "id_str": "1100221623301562368", "text": "screen_nam bueno lo van bloquear de todo l
{"created_at": "Tue Feb 26 02:30:50 +0000 2019", "id": 1100221623783706624, "id_str": "1100221623783706624", "text": "plot twist guy much drama girl ", "source"
{"created_at": "Tue Feb 26 02:30:50 +0000 2019", "id": 1100221622470946816, "id_str": "1100221622470946816", "text": "quick last minut word list review wish luc
{"created_at": "Tue Feb 26 02:30:51 +0000 2019", "id": 1100221626317180928, "id_str": "1100221626317180928", "text": "screen_nam ye close long ass plane trip ",
```

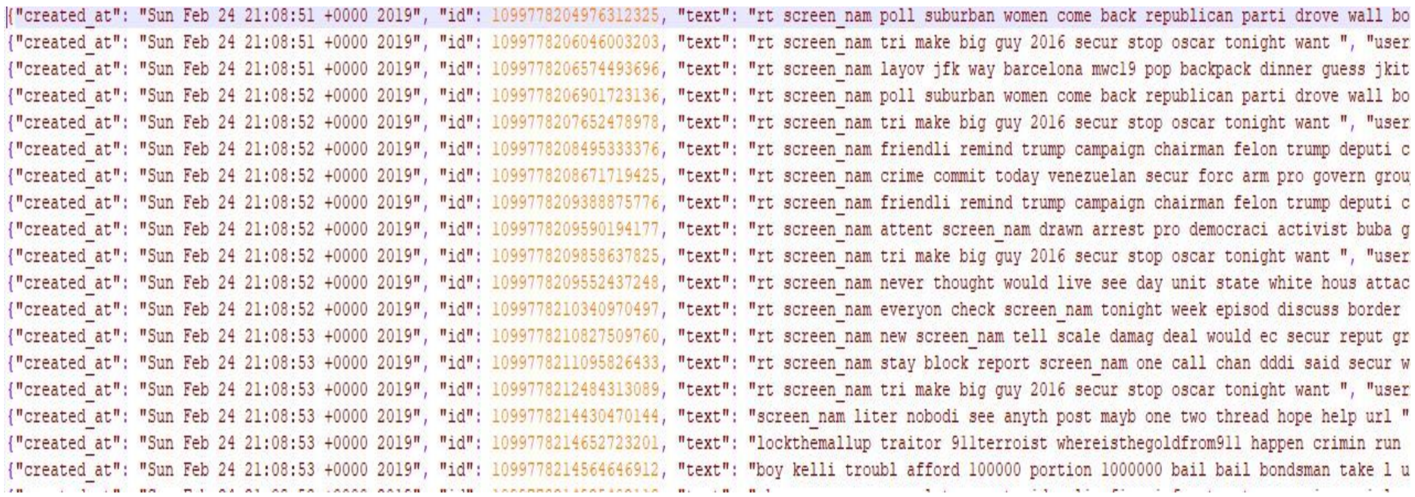
Preprocessing of the data:

1. We preprocessed the text part of the tweets. The functions that we have implemented includes:
    - Tokenizing
    - lamentizing
    - Removing non ascii characters
    - Removing stopwords
    - Replace contractions
    - Strip html
    - Stemming

2. We have collected the data/tweets in JSON format. The raw data JSON format had 19 fields on average

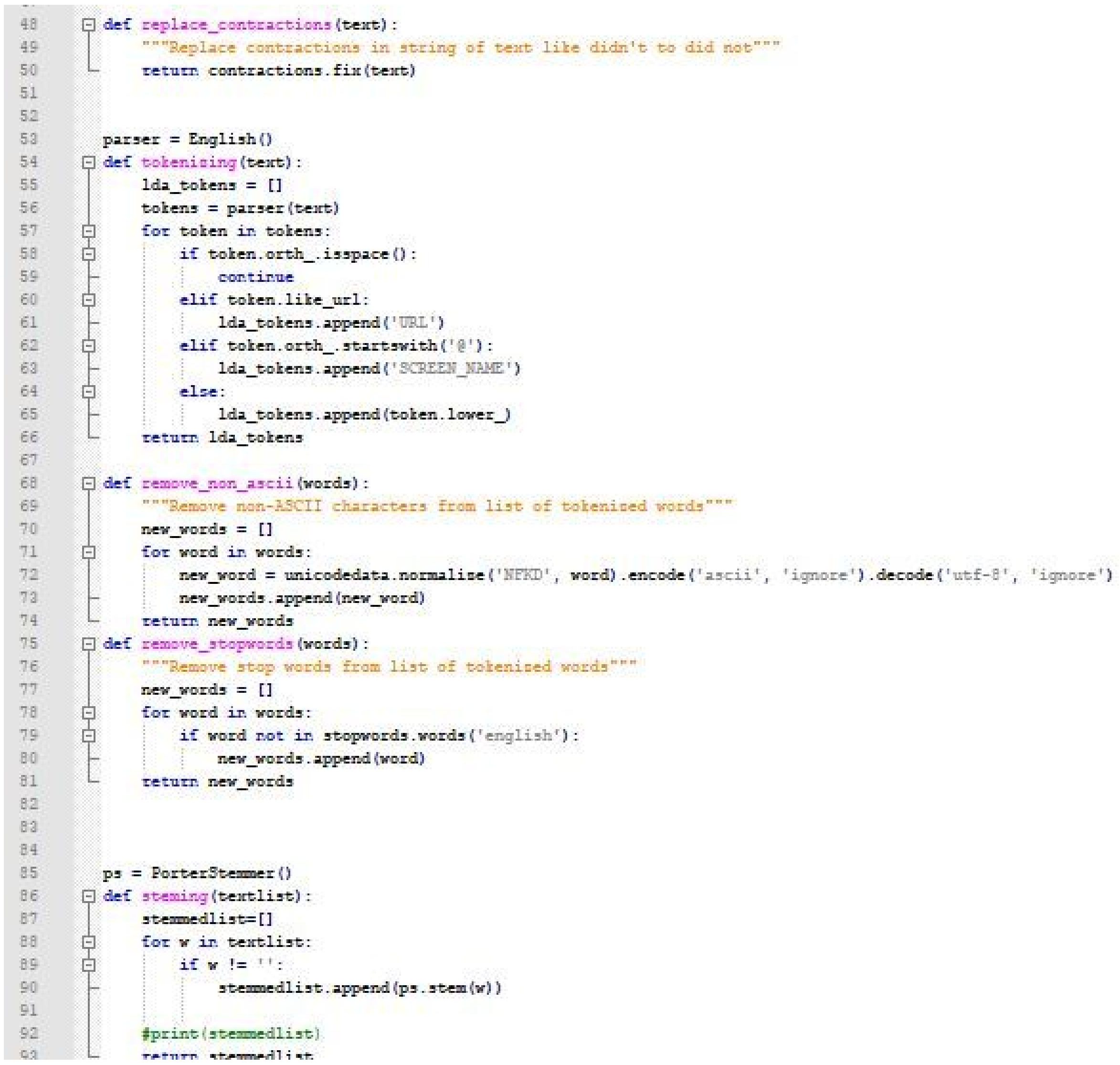
3. We then indexed the data using only 5 fields.

A sample of modified data is given below:



```
{"created_at": "Sun Feb 24 21:08:51 +0000 2019", "id": 1099778204976312325, "text": "rt screen_nam poll suburban women come back republican parti drove wall bo
{"created_at": "Sun Feb 24 21:08:51 +0000 2019", "id": 1099778206046003203, "text": "rt screen_nam tri make big guy 2016 secur stop oscar tonight want ", "user
{"created_at": "Sun Feb 24 21:08:51 +0000 2019", "id": 1099778206574493696, "text": "rt screen_nam layov jfk way barcelona mwc19 pop backpack dinner guess jkit
{"created_at": "Sun Feb 24 21:08:52 +0000 2019", "id": 1099778206901723136, "text": "rt screen_nam poll suburban women come back republican parti drove wall bo
{"created_at": "Sun Feb 24 21:08:52 +0000 2019", "id": 1099778207652478978, "text": "rt screen_nam tri make big guy 2016 secur stop oscar tonight want ", "user
{"created_at": "Sun Feb 24 21:08:52 +0000 2019", "id": 1099778208495333376, "text": "rt screen_nam friendli remind trump campaign chairman felon trump deputi c
{"created_at": "Sun Feb 24 21:08:52 +0000 2019", "id": 1099778208671719425, "text": "rt screen_nam crime commit today venezuelan secur forc arm pro govern grou
{"created_at": "Sun Feb 24 21:08:52 +0000 2019", "id": 1099778209388875776, "text": "rt screen_nam friendli remind trump campaign chairman felon trump deputi c
{"created_at": "Sun Feb 24 21:08:52 +0000 2019", "id": 1099778209590194177, "text": "rt screen_nam attent screen_nam drawn arrest pro democraci activist buba g
{"created_at": "Sun Feb 24 21:08:52 +0000 2019", "id": 1099778209858637825, "text": "rt screen_nam tri make big guy 2016 secur stop oscar tonight want ", "user
{"created_at": "Sun Feb 24 21:08:52 +0000 2019", "id": 1099778209552437248, "text": "rt screen_nam never thought would live see day unit state white hous attac
{"created_at": "Sun Feb 24 21:08:52 +0000 2019", "id": 1099778210340970497, "text": "rt screen_nam everyon check screen_nam tonight week episod discuss border
{"created_at": "Sun Feb 24 21:08:53 +0000 2019", "id": 1099778210827509760, "text": "rt screen_nam new screen_nam tell scale damag deal would ec secur reput gr
{"created_at": "Sun Feb 24 21:08:53 +0000 2019", "id": 1099778211095826433, "text": "rt screen_nam stay block report screen_nam one call chan dddi said secur w
{"created_at": "Sun Feb 24 21:08:53 +0000 2019", "id": 1099778212484313089, "text": "rt screen_nam tri make big guy 2016 secur stop oscar tonight want ", "user
{"created_at": "Sun Feb 24 21:08:53 +0000 2019", "id": 1099778214430470144, "text": "screen_nam liter nobodi see anyth post mayb one two thread hope help url "
{"created_at": "Sun Feb 24 21:08:53 +0000 2019", "id": 1099778214652723201, "text": "lockthemallup traitor 911terroist whereisthegoldfrom911 happen crimin run
{"created_at": "Sun Feb 24 21:08:53 +0000 2019", "id": 1099778214564646912, "text": "boy kelli troubl afford 100000 portion 1000000 bail bail bondsman take 1 u
```

Code snippet of the preprocessing is given below

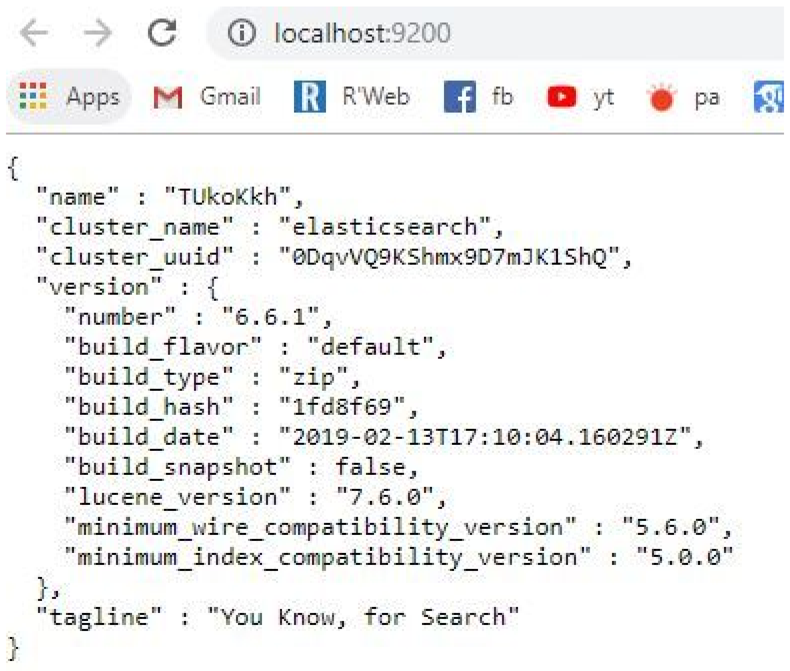


```python
def replace_contractions(text):
    """Replace contractions in string of text like didn't to did not"""
    return contractions.fix(text)


parser = English()
def tokenizing(text):
    lda_tokens = []
    tokens = parser(text)
    for token in tokens:
        if token.orth_.isspace():
            continue
        elif token.like_url:
            lda_tokens.append('URL')
        elif token.orth_.startswith('@'):
            lda_tokens.append('SCREEN_NAME')
        else:
            lda_tokens.append(token.lower_)
    return lda_tokens

def remove_non_ascii(words):
    """Remove non-ASCII characters from list of tokenized words"""
    new_words = []
    for word in words:
        new_word = unicodedata.normalize('NFKD', word).encode('ascii', 'ignore').decode('utf-8', 'ignore')
        new_words.append(new_word)
    return new_words
def remove_stopwords(words):
    """Remove stop words from list of tokenized words"""
    new_words = []
    for word in words:
        if word not in stopwords.words('english'):
            new_words.append(word)
    return new_words



ps = PorterStemmer()
def steming(textlist):
    stemmedlist=[]
    for w in textlist:
        if w != '':
            stemmedlist.append(ps.stem(w))

    #print(stemmedlist)
    return stemmedlist
```

Indexing using Lucene:

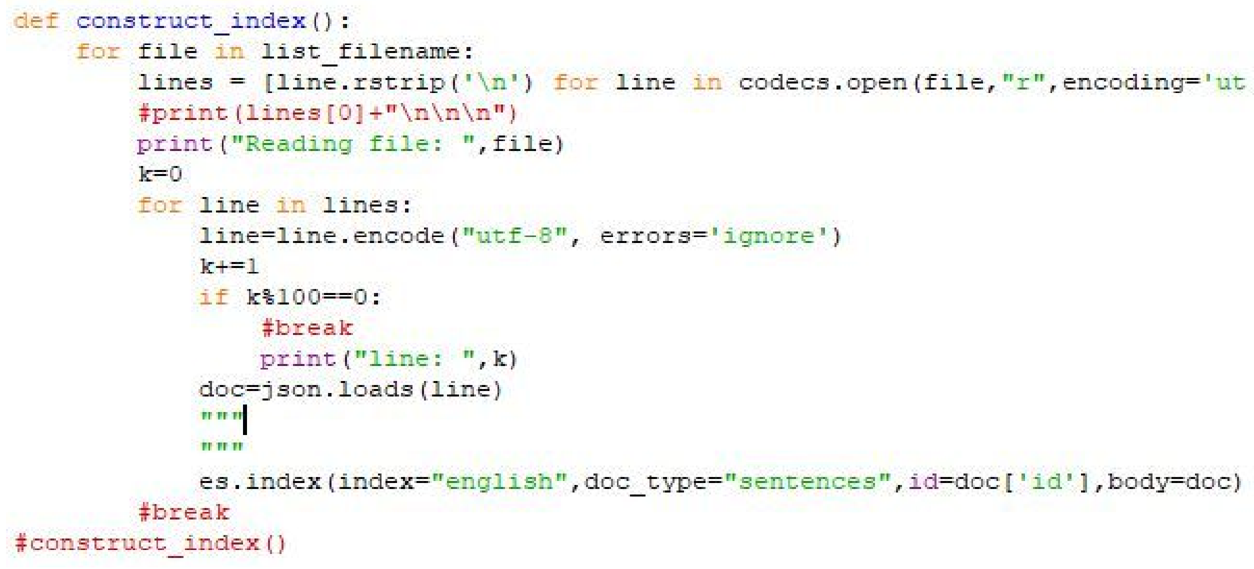
1. We have used Elastic search package from python to index the tweets.
2. Elastic search uses Lucene in background to index.
3. We have indexed the whole tweets based on location and text.
4. We had built a local server for the elastic search module.

A brief description of how elastic search and Lucene works is given in the following section.
Now a figure of local elastic search engine ruuning:



```
{
  "name" : "TUkoKkh",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "0DqvVQ9KShmx9D7mJK1ShQ",
  "version" : {
    "number" : "6.6.1",
    "build_flavor" : "default",
    "build_type" : "zip",
    "build_hash" : "1fd8f69",
    "build_date" : "2019-02-13T17:10:04.160291Z",
    "build_snapshot" : false,
    "lucene_version" : "7.6.0",
    "minimum_wire_compatibility_version" : "5.6.0",
    "minimum_index_compatibility_version" : "5.0.0"
  },
  "tagline" : "You Know, for Search"
}
```

The code snippet of elastic search indexing is given below:

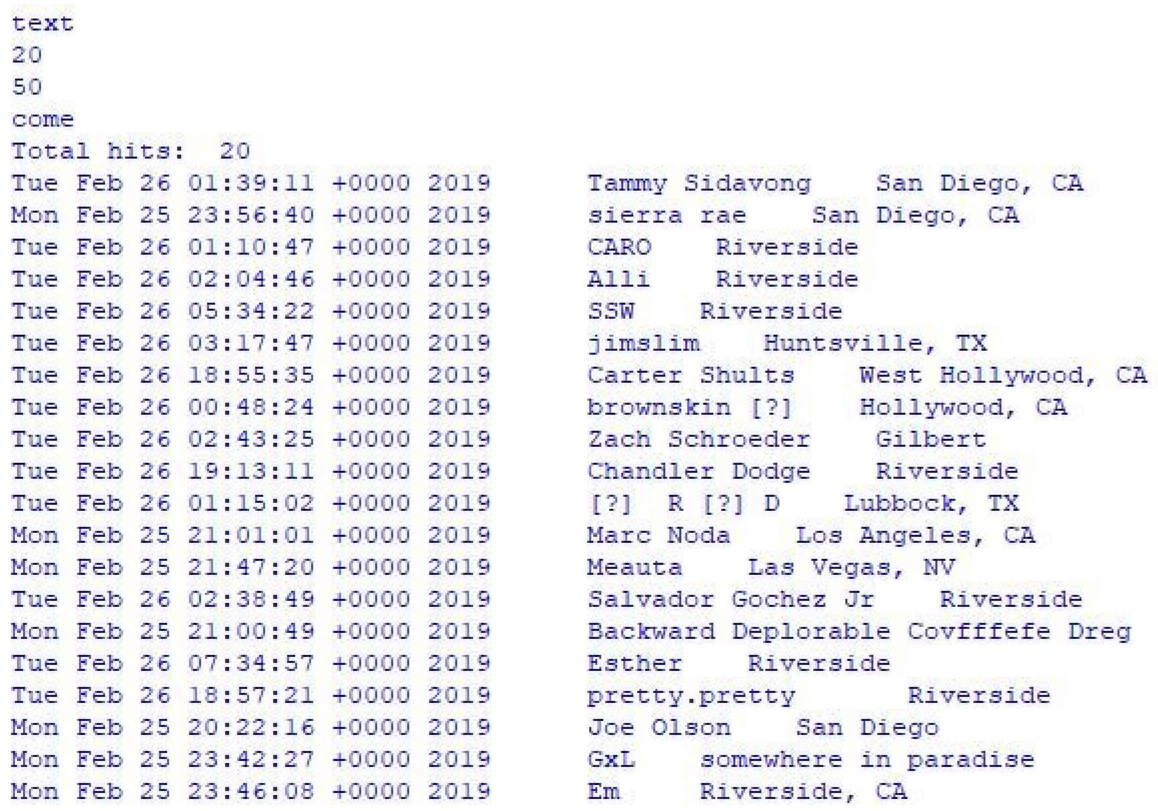


```
def construct_index():
    for file in list_filename:
        lines = [line.rstrip('\n') for line in codecs.open(file,"r",encoding='ut
        #print(lines[0]+"\n\n\n")
        print("Reading file: ",file)
        k=0
        for line in lines:
            line=line.encode("utf-8", errors='ignore')
            k+=1
            if k%100==0:
                #break
                print("line: ",k)
            doc=json.loads(line)
            """
            """
            es.index(index="english",doc_type="sentences",id=doc['id'],body=doc)
        #break
#construct_index()
```

Search using lucene (Elastic search):

1. Elastic search package offers built in search option for the indexed data
2. BM25 is the default ranking algorithm.
3. We can limit the search result in the parameter.

A glimpse of search result is given below:

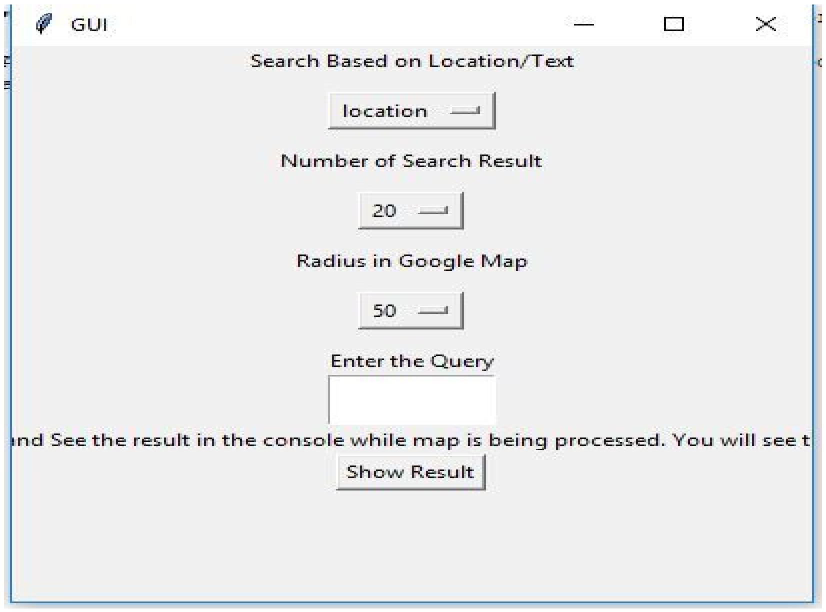


```
text
20
50
come
Total hits:  20
Tue Feb 26 01:39:11 +0000 2019      Tammy Sidavong    San Diego, CA
Mon Feb 25 23:56:40 +0000 2019      sierra rae    San Diego, CA
Tue Feb 26 01:10:47 +0000 2019      CARO    Riverside
Tue Feb 26 02:04:46 +0000 2019      Alli    Riverside
Tue Feb 26 05:34:22 +0000 2019      SSW    Riverside
Tue Feb 26 03:17:47 +0000 2019      jimslim    Huntsville, TX
Tue Feb 26 18:55:35 +0000 2019      Carter Shults    West Hollywood, CA
Tue Feb 26 00:48:24 +0000 2019      brownskin [?]    Hollywood, CA
Tue Feb 26 02:43:25 +0000 2019      Zach Schroeder    Gilbert
Tue Feb 26 19:13:11 +0000 2019      Chandler Dodge    Riverside
Tue Feb 26 01:15:02 +0000 2019      [?]  R [?] D    Lubbock, TX
Mon Feb 25 21:01:01 +0000 2019      Marc Noda    Los Angeles, CA
Mon Feb 25 21:47:20 +0000 2019      Meauta    Las Vegas, NV
Tue Feb 26 02:38:49 +0000 2019      Salvador Gochez Jr    Riverside
Mon Feb 25 21:00:49 +0000 2019      Backward Deplorable Covfffefe Dreg
Tue Feb 26 07:34:57 +0000 2019      Esther    Riverside
Tue Feb 26 18:57:21 +0000 2019      pretty.pretty       Riverside
Mon Feb 25 20:22:16 +0000 2019      Joe Olson    San Diego
Mon Feb 25 23:42:27 +0000 2019      GxL    somewhere in paradise
Mon Feb 25 23:46:08 +0000 2019      Em     Riverside, CA
```

GUI implementation:

1. We have built a GUI in python using Tkinter package.
2. The GUI had 3 input field.
    - Search based on Location/text
    - Number of search result
    - Searching radius
3. These input fields values were used to find the search result and showing the result in google map.

The implemented GUI looks like this:


GUI

Search Based on Location/Text

location

Number of Search Result

20

Radius in Google Map

50

Enter the Query

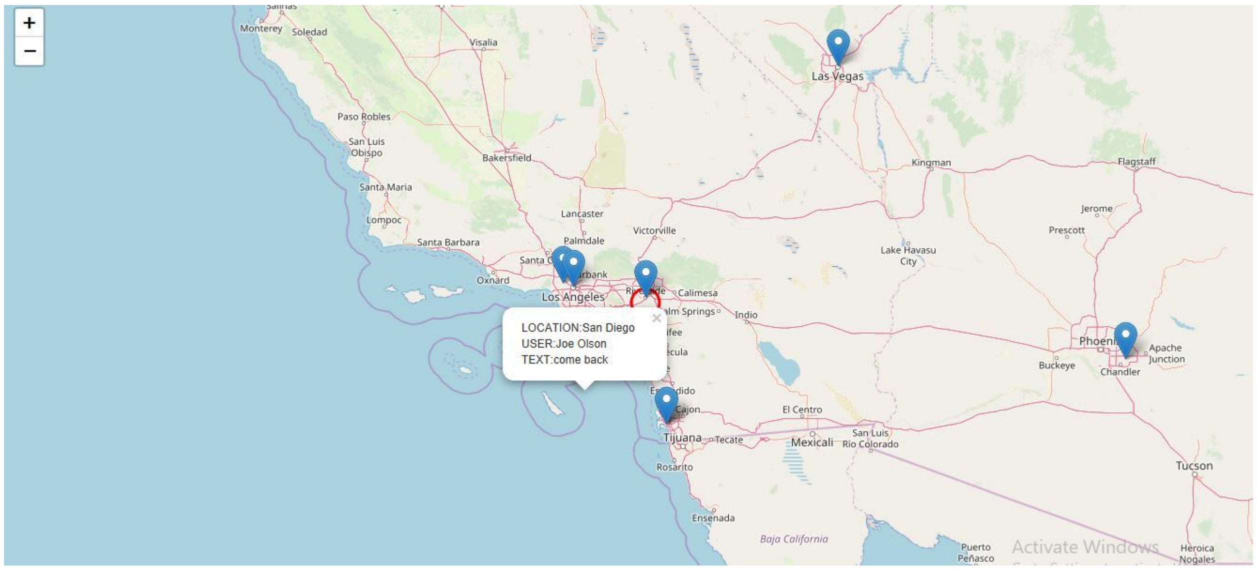
nd See the result in the console while map is being processed. You will see t

Show Result

Showing the result in Google map:

1. We have shown the search result in google map in the area radius defined by the user centered at users location
2. We have used geocoder package (to find the current location of the user), geopy (to convert the text address to coordinate and calculate the distance in miles) and follium package (to draw the google map html)
3. The result map showed the location, text and the username of the tweets in the specified regions defined by radius.

The final result looks like the following figure:


LOCATION:San Diego
USER:Joe Olson
TEXT:come back

How to run the project:

1. To index the data using Lucene, we have to enable/uncomment the function named 'construct_index'. Before that we have to run the local server of elastic search which was downloaded in local machine
2. Then we have to run the code 'elastic search.py' using F5 key . Note that you have to have the data in another folder in the current directory anmed 'moddata'. Also we have to install all the required packages which is mentioned in README.txt.

Discussion of Tweeter Search:

This project was a collaborative project by Risul Islam, David Swanson and Jhaymer Sabino. We have learned a lot of things like how stream collection, indexing, searching, google map works. A nice GUI and showing the result in Google map enriched our project. Still there are many things that can be added with our project. We can implement the various ranking algorithms within Lucene. Overall, this was a very good project where we learnt team spirit and helps us to understand the basics of the search engine.

As a part of bonus mark, we have also implement the Web searching of University related web pages. That part and also a brief description of the packages used are described below:

Web Search by David Swanson

https://github.com/CS172-UCR/finalproject-hookedonpythonics/tree/master/web_search

My part of the project has a lot of code because, as a student, I prefer to write my own helpers. Aside from Elasticsearch, Requests, NLTK Stemmer and html.parser, my project does not depend on any high-level modules.

Milestones Completed

- URL scraper, web content scraper, index builder, and search engine with document snippets.
- Also implemented the simhash algorithm from the slides, an N-gram generator, various iterators and utility functions.

Items not Completed

- Browser-based search input and display.
    - Input and output are via command line

Features added after demo

- Snippets

What I learned

- Python is not my main language, so I learned a lot about Python.
- Dealt with incompatibilities between Python 2 and 3, and modules that work in one and not the other.
- Learned to make a virtual environment to control versions using pipenv and pip
- Learned never to compile pylucene from source. Spent hours on that, then installed Elasticsearch in minutes.
- Learned more about hashing

Output samples (In directory: output_samples)

https://github.com/CS172-UCR/finalproject-hookedonpythonics/tree/master/web_search/output_samples

- Search.txt
  - Output from sample searches, returning document number, url and a short snippet from ten results
- buildIndex.txt
  - Output from index builder
- Url_scrape_log_200urls.txt
  - Output from url_scraper showing duplicates removed (exact match) and a list of rejected urls (criteria defined in urlscraper.py)
- Urllist.txt
  - A list of 3500 urls scraped from 3 seeds
- web_scrape_log_50MB.txt
  - Output from a web content scrape run
- Scrape_badlist.txt
  - A list of urls rejected during web scrape (criteria defined in docscraper.py)
- Ngrams.txt
  - Sample output from N-gram generator with original text, N-grams and stemmed N-grams
- Simhash_test.txt
  - Fed simhash 5 short documents:
  - 1 and 2 unique
  - 4 and 5 identical
  - 3 differing from 4 and 5 by a few words

    - The algorithm allowed 3 and rejected 4 and 5 by similarity to 3

Program includes these files:

- dociters.py
- urlscraper.py
- docscraper.py
- elastic.py
- myutils.py
- simhash.py
- app.py

dociters.py

Many tasks involve tokenizing a line of text and parsing word-by-word. I developed a family of classes that abstract everything down to 'nextWord' and 'done'. The classes parse strings, lists, files and web documents. You can use a strategy pattern to provide a different data source without changing the client implementation. Additional classes include the N-gram iterator and a directory iterator. Most of the iterators are related either by inheritance or composition.

- class TextIter(object):
    - Input is string or list
- class SmallFileIter(TextIter), class FileIter(TextIter):
    - Input is a file name.
    - One class dumps the file into memory; the other gets one line at a time from file.
- class WebDocIter(TextIter):
    - Input is a url.
- class XMLIter(object):
    - Left over from homework 2. Wraps a file iterator.
- class NGram(object):
    - Input is a wordIterator of any type, and a value for N. (See directory: output_samples)
- class DirIter(object):
    - Input is a directory name.
    - Generates sequential file names, opens json files and returns object.

Urlscraper.py

Uses simple string search to locate urls.

- class urlFinder(object):
    - Parses a single web document.
    - Checks header and text in url; rejects unwanted types or bad urls.

        - Builds list and removes duplicates. (Clears duplicates after every run, because that is cheaper than hitting a web page twice.)
- class urlScraper(urlFinder):
    - Calls parent functions to parse documents until number of urls collected reaches assigned amount.
    - Removes duplicates again to catch cross-feed between related websites
    - See directory: output_samples

Docscraper.py

I did not use Beautiful Soup. Instead I found a simple module (html.parser) that fires event handlers when it encounters an html tag. I chose relevant or non-relevant content based on the type of tag enclosing it.

- class HTMLTagHandler(HTMLParser):
    - Inherits from html.parser; overrides the event handlers for various tags and data.
    - Uses dict of function pointers to map tag types to actions.
    - Rejects documents that don't say 'lang' and 'en'. Rejects documents that don't say 'charset' and 'utf-8'. (This weeds out icons and stray css files)
    - Skips style, script and option tags. Checks length on anchor tags; skips one- and two-word links.
        - You can wrap a whole paragraph in a link; it doesn't make sense to skip them completely. But short text in a link is probably just a link.
- class DocScraper(object):
    - Gets url list from file and visits every page until stored data reaches assigned amount.
    - Saves three types of file:
        - HTML file with complete document
        - JSON file with stemmed content (for search)
            - Fields: docno, url, simhash, last-modified, title, content
        - TXT file with un-stemmed content (for snippets)
    - Uses simhash to reject near-duplicates
    - See directory: output_samples

elastic.py

- def buildIndex():
    - Calls a directory iterator (see word iterators, above) to get json objects for Elasticsearch index.
    - See directory: output_samples
- class WebIndex(object):
    - Runs Elasticsearch, gets result and displays.
    - Uses simple string search to locate snippets for search results.

    - The problem with snippets:
        - A stemmer is a sort of hash, so you can't unstem a word. But you can't display stemmed snippets.
        - The solution is to keep an un-stemmed copy of the same content stored in the json files.
        - Generate snippets from the un-stemmed file: stem each word individually and check for matches with the stemmed query. On a match, get the un-stemmed line where you found it, along with the position of the word, then edit the line down to the five words preceding and following the found word. Then choose the longest line to display as the snippet. Not the best solution.

myutils.py

Not going to describe all the functions in this file, but some are interesting.

- def fixPath(filename, subdir=defaultSubDir):
    - Generates absolute path to app so file open never fails (never?)
- class globalStopWordTest(object):
    - Many classes check the same stop words, so this is a global singleton.
- def nestedFind(l1, target): # for very small nested collections
    - Recursively searches for a string wrapped in maps or lists.
- def hammingDistance_int(a, b, nBytes=4):
    - For simhash
- class UQGen(object):
    - Returns an incrementing value as integer or formatted string '00001' for unique file names. I
    - mplements Python 'magic' functions, so you can call it like this int(uq) or str(uq) and get a different value every time.
    - Can save state to file, for resuming a previous task after system shutdown.
- class Stemmer_killPunct, class dummyStemmer:
    - Wraps the porter stemmer with a punctuation remover.
    - Substitute dummyStemmer to defeat.
- class FileWriter(dummyFileWriter), class dummyFileWriter(object):
    - Wraps standard file writer with a logger, to keep track of size written.
    - Substitute dummyFileWriter to defeat.

simhash.py

- A simple implementation, just hashes the text
- Follows the slide example and works quite well.
- Uses a CRC32 hash.
- For test, see directory: output_samples

app.py

- Main function with a simple menu

Discussion of Web Search:

In this part we mainly focused on how we can crawl the web pages. Various web crawling techniques were implemented from scratch. We tried to remove duplicate, implement the frontier queue and store the data efficiently. Searching in the crawled text is trivial then just like Tweeter Search. This port gave us a working knowledge of how crawling works.

Reference:

1. https://www.elastic.co/blog/elasticsearch-6-0-0-released
2. https://www.mapdevelopers.com/geocode_tool.php
3. https://python-visualization.github.io/folium/
4. https://www.nltk.org/book/ch03.html
5. https://stackoverflow.com/questions/27712472/choosing-between-simhash-and-minhash-for-a-production-system