Search Engine for CORD-19 dataset

Aparna Dutt
Prajwal Chandra
Pramod Sai Kudapa
Anil Rayala

Introduction:

COVID-19 is a topic that needs no introduction. The fatal respiratory virus originated in Wuhan, China last December and has since affected approximately 5M people across the globe, claiming nearly 300k lives as of today. Since its origin, scientists and medical researchers all over the world are scrambling to gain deeper insights about the disease. To aid researchers in this process, the Center for Security and Emerging Technologies (CSET) has partnered with a coalition of leading research groups to consolidate a vast number of scholarly articles into one dataset called CORD-19.

Medical researchers and health professionals can utilize this huge pool of scientific literature on CoronaVirus to find answers to high priority questions about the pandemic. However, due to the massive size of the dataset, it is challenging to navigate through it to access information about specific topics on an individual level. A tool to navigate the dataset can prove to be beneficial for the research community at large.

Goal and Big Picture:

Our goal for the project was to build a search engine for the CORD-19 dataset that, given a query, would provide relevant research articles. CORD-19 is a dataset that is constantly growing. The size of the dataset during the proposal of the project was 8GB. To handle information at such scale, we implemented Big Data frameworks.

Data:

The dataset used in our project is COVID-19 Open Research Dataset (CORD-19). It was prepared by CSET in collaboration with Allen Institute, Chan Zuckerberg Initiative, etc. It comprises around 83,000 articles in JSON format about COVID-19 and other coronaviruses. Of these articles, 70,000 are full-text articles on the novel coronavirus (SARS-CoV-2) and its associated illness COVID-19.

Currently, the size of the dataset is around 9GB. It comes with a metadata.csv which provides auxiliary details about the articles in the dataset. The description of key features in metadata.csv are shown in table 1. The articles in dataset are available in json format. The schema of json files is shown in Figure 1.

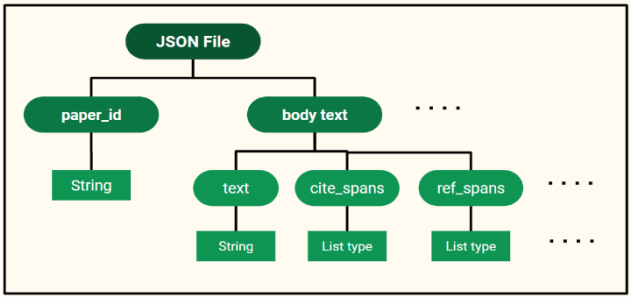

Figure 1. Truncated Schema of JSON files

Attribute	Description
Sha/paper_id	Unique id to each article
source_x	Source of the article (journal, publication)
title	Title of an article
DOI	Digital Object Identifier of an article
abstract	Abstract of the paper
pdf_json_files	Path/file_name of the article in the dataset

Table 1. Some key attributes in metadata file

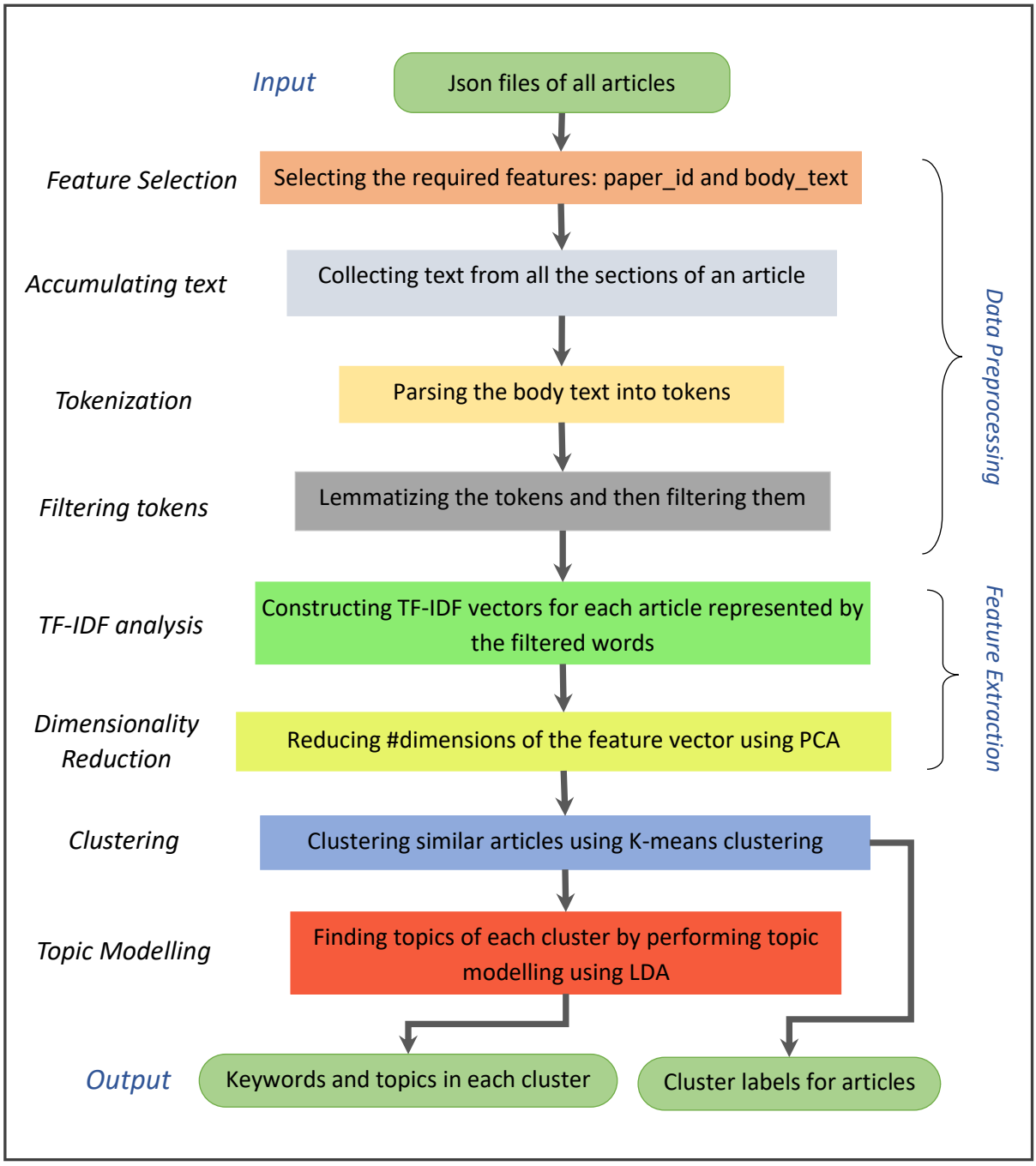

Figure 2. Flow Chart for modules in our method

Methods:

Diagrammatic representation of the method can be seen in Figure 2. Details of individual module are as follows:

Module 1: Data Preprocessing

Input to this first module will be json files. Only the required features (paper_id, body_text) in the json file are extracted. According to given schema of the json format, there are multiple text attributes corresponding to each section of an article. Hence, we have concatenated string values of all the text objects of body_text to form one full text string. Now, this full text of an article is parsed and converted into tokens. For this, we have used parser available in en_core_sci_sm package in python. Next, individual tokens are lemmatised followed by filtering to remove stop-words and punctuations. Standard stop-words in English language are obtained from spacy package and punctuations are obtained from string package. The list/vector of words obtained after performing the above steps represents each article text. Vectors of all the json file will be given as input to the module 2. We have used Spark framework for this module.

Module 2: Feature Extraction

From the vectors obtained from previous module, we performed TF-IDF analysis. For each vector, we first computed term frequency (TF) values for each word in the vector. To compute TF values, we employed the "fit and transform" function in CountVectoriser module from pyspark.ml.features library. Each word vector was provided as input to the aforementioned function resulting in a sparse vector of TF values for words in the input vector as output. Next, we computed Inverse Document Frequency (IDF) values for all the unique words across the entire corpus. Similarly the "fit and transform" function from IDF module in pyspark.ml.features library was used for this step. These IDF values were used to normalise the TF values for each word vector.

The number of dimensions in the resultant vector from TF-IDF computation proved to be too high. To reduce the dimension count, we performed Dimensionality Reduction using Principal Component Analysis. We wanted our function to select the output number of dimensions such that it would preserve 95% variance from the original data. The criteria was selected to avoid noise that is typically captured in dimensions with lower variance. Noise or outliers will be removed as we have considered 95% variance. The new word vectors will be given as input to Module 3.

Module 3: K-means Clustering

Having processed word vectors representing each article, we then grouped them together so that multiple articles written on a similar topic can be returned for a given query. For that, we performed K-means Clustering on the word vectors. K-means library function in tensorflow was used. For a sample input, we have considered the number of clusters to be 5 and 10 iterations for updating the centroids and clusters.

Module 4: Topic Modelling

Articles are clustered together based on similarity in the previous module. However, a search query generally contains words or phrases. How do we decide which documents should be returned to the user based on keywords in the query? The answer lies with topic modelling. Topic modelling is a method used for unsupervised classification of documents. It identifies topics in a set of documents. Latent Dirichlet Allocation (LDA) is a well-known topic modelling technique. We have implemented LDA to compute which topics are significant in each cluster. For a cluster, top keywords in all the topics can be selected and those keywords can be used to label that cluster. Now, based on keywords in the

user query, the corresponding cluster can be selected, and selected documents can be returned as output.

Results:

Feature Extraction:

TF-IDF scores are generated for each of the unique words across all documents and the corresponding word vectors, generated after data pre-processing, are converted into TF-IDF vectors. The TF-IDF vectors generated in our case have a high dimensionality of 24000. Sample output of TF-IDF vector is shown:

```
['human', 'gut', 'colonize', 'complex', 'microbial', 'community', 'population']
0.8082
4.2547
0.9827
4.6846
1.324
1.7162
SparseVector(24887, {0: 0.8082, 1: 4.2547, 2: 0.9827, 3: 4.6846, 4: 1.324, 5: 1.7162, 6: 0.1837,
```

After performing PCA, the number of dimensions reduced from around 24000 to 66. Sample output is shown below:

```
array([[ -5.56070592,  -6.53417904,  -5.50029415, ...,  -0.96156143,
          0.54879508,   1.92024087],
       [-10.97722682, -10.4667689 ,  -5.70243529, ...,   1.27117429,
          0.94598386, -12.90131359],
       [-12.78396943, -11.07820543, -10.35576097, ...,  -2.13178328,
         -8.90397443,  -4.43808902],
       ...,
       [ -3.0124493 ,  -8.52985179,   2.03882229, ...,  -1.73272053,
          1.03586537,   3.30527233],
       [-12.6336302 ,  -8.87571341,  -6.5443216 , ..., -31.84546358,
         13.4823134 ,   4.71082593],
       [-12.97369419, -14.79586829,  -0.30296833, ...,   2.33837477,
         -1.19773225,   2.79290302]])
```

Clustering:

For a small sample of documents, the optimal value for K came out to be 5 (elbow method). Upon running the K-means algorithm on tensorflow, we cluster similar documents together. Cluster 0 and some of its constituent documents can be seen in the below figure.

```
Documents belonging to cluster 0 are
PMC7108572
PMC7108471
PMC7149786
PMC7090783
PMC7157466
PMC7127629
PMC6669655
PMC3974838
```

Topic Modelling:

We have chosen 10 topics per cluster and top 3 words for each topic. We ended up with 30 keywords per cluster. A sample of 30 keywords for the first cluster can be seen below.

```
nsp
protein
pedv
air
time
model
bcov
titers
vaccine
virus
viral
study
hiv-
platelets
caregivers
cells
virus
viral
wine
mice
alcohol
virus
patients
using
```

Search Query:

Users can enter the search query on the command prompt, output will be a list of documents with their IDs. For the input “Methods for coordinating data-gathering with standardized nomenclature”. We observed that more words in the user query matched with keywords from cluster 1 and output document can be seen in the adjacent figure.

```
Enter: Methods for coordinating data-gathering with standardized nomenclature.
Document ID is PMC7108572
Document ID is PMC7108471
Document ID is PMC7149786
Document ID is PMC7090783
Document ID is PMC7157466
Document ID is PMC7127629
Document ID is PMC6669655
Document ID is PMC3974838
Document ID is PMC3660175
Document ID is PMC7094657
Document ID is PMC6084361
Document ID is PMC7080050
Document ID is PMC7108285
Document ID is PMC4005712
Document ID is PMC7127038
Document ID is PMC7105175
Document ID is PMC7124324
Document ID is PMC7114828
Document ID is PMC7128082
Document ID is PMC4681270
Document ID is PMC4179854
Document ID is PMC7167034
Document ID is PMC7137925
Document ID is PMC7091382
Document ID is PMC7173517
Document ID is PMC5499171
Document ID is PMC3342007
Document ID is PMC7119600
Document ID is PMC7123247
Document ID is PMC7110209
Document ID is PMC7131559
Document ID is PMC7125861
Document ID is PMC4296680
```

Conclusion:

Clustering helps group similar documents which are otherwise immediately unnoticeable to the human eye. As such, rather than trying to evaluate the relevance of a search query across each of the documents in the database, a more effective method would involve searching clusters. The latter approach significantly outperforms in runtime, a critical factor when providing scalable solutions. Through approaches like Topic modelling, we can obtain a list of topics and their keywords, which can serve as an effective summary of a group of documents. If we view a user query as something that is related to a specific topic, our approach comes in handy to find relevant documents (cluster) for the given query.

In sum, by combining various individual techniques we can build a more accurate and scalable solution. As detailed earlier, Clustering and Topic modelling helped us achieve our goals far more effectively than traditional vector space techniques, which would involve creating vectors for each document and then directly using them to provide results for a given search query.

References:

- https://www.kaggle.com/allen-institute-for-ai/CORD-19-research-challenge
- https://medium.com/analytics-vidhya/building-a-coronavirus-research-literature-search-engine-3fae767b41ba
- https://towardsdatascience.com/topic-modeling-and-latent-dirichlet-allocation-in-python-9bf156893c24
- https://www.kaggle.com/maksimeren/covid-19-literature-clustering
- https://spark.apache.org/docs/latest/mllib-feature-extraction.html
- https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.LatentDirichletAllocation.html