Article Similarities Report

Introduction:

This project is about querying New York Times article body using API. This API is an "Article Search API" that searches for New York Times article by keyword. The dataset is being stored into persistent storage on my machine, where I used *SQLite*. The program then allow user to input the keyword that will search for the body of the article contains the keyword. After the article body is being matched with the keyword, then the program computes the text similarities between the relevant articles and display the similarities. This is being computed through *PySpark*. Since the result articles are too many to compute similarity, I chose the first 5 articles to compute. The similarities are displayed as the frequency of the words appears on both of the articles that has been compared.

Data Collection into SQLite:

API is available on website "https://developer.nytimes.com/get-started". Also, the details of this API are available at "https://developer.nytimes.com/docs/articlesearch-product/1/overview". Data is collected from New York Time database and are collected based upon my request, such as section and topics. I filtered my search by having restricting my search to articles from Sports, which will be the section. Then, I used five different topics, which are "olympic", "medal", "athletes", "basketball", and "champion", that will be the search queries in my API that will retrieve articles related to these topics. For every query word, I chose to collect 100 articles so that I can have a total of 500 articles from topic sports with various keywords. Be aware that if trying to collect 500 articles at a

time will possibly generate a “dead kernel” in Jupyter Notebook because of the huge running time. For every time I collect the data from New York Times, the data would be in format of meta-data, which does not contain the body of the article.

Example of meta-data retrieved directly from The New York Times API:

	_id	abstract	blog	byline	document_type	headline	keywords	lead_paragraph	multimedia	news_desk	print_page
0	5cb711c140d36ff0709d3940	The Olympic media partner will dive into the s...	{}	{'original': 'By Kevin Draper', 'person': [{'f...	article	{'main': 'NBC Will Lead Ad Sales for the Los A...	[{'name': 'subject', 'value': 'Olympic Games (...	The Summer Olympics won't arrive in Los Angele...	[{'rank': 0, 'subtype': 'xlarge', 'caption': N...	Sports	12

b_date	section_name	snippet	source	subsection_name	type_of_material	uri	web_url	word_count
19-04-4+0000	Sports	The Olympic media partner will dive into the s...	The New York Times	NaN	News	nyt://article/ad2e2c60-e224-5e8e-bd9f-bb596880...	https://www.nytimes.com/2019/04/17/sports/nbc-...	439

Therefore, from there I am only storing the “_id” and “web_url” of the article. Article body is not directly available from querying from the API, instead it requires one more step to compute. Let’s see the first web URL’s article body.

Example of text in web url from meta-data:

```
response1 = requests.get(docs_df['web_url'][0])
```

```
response1.text
```

```
'<!DOCTYPE html>\n<html lang="en" itemId="https://www.nytimes.com/2019/04/17/sports/nbc-olympics.html" itemType="htt
p://schema.org/NewsArticle" itemScope="true" class="story" xmlns:og="http://opengraphprotocol.org/schema/">\n  <head>
\n    <title data-rh="true">NBC Will Lead Ad Sales for the Los Angeles Olympics - The New York Times</title>\n    <me
ta data-rh="true" itemprop="inLanguage" content="en-US"/><meta data-rh="true" property="article:published" itemprop
="datePublished" content="2019-04-17T11:45:04.000Z"/><meta data-rh="true" property="article:modified" itemprop="dateM
odified" content="2019-04-18T04:42:53.352Z"/><meta data-rh="true" http-equiv="Content-Language" content="en"/><meta d
ata-rh="true" name="robots" content="noarchive"/><meta data-rh="true" name="articleid" itemprop="identifier" content
="100000006464472"/><meta data-rh="true" name="nyt_uri" itemprop="identifier" content="nyt://article/ad2e2c60-e224-5e
8e-bd9f-bb5968802e1e"/><meta data-rh="true" name="pubp_event_id" itemprop="identifier" content="pubp://event/c866fcb6
b86b419896fa311cc7e5345d"/><meta data-rh="true" name="description" itemprop="description" content="The Olympic media
partner will dive into the sponsorship sales market usually handled by the local organizing committee to create one-s
top shopping for advertisers."/><meta data-rh="true" name="image" itemprop="image" content="https://static01.nyt.com/
images/2019/04/17/sports/17nbc-Olympics/merlin_137394414_02fde79d-db80-4f48-a87e-9ea82bba4c6f-facebookJumbo.jpg"/><me
ta data-rh="true" name="byl" content="By Kevin Draper"/><meta data-rh="true" name="thumbnail" itemprop="thumbnailUrl"
content="https://static01.nyt.com/images/2019/04/17/sports/17nbc-Olympics/merlin_137394414_02fde79d-db80-4f48-a87e-9e
a82bba4c6f-thumbStandard.jpg"/><meta data-rh="true" name="news_keywords" content="Olympic Games (2028),Advertising Ma
rketing,International Olympic Committee,NBCUniversal,US Olympic Committee,Linda Yaccarino"/><meta data-rh="true" name
="pdate" content="20190417"/><meta data-rh="true" property="og:url" content="https://www.nytimes.com/2019/04/17/sport
s/nbc-olympics.html"/><meta data-rh="true" property="og:type" content="article"/><meta data-rh="true" property="og:ti
```

Therefore, it requires a function to retrieve the body of the article, which is done by this function.

Extracting body of the article function:

```
# function of getting the text body from the article
def extract_art_body(response):
    article=response.text[response.text.find('<section name="articleBody"'):response.text.find('<div class="bottom-of-a
    while True:
        i1 = article.find('<')
        i2 = article.find('>')

        if i1<0 or i2<0:
            break
        if i1==0:
            article = article[i2+1:]
        else:
            article = article[:i1]+' '+article[i2+1:]
    return article
```

Next, I am storing the article body along with every corresponding article ID and URL through adding each article to the SQLite one by one. Since each page of querying of the article using API gives maximum of 10 articles at a time, the inner loop is required to collect body of the articles.

Example code for topic “olympic”:

```
## get 100 articles of New York Times from the section "Sports" and topic "olympic"

for page in range(10):
    url = "https://api.nytimes.com/svc/search/v2/articlesearch.json?q=olympic&fq=news_desk:(\"Sports\")&page=" + str(pa
    response = requests.get(url)
    docs = json.loads(response.content.decode('utf-8'))['response']['docs']
    docs_df = pd.DataFrame(docs)

    # extract the articles in each page (each page contains 10 articles)
    body = []
    for i in range(10):
        response1 = requests.get(docs_df['web_url'][i])
        body.append(extract_art_body(response1))
    docs_df.insert(0,'art_body', body,allow_duplicates = False)

    docs_df1 = docs_df[['_id','web_url','art_body']]
    docs_df1.to_sql("dataset",connection,if_exists = "append",index=False)
    time.sleep(6)
```

As shown in the code, I first store the meta-data into the pandas data frame, then insert another column named “art_body” into the meta-data that has been already stored in to pandas data frame, which corresponds to its body of the article. Then, store the three columns (“_id”, “web_url”, “art_body”) into the SQLite database.

Example data in SQLite database:

```
table1.head()
```

	_id	web_url	art_body
0	5cb7121a49f0eacbf1f84e38	https://www.nytimes.com/2019/04/17/sports/nbc-...	The Summer Olympics won't arrive in Los Angele...
1	5be9b0383a125f5075bf9d7a	https://www.nytimes.com/2018/11/12/sports/calg...	CALGARY, Alberta — Hobbled by a reputation for...
2	5c6d7a433a125f5075c0a9d6	https://www.nytimes.com/2019/02/20/sports/kikk...	PENTICTON, British Columbia — Inside the livin...
3	5c3893663a125f5075c03eaa	https://www.nytimes.com/2019/01/11/world/europ...	LONDON — The president of Japan's Olympic Comm...
4	5c677cdf3a125f5075c0a06a	https://www.nytimes.com/2019/02/15/sports/biat...	Some of the world's top biathletes descended t...

Design of Query by importing PySpark:

When I design the query component, the first thing I did is import the PySpark, because PySpark has great functions that can easily split the words of article body and count the frequency of them. Also, one of the great advantages of PySpark is that the running time. There are some benefits using PySpark in this project since PySpark creates Resilient Distributed Dataset, which is the distributed memory abstraction for in-memory computations on large clusters and it allows persistence of intermediate result in memory, so it makes the running time very short. By knowing these facts, I decided to use PySpark to compute the text similarities of the articles that are being queried. However, importing the PySpark was one of the largest struggles I faced throughout the whole process. Although I have successfully installed and imported the PySpark, I keep getting errors when running the SparkSession. After long time of struggling and Googling, I finally figure out the reasons of the errors. It was because of the version of JAVA Program in my local computer. I had java version "12.0.1" but it should be java version "1.8.0_211". In order to resolve this problem, first download the Java SE 8u211 from Oracle, then check the version by “echo $JAVA_HOME” in local terminal. Then type “export

JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk1.8.0_211.jdk/Contents/Home" into the terminal. Retype "echo $JAVA_HOME" will finally get the correct version of Java. Then by the code below, PySpark will be available to use finally. If someone else is unable to load PySpark, I strongly encourage to follow this procedure to solve the problem.

```
# Install PySpark (only need to do once, uncomment below three lines if installing for the first time)
# !curl -O https://d3kbcqa49mib13.cloudfront.net/spark-2.2.0-bin-hadoop2.7.tgz
# !tar -xvf spark-2.2.0-bin-hadoop2.7.tgz
# !pip install findspark
```

```
# import PySpark
import os
import findspark
os.environ["PYSPARK_PYTHON"] = "python3"
findspark.init("spark-2.2.0-bin-hadoop2.7",)
from pyspark.sql import SparkSession, Column, Row, functions as F
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/eunbinko/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
# set JAVA HOME directory
os.environ['JAVA_HOME']='/Library/Java/JavaVirtualMachines/jdk1.8.0_211.jdk/Contents/Home'
```

After loading the PySpark, let's use RDD to find out the text similarities. My definition of similarity is counting the common words that are appearing in the two set of articles (any set is fine, and I manually selected the first 5 articles).

Design of text similarities:

I have generated RDD of all of the 500 articles body that is called "body_RDD". Then use "input" to let users input the desiring keyword. This could be either a word or a phrase such as "olympic" or "gold medal". I have transformed the article body to be lower case using "map" function in RDD in order to match the keywords with article body to be not case sensitive. Then use "filter" in RDD to match the article body with the keywords. Now we have "body_search_RDD" that contains all the articles that matches with keywords. Among those articles, select the first five articles and create 5 unique RDDs that contains each of

first 5 articles. Then use “flatMap” to split the words in the articles. Since the words may contain some non-alphabets such as apostrophes in “won’t” or some numbers, use regex to remove all the non-alphabets. Then anti-join with the stop words to remove the unnecessary words to compute text similarities better.

Example of non-stop words first articles that is being matched with keywords:

```
# non-stop words in article 1
non_stop_word_1
```

```
['summer',
 'olympics',
 'wont',
 'arrive',
 'los',
 'angeles',
 'another',
 'nine',
 'years',
 'local',
 'organizing',
 'committee',
 'begin',
 'selling',
```

Then find the common words between the articles by comparing the non-stop words in article 1 and 2, article 3 and 4, article 1 and 5, article 1 and 4, and finally article 3 and 5. By doing so, the five different set of articles can compute the text similarities. Use “map” function and “reduceByKey” function in RDD to get the frequencies of words that are in common. Then input the store the result into the pandas data frame for a better visualization.

Result of queries:

After all the computation, the result of the data frame would contain three columns: the two sets of articles’ bodies that are being compared, which are located at the first and the

second columns, and the text similarities between the them, which are located at the third column.

Example of result in text similarities:

```
# show result
df_similarity
```

	Article a	Article b	Similarity between a and b
0	[the summer olympics won't arrive in los angel...	[calgary, alberta — hobbled by a reputation fo...	[(8, olympic), (7, committee), (6, olympics), ...
1	[penticton, british columbia — inside the livi...	[london — the president of japan's olympic com...	[(18, said), (7, olympic), (6, last), (5, year...
2	[the summer olympics won't arrive in los angel...	[some of the world's top biathletes descended ...	[(8, olympic), (6, olympics), (5, usoc), (4, s...
3	[calgary, alberta — hobbled by a reputation fo...	[london — the president of japan's olympic com...	[(19, olympic), (16, games), (9, said), (9, of...
4	[penticton, british columbia — inside the livi...	[some of the world's top biathletes descended ...	[(18, said), (7, olympic), (7, medal), (6, las...

Since this pandas data frame doesn't show fully, we can select the column and row as "dataframe.loc[row][col]" to see the full data.

Example of the text similarities for the first two articles:

```
# look at first row of dataframe fully (similarities in first row)
df_similarity.loc[0][2]
```

```
[(8, 'olympic'),
 (7, 'committee'),
 (6, 'olympics'),
 (6, 'games'),
 (5, 'advertising'),
 (4, 'said'),
 (4, 'local'),
 (3, 'years'),
 (3, 'went'),
 (3, 'marketing')]
```

The above similarities can be seen as the two articles are about olympic games since the frequency is high in word "olympic". There are some possible future work can be done. For example, we can create another database for the query words, then store the similarities with the query words. By doring so, there would be no repeating jobs since once the words have been searched, the result will be stored, so that users can just look onto the database to see the already serached words with its results.