CSCE 771 Project Report Yunya Zhang

Project Title

Identifying documents related to a COVID-19 medical procedure

1. Background

Given that COVID-19 still spreads rapidly and large number of literatures and documents related to COVID-19 are published everyday and everywhere, it is difficult for health professionals, sanitation staff, and many other essential personnel to keep up with new information on the virus while they are out there fighting the virus and keeping the world afloat.

In response to the COVID-19 pandemic, the White House and a coalition of leading research groups have prepared the COVID-19 Open Research Dataset (CORD-19), which we will use for this project. CORD-19 keeps being updated and is currently around 20 GB with 280,000 scholarly articles about COVID-19, SARS-CoV-2, and related coronaviruses. This freely available dataset is provided to the global research community to apply recent advances in natural language processing and other AI techniques to generate new insights in support of the ongoing fight against this infectious disease.

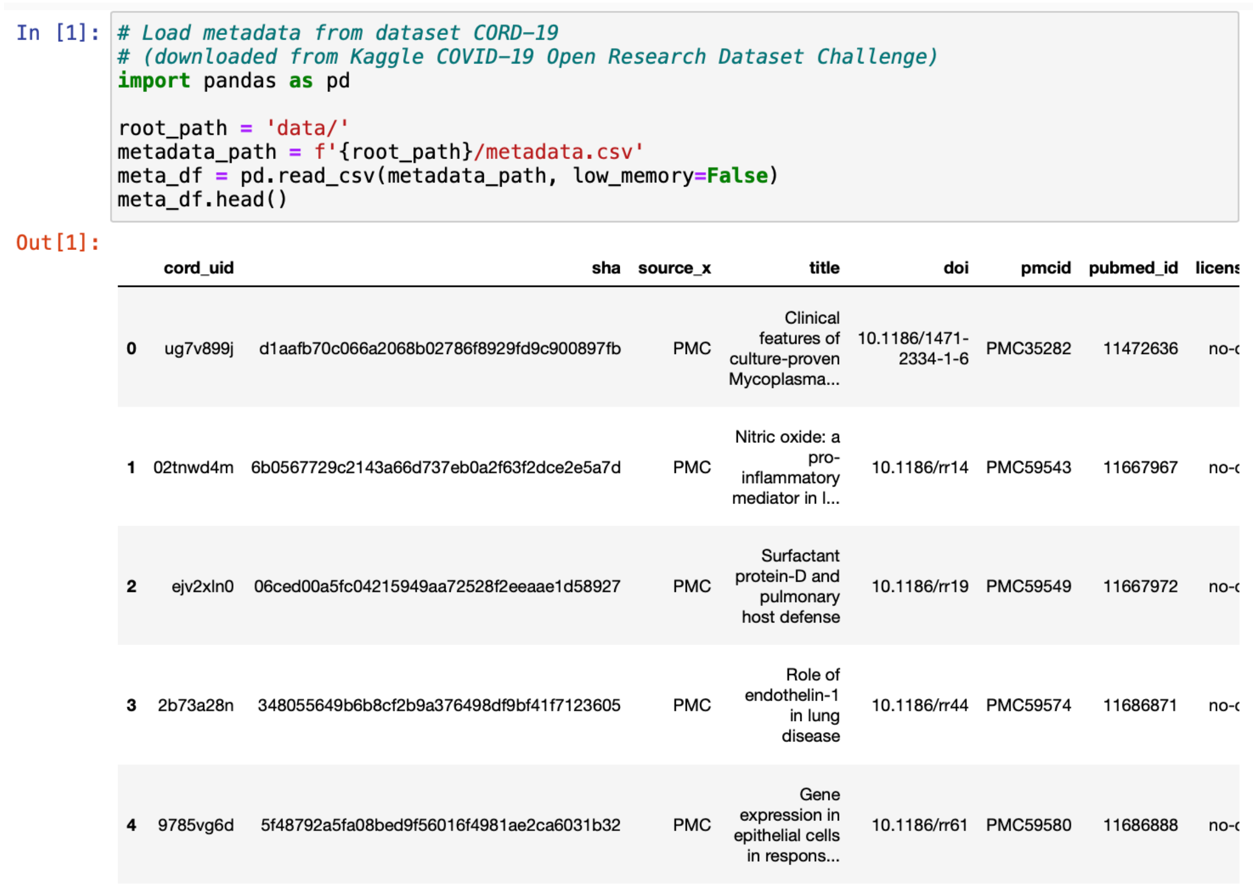
2. Problem

The aim of this project is to analyze documents that are related to COVID-19, and identify whether any of them is related to a COVID-19 medical procedure by searching keywords, questions, or paragraph, which will make it easier for trained professionals to sift through tons of publications.

3. Approach

3.1 Loading the data

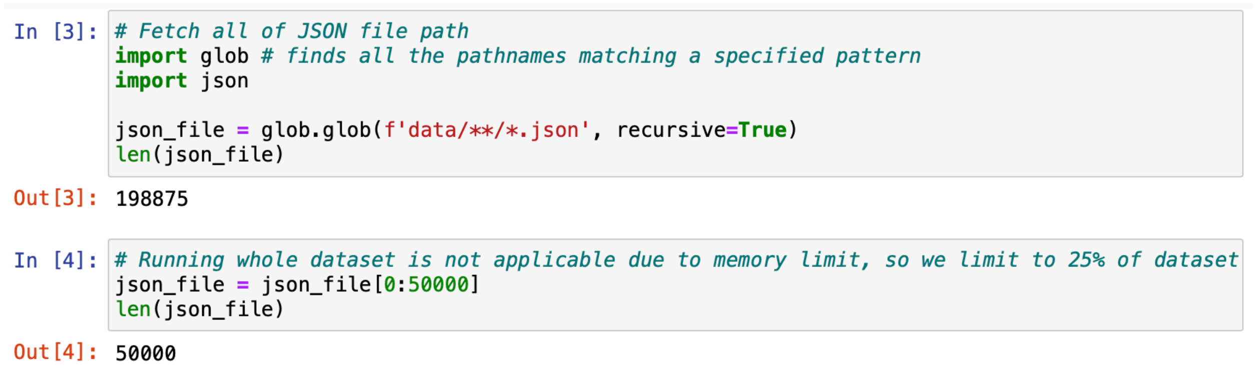
The metadata of the dataset was first loaded to check what information has been provided for the CORD-19 dataset. Among 19 attributes, we use “title” to link “url” with each document in the JSON files. Since we need to analyze the actual content of the documents, we set path to fetch all JSON files. Unfortunately, since running the next steps on the whole dataset (200,000 instances of JSON files) is not applicable due to computer memory limit, so that we limit to 25% of the dataset (50,000 instances). Then, with the help of file reader function, the documents were read into a DataFrame that can be used easily with the attributes of only paper_id, title, body_text, and url.



```
In [1]: # Load metadata from dataset CORD-19
        # (downloaded from Kaggle COVID-19 Open Research Dataset Challenge)
        import pandas as pd

        root_path = 'data/'
        metadata_path = f'{root_path}/metadata.csv'
        meta_df = pd.read_csv(metadata_path, low_memory=False)
        meta_df.head()
```

Out[1]:

	cord_uid	sha	source_x	title	doi	pmcid	pubmed_id	licens
0	ug7v899j	d1aafb70c066a2068b02786f8929fd9c900897fb	PMC	Clinical features of culture-proven Mycoplasma...	10.1186/1471-2334-1-6	PMC35282	11472636	no-c
1	02tnwd4m	6b0567729c2143a66d737eb0a2f63f2dce2e5a7d	PMC	Nitric oxide: a pro-inflammatory mediator in l...	10.1186/rr14	PMC59543	11667967	no-c
2	ejv2xln0	06ced00a5fc04215949aa72528f2eeaae1d58927	PMC	Surfactant protein-D and pulmonary host defense	10.1186/rr19	PMC59549	11667972	no-c
3	2b73a28n	348055649b6b8cf2b9a376498df9bf41f7123605	PMC	Role of endothelin-1 in lung disease	10.1186/rr44	PMC59574	11686871	no-c
4	9785vg6d	5f48792a5fa08bed9f56016f4981ae2ca6031b32	PMC	Gene expression in epithelial cells in respons...	10.1186/rr61	PMC59580	11686888	no-c



```
In [3]: # Fetch all of JSON file path
        import glob # finds all the pathnames matching a specified pattern
        import json

        json_file = glob.glob(f'data/**/*.json', recursive=True)
        len(json_file)

Out[3]: 198875

In [4]: # Running whole dataset is not applicable due to memory limit, so we limit to 25% of dataset
        json_file = json_file[0:50000]
        len(json_file)

Out[4]: 50000
```

In [5]:
```python
# Helper function to read JSON file based on json_schema.txt in the dataset
# We only need 'paper_id', 'title', and 'body_text' from JSON file
# Not all the documents have abstract, so we don't consider it

class FileReader:
    def __init__(self, file_path):
        with open(file_path) as file:
            content = json.load(file)
            self.paper_id = content['paper_id']
            self.title = content['metadata']['title']
            # combine body text in different sections
            self.body_text = []
            for entry in content['body_text']:
                self.body_text.append(entry['text'])
            self.body_text = '\n'.join(self.body_text)
    def __repr__(self):
        return f'{self.paper_id}: {self.title}...{self.body_text[:200]}...'
first_row = FileReader(json_file[0])
print(first_row)
```

```
PMC7405720: Potential of combating transmission of COVID-19 using novel self-cleaning super
hydrophobic surfaces: part II—thermal, chemical, and mechanical durability...COVID-19 can b
e transmitted through airborne respiratory droplets, ejected as a result of coughing or sne
ezing through human contact with contaminated surfaces (Yang and Wang 2020; Gralinski and M
enac...
```

In [6]:
```python
# Read the documents into DataFrame that can be used easily
# Combine 'url' in metadata with other necessary information in JSON files

dict_file = {'paper_id': [], 'title': [], 'body_text': [], 'url': []}
for idx, entry in enumerate(json_file):
    if idx % (len(json_file) // 10) == 0:
        print(f'Processing index: {idx} of {len(json_file)}')

    content = FileReader(entry)
    dict_file['paper_id'].append(content.paper_id)
    dict_file['title'].append(content.title)
    dict_file['body_text'].append(content.body_text)

    # get 'url from metadata information
    meta_data = meta_df.loc[meta_df['title'] == content.title]
    if len(meta_data['url']) == 0:
        dict_file['url'].append('NaN')
    else:
        # if there are multiple url, just take the first one for later use in HTML reference
        if str(meta_data['url'].values[0]).find(';')!= -1:
            url = meta_data['url'].values[0].split(';')[0]
            dict_file['url'].append(url)
        else:
            dict_file['url'].append(meta_data['url'].values[0])

df = pd.DataFrame(dict_file, columns=['paper_id', 'title', 'body_text', 'url'])
```

In [7]:
```python
df.head()
```

Out[7]:

	paper_id	title	body_text	url
0	PMC7405720	Potential of combating transmission of COVID-1...	COVID-19 can be transmitted through airborne r...	https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7...
1	PMC7463272	Editorial Commentary: It Takes Two to Tango: T...	Return to sport after meniscal surgery, as aft...	https://www.ncbi.nlm.nih.gov/pubmed/32891247/
2	PMC7091850	The impact of COVID-19 on the provision of don...	The provision of donors for life-saving hemato...	https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7...
3	PMC7188043	The Response of the Anesthesia & Analgesia Com...	Name: Thomas R. Vetter, MD, MPH.\nContribution...	https://www.ncbi.nlm.nih.gov/pubmed/32332295/
4	PMC7478798	Treatment to reduce vascular calcification in ...	It is well recognized that coronary artery cal...	NaN

3.2 Data Pre-processing

The text was then cleaned up to remove any documents that are not English, and any math packages used in the documents. The text body was then parsed and tokenized, and the stop words was found and removed using Spacy. However, the stop words in Spacy are the words that are used in everyday English text, but research papers will often use words that do not actually contribute to the meaning and are not considered everyday stop words. In this case, we add some customized stop words to further clean up research papers.

```
In [8]: # Handle multiple languages. Drop any language that is not English.
        # Determine the language of each paper in the dataframe.

        from tqdm import tqdm # show progress bar
        from langdetect import detect
        from langdetect import DetectorFactory

        # set seed to enforce consistent results
        DetectorFactory.seed = 0

        # hold label - language
        languages = []

        # go through each text
        for i in tqdm(range(0,len(df))):
            # split by space into list, take the first x index, join with space
            try:
                lang = detect(df.iloc[i]['body_text'])
            except Exception as e:
                lang = "unknown"
                pass
            languages.append(lang)
```

```
100%|██████████| 50000/50000 [20:00<00:00, 41.64it/s]
```

```
In [9]: # Count each language in the dataset
        languages_dict = {}
        for lang in set(languages):
            languages_dict[lang] = languages.count(lang)

        print("Total: {}\n".format(len(languages)))
        print(languages_dict)
```

```
Total: 50000

{'lt': 1, 'pt': 6, 'it': 249, 'da': 1, 'en': 48701, 'zh-cn': 10, 'nl': 70, 'es': 167, 'ja':
1, 'so': 2, 'et': 1, 'af': 1, 'no': 49, 'de': 457, 'tl': 1, 'fr': 179, 'sw': 3, 'hu': 1, 'c
a': 11, 'unknown': 84, 'pl': 3, 'id': 1, 'sq': 1}
```

```
In [10]: # Drop any language other than English
         df['language'] = languages
         df = df[df['language'] == 'en']
```

```
In [15]: # Clean up the text
         import scispacy
         import spacy
         import en_core_sci_lg  # for processing biomedical, scientific or clinical text

         nlp = en_core_sci_lg.load(disable=["tagger", "parser", "ner"])
         nlp.max_length = 8000000
```

```
In [16]: import re

         # Convert the text to lowercase
         df['processed_text'] = df['body_text'].map(lambda x: x.lower())

         # remove the math packages used in the documents
         df['processed_text'] = [re.sub('\\+\S*\b', '', sent) for sent in df['processed_text']]

         # Remove new line characters
         df['processed_text'] = [re.sub('\s+', ' ', sent) for sent in df['processed_text']]
```

```
In [17]: def spacy_tokenizer(sentence):
             return [word.lemma_ for word in nlp(sentence) if not (word.like_num or word.is_stop or w
```

```
In [18]: # customized stop word for scientific documents
         customize_stop_words = [
             'doi', 'preprint', 'copyright', 'org', 'https', 'et', 'al', 'author', 'figure', 'table',
             'rights', 'reserved', 'permission', 'use', 'used', 'using', 'biorxiv', 'medrxiv', 'licen
             '-PRON-', 'ml', 'mg'
         ]

         # Mark them as stop words
         for w in customize_stop_words:
             nlp.vocab[w].is_stop = True
```

3.3 Vectorization

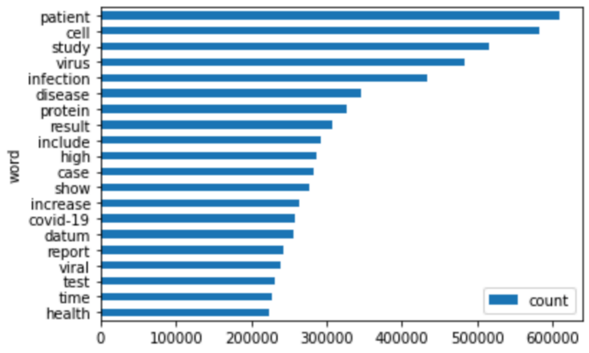
Each document was then vectorized into a feature vector using Term Frequency (TF) Vectorizer from scikit-learn, so that the string formatted document was converted into a matrix of token counts. Most frequent words are shown in the figure.

TF-IDF is considered to provide better vectorization and was tried in this project as well, but it did not work out in the later finding related documents steps. I think this is because the search questions are usually too short. TF-IDF is not applicable to the corpus with too little tokens. TF-IDF will be further studied with longer search contents such as paragraph or other document in the future work.

```
In [19]: # Convert a collection of text documents to a matrix of token counts
         # Tried TF-IDF, but didn't work out in the later topic modeling step

         from sklearn.feature_extraction.text import CountVectorizer

         all_texts = df['processed_text']
         count_vectorizer = CountVectorizer(tokenizer = spacy_tokenizer, min_df=2)
         data_vectorized = count_vectorizer.fit_transform(tqdm(all_texts))

         100%|██████████| 48701/48701 [19:20<00:00, 41.97it/s]
```

```
In [20]: data_vectorized.shape

Out[20]: (48701, 451961)
```

```
In [21]: # Most frequent words
         import numpy as np

         word_count = pd.DataFrame({'word': count_vectorizer.get_feature_names(), 'count': np.asarray
         word_count.sort_values('count', ascending=False).set_index('word')[:20].sort_values('count',

Out[21]: <AxesSubplot:ylabel='word'>
```



3.4 Topic Modeling

Topic Modeling was applied on this vectorized documents using Latent Dirichlet Allocation (LDA) to discover topics for each document. LDA learns the relationships between words, topics, and documents by assuming documents are generated by a particular probabilistic model. In LDA, each document can be described by a distribution of topics and each topic can be described by a distribution of words. This allows us to view each article as a mixture of these topics. The problem with interpreting topics this way is that common terms in the corpus often appear near the top of such lists for multiple topics, making it hard to differentiate the meanings of these topics.

In [23]:
```python
# Use LDA (Latent Dirichlet Allocation) where each document can be described by a
# distribution of topics and each topic can be described by a distribution of words

from sklearn.decomposition import LatentDirichletAllocation

# Reset the number of topics = 50 based on the perplexity value from evaluation in the last
lda_model = LatentDirichletAllocation(n_components=50, random_state=42, n_jobs=-1)
lda_output = lda_model.fit(data_vectorized)
joblib.dump(lda_output, 'lda.csv')
```

Out[23]:
```
['lda.csv']
```

In [24]:
```python
# Helper function to print topics to take a look
def print_top_words(model, vectorizer, n_top_words):
    feature_names = vectorizer.get_feature_names()
    for topic_idx, topic in enumerate(model.components_):
        message = "\nTopic #%d: " % topic_idx
        message += " ".join([feature_names[i]
                             for i in topic.argsort()[:-n_top_words - 1:-1]])
        print(message)
    print()
```

In [25]:
```python
print_top_words(lda_model, count_vectorizer, n_top_words=25)
```

```

Topic #0: ace2 brain increase receptor effect heart rat mouse expression neuron cardiac ii
muscle ace study level ang kidney cell function injury animal angiotensin tissue model

Topic #1: cell expression mouse protein level group gene control show increase significantl
y result induce study usa analysis pathway response effect compare datum treatment infectio
n min apoptosis

Topic #2: air particle temperature concentration study flow water high filter area show aer
osol time rate droplet increase condition size system result low environment large exposure
pressure

Topic #3: cell infection response mouse immune expression virus cytokine viral cd8 macropha
ge cd4 type receptor induce infect gene human express study increase show activation antige
n production

Topic #4: patient infection pneumonia study antibiotic child isolate bacterial result s. cl
inical asthma resistance strain group hospital method test antimicrobial treatment culture
increase pathogen high bacterium
```

In [26]:
```python
# Each article is a mixture of topics / a distribution over topics
doc_topic_dist = pd.DataFrame(lda_model.transform(data_vectorized))
doc_topic_dist.to_csv('doc_topic_dist.csv', index=False)
```

In [27]:
```python
doc_topic_dist.head()
```

Out[27]:

	0	1	2	3	4	5	6	7	8	9	...	40	41
0	0.000018	0.000018	0.015816	0.000018	0.000018	0.000018	0.000018	0.000018	0.013102	0.000018	...	0.085247	0.616780
1	0.000041	0.000041	0.013539	0.000041	0.000041	0.000041	0.000041	0.000041	0.085792	0.000041	...	0.000041	0.029389
2	0.000091	0.000091	0.000091	0.000091	0.000091	0.000091	0.000091	0.000091	0.409789	0.000091	...	0.091960	0.000091
3	0.000800	0.000800	0.000800	0.000800	0.171647	0.000800	0.000800	0.000800	0.311416	0.000800	...	0.000800	0.000800
4	0.101039	0.000010	0.011634	0.000010	0.000010	0.000010	0.000010	0.000010	0.057980	0.000010	...	0.000010	0.000010

5 rows × 50 columns

3.5 Finding Related Documents

By mapping what specific question we are looking for into the topic space, we can then find documents related to this specific question. In our case, we use Jensen-Shannon divergence to measure the similarity between documents. In probability theory and statistics, the Jensen–Shannon divergence is a method of measuring the similarity between two probability distributions. It also has the nice property to be bounded by 0 and 1, which is quite convenient for a similarity score. Widgets are created for typing in the question.

```
In [28]: # Check documents to see whether they are related to COVID-19
         # Although the dataset is about COVID-19, just assume that this program can be used to other
         # dataset containing documents rather than COVID-19

         is_covid19_article = df.body_text.str.contains('covid-19|sars-cov-2|2019-ncov|sars coronavir
```

```
In [29]: # Find k nearest documents by Jensen-Shannon divergence in topic space
         # In probability theory and statistics, the Jensen–Shannon divergence is a method of measuri
         # the similarity between two probability distributions.
         # https://medium.com/datalab-log/measuring-the-statistical-similarity-between-two-samples-us

         from scipy.spatial.distance import jensenshannon
         import numpy as np

         def get_k_nearest_docs(doc_dist, k=5, only_covid19=False, get_dist=False):
             global topic_dist

             if only_covid19:
                 topic_dist = doc_topic_dist[is_covid19_article][:len(is_covid19_article)]

             distances = topic_dist.apply(lambda x: jensenshannon(x, doc_dist), axis=1)
             k_nearest = distances[distances != 0].nsmallest(n=k).index

             if get_dist:
                 k_distances = distances[distances != 0].nsmallest(n=k)
                 return k_nearest, k_distances
             else:
                 return k_nearest
```

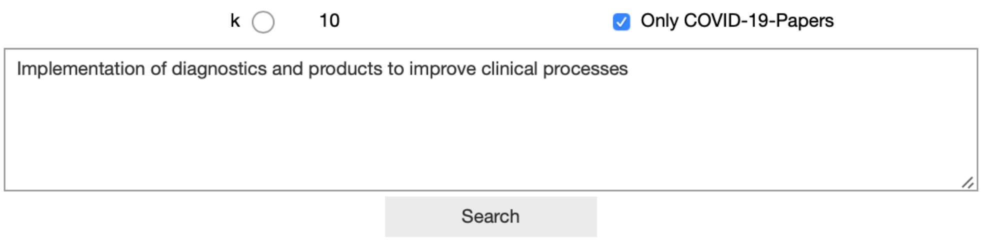
```
In [30]: from IPython.display import HTML, display

         # Helper function to find related articles, given the text of search content
         def relevant_articles(texts, k=10, only_covid19=False):
             text = [texts] if type(texts) is str else texts

             text_vectorized = count_vectorizer.transform(text)
             text_topic_dist = pd.DataFrame(lda_model.transform(text_vectorized))

             for index, bullet in enumerate(text):
                 print(bullet)
                 recommended, dist = get_k_nearest_docs(text_topic_dist.iloc[index], k, only_covid19,
                 recommended = df.iloc[recommended].copy()
                 recommended['similarity'] = 1 - dist

             h = '<br/>'.join(['<a href="' + str(l) + '" target="_blank">'+ n + '</a>' +' (Similarity
             display(HTML(h))
```

```
In [31]: from ipywidgets import interact, Layout, HBox, VBox, Box
         import ipywidgets as widgets
         from IPython.display import clear_output

         # Type in any kind of text (abstract, paragraph, full text, keywords, questions, ...) in the
         # and find related articles.
         def relevant_articles_for_text():
             textW = widgets.Textarea(
                 value='',
                 placeholder='Type something',
                 description='',
                 disabled=False,
                 layout=Layout(width='70%', height='100px')
             )
             covidW = widgets.Checkbox(value=True,description='Only COVID-19-Papers',disabled=False,
             kWidget = widgets.IntSlider(value=10, description='k', max=50, min=1, layout=Layout(widt

             button = widgets.Button(description="Search")

             display(VBox([HBox([kWidget, covidW], layout=Layout(width='70%', justify_content='space-
                 textW, button], layout=Layout(align_items='center')))

             def on_button_clicked(b):
                 clear_output()
                 display(VBox([HBox([kWidget, covidW], layout=Layout(width='70%', justify_content='sp
                     textW, button], layout=Layout(align_items='center')))
                 relevant_articles(textW.value, kWidget.value, covidW.value)

             button.on_click(on_button_clicked)
```

4. Demonstration

Type in the search question, the related documents will be returned. If url is provided in the dataset, click the document will direct to the full document. If more questions would like to be tried, please let me know.

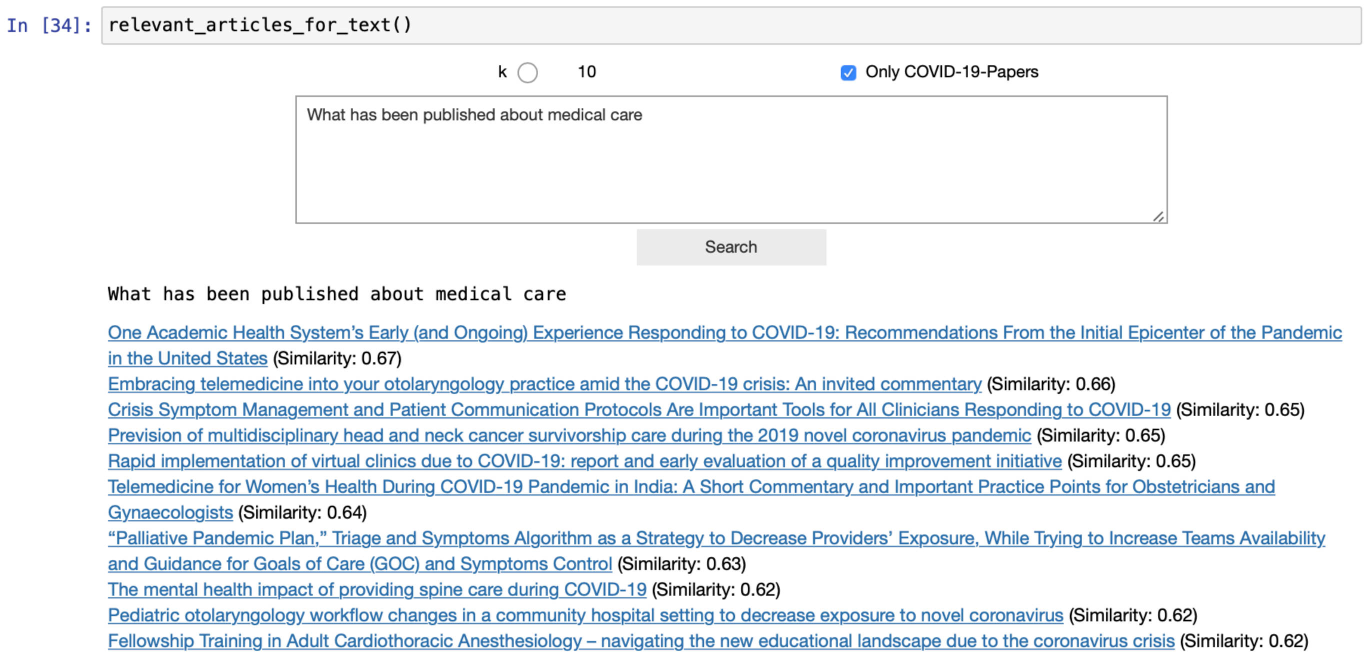
```
In [33]: relevant_articles_for_text()
```



Implementation of diagnostics and products to improve clinical processes

```
Implementation of diagnostics and products to improve clinical processes
```

[MIPs for Commercial Application in Low-cost Sensors and Assays – an Overview of the Current Status Quo] (Similarity: 0.70)
[Magnetic-Nanosensor-Based Virus and Pathogen Detection Strategies before and during COVID-19] (Similarity: 0.61)
[Electroanalysis from the past to the twenty-first century: challenges and perspectives] (Similarity: 0.56)
[Monitoring Proteolytic Activity in Real Time: A New World of Opportunities for Biosensors] (Similarity: 0.55)
[COVID-19 diagnostic approaches: different roads to the same destination] (Similarity: 0.49)
[Detection of COVID-19: A review of the current literature and future perspectives] (Similarity: 0.47)
[Molecular diagnosis of COVID-19: Current situation and trend in China (Review)] (Similarity: 0.45)
[Potential Diagnostic Systems for Coronavirus Detection: a Critical Review] (Similarity: 0.43)
[3D PRINTING OF FACE SHIELDS DURING COVID-19 PANDEMIC: A TECHNICAL NOTE] (Similarity: 0.43)
[COVID-19 diagnostic testing: Technology perspective] (Similarity: 0.41)



```
In [34]: relevant_articles_for_text()
```

k ○ 10 ✓ Only COVID-19-Papers

What has been published about medical care

```
What has been published about medical care
```

One Academic Health System's Early (and Ongoing) Experience Responding to COVID-19: Recommendations From the Initial Epicenter of the Pandemic in the United States (Similarity: 0.67)
Embracing telemedicine into your otolaryngology practice amid the COVID-19 crisis: An invited commentary (Similarity: 0.66)
Crisis Symptom Management and Patient Communication Protocols Are Important Tools for All Clinicians Responding to COVID-19 (Similarity: 0.65)
Prevision of multidisciplinary head and neck cancer survivorship care during the 2019 novel coronavirus pandemic (Similarity: 0.65)
Rapid implementation of virtual clinics due to COVID-19: report and early evaluation of a quality improvement initiative (Similarity: 0.65)
Telemedicine for Women's Health During COVID-19 Pandemic in India: A Short Commentary and Important Practice Points for Obstetricians and Gynaecologists (Similarity: 0.64)
"Palliative Pandemic Plan," Triage and Symptoms Algorithm as a Strategy to Decrease Providers' Exposure, While Trying to Increase Teams Availability and Guidance for Goals of Care (GOC) and Symptoms Control (Similarity: 0.63)
The mental health impact of providing spine care during COVID-19 (Similarity: 0.62)
Pediatric otolaryngology workflow changes in a community hospital setting to decrease exposure to novel coronavirus (Similarity: 0.62)
Fellowship Training in Adult Cardiothoracic Anesthesiology – navigating the new educational landscape due to the coronavirus crisis (Similarity: 0.62)

5. Evaluation

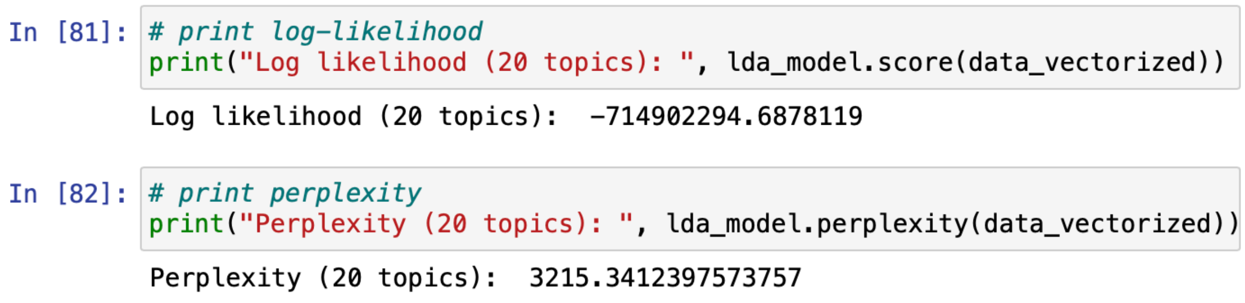
To evaluate whether the documents that were found by the above program is actually related to the search question, we need to go over the contents in the documents. However, it would be difficult to implement especially when we get large amounts of related documents if we search through the whole large dataset (in this project, we only search through 25% of the dataset and showed 10 related documents for each search content). Therefore, in this case, we will evaluate the topic modeling step which will lead to provide the related documents.

The latent space discovered by these topic models is generally meaningful and useful, but evaluating such assumptions is challenging due to its unsupervised training process. Besides, there is a no-gold standard list of topics to compare against every corpus. However, we need to identify if a trained model is objectively good or bad, and we also need to have an ability to compare models with different parameters. Traditionally, and still for many practical applications, "eyeballing" approaches of top N words or topics are used to evaluate if correct information has been learned about the corpus. But ideally, we would like to capture this information in a single metric that can be maximized and compared.

The model performance in our case was evaluated using log-likelihood and perplexity (exp(-1. * log-likelihood per word)), which is a built-in feature in scikit-learn LDA. Perplexity is one of the intrinsic evaluation metric, and is widely used for language model evaluation. Perplexity metric can be considered as measuring how probable some new unseen data is given the model that was learned earlier. That is to say, how well does the model represent or reproduce the statistics of the held-out data.

As I went along with the project and searched more information online, I noticed that recent studies have shown that perplexity and human judgement are often not correlated. Optimizing for perplexity may not provide us human interpretable topics. In stead, topic coherence measurement is suggested to be a better way to evaluate topic modeling. Topic coherence measures score a single topic by measuring the degree of semantic similarity between high scoring words in the topic. However, topic coherence is a built-in feature in Gensim, and is not carried by scikit-learn. Since this project started by using scikit-learn, we sticked to use perplexity to evaluate our models. Investigation of topic coherence measurement can be the future work.

I was trying to use GridSearch by changing number of topics (10, 20, 30, 40, 50) and learning decay (0.5, 0.7, 0.9) to find the best number of topics, but it took 6 hours and still had not finished due to large amount of data. Thus, in stead, I just manually changed the number of topics in the LDA step without altering learning decay to obtain different perplexity for comparison roughly. Generally, a model with higher log-likelihood and lower perplexity is considered to be good. We can see that 50 topics give us higher likelihood and lower perplexity, so that this is why we run the LDA with 50 topics in the earlier steps.



```
In [81]: # print log-likelihood
         print("Log likelihood (20 topics): ", lda_model.score(data_vectorized))

         Log likelihood (20 topics):  -714902294.6878119

In [82]: # print perplexity
         print("Perplexity (20 topics): ", lda_model.perplexity(data_vectorized))

         Perplexity (20 topics):  3215.3412397573757
```

```
In [65]: # print log-likelihood
         print("Log likelihood (30 topics): ", lda_model.score(data_vectorized))

         Log likelihood (30 topics):  -709967003.7033776

In [66]: # print perplexity
         print("Perplexity (30 topics): ", lda_model.perplexity(data_vectorized))

         Perplexity (30 topics):  3040.9908279428296
```

```
In [89]: # print log-likelihood
         print("Log likelihood (40 topics): ", lda_model.score(data_vectorized))

         Log likelihood (40 topics):  -706579114.8913338

In [90]: # print perplexity
         print("Perplexity (40 topics): ", lda_model.perplexity(data_vectorized))

         Perplexity (40 topics):  2926.809957154544
```

```
In [98]: # print log-likelihood
         print("Log likelihood (50 topics): ", lda_model.score(data_vectorized))

         Log likelihood (50 topics):  -703622194.1575209

In [99]: # print perplexity
         print("Perplexity (50 topics): ", lda_model.perplexity(data_vectorized))

         Perplexity (50 topics):  2830.663314451241
```

6. Discussion

6.1 Achievement

Through topic modeling by Latent Dirichlet Allocation (LDA), 50 topics were discovered from the vectorized documents of 25% of CORD-19 dataset. By typing in keywords, questions, or even abstract, paragraph, full text about medical procedure, the documents that are related to the search content were identified and ranked by a similarity score. The topic model that was used to help find related documents was evaluated by measuring scores of likelihood and perplexity.

6.2 Challenges

My undergraduate is not computer science and have not used Python before. The two graduate courses I have taken are C++ homework based. The major challenge for me is to learn how to use Python and its various packages, and learn how to do a project given the limited time with other courses and other family stuff going around. I have used some code from references and tried different methods, and spent most of the time to modify the code and make the code working for this project since it is kind of difficult for me to figure out how to fix the errors. Through the whole process, I made some progress in coding and was glad to apply some knowledge of what I have learned from the class into this project.

6.3 Future work

The only way to do this project I can think of is topic modeling since this CORD-19 dataset is quite new and not labeled. There are probably other better ways to do this project. After I got comments and insights from the professor and other students, I will further try other better approaches. TF-IDF should perform better than TF vectorization, so later I will continue to try TF-IDF with longer corpus of search content. I will also investigate topic coherence measurement in Gensim to better evaluate this project.