Contents

1 Abstract

The purpose of this project is to create a search bar for User to get the most relevant data from all the articles that have been gathered related to Covid-19. So we are to create NLP (Natural Language processing) model, user could either give a question through command line or web-app and using our model and we are to get the most relevant article and sentence from that article based on the question asked. Since dataset contains millions of articles relating to Covid-19 or similar disease, we had a constraint on the computational powers not on data. So for this project, I choose 10,000 articles then dropped all the articles which had any null values present in the complementary fields and I thought after this project I could continue to complete few other tasks of this Kaggle competition, so after weeding those out I had around 7,500 articles left.

As always, the first step of this project was to explore the data before making any assumptions. So the activities involved in performing EDA (Exploratory Data Analysis) were, checking for null values or duplicates if any. Checking the distribution of data based on the time when those articles were created. Then I plotted few wordcloud's and frequency plot (n-grams) to better understand how words are distributed among our articles. I found out there were many impurities (junk data), so to find out more about those, I checked for mathematical formulas, HTML tags, punctuation marks and emojis.

So based on insights gained from EDA it was clear that my data was too impure to directly pass into a model. As if u pass junk and train your model on it then your prediction would also be useless. Thus, I started with creating a few functions for replacing maths equations and urls with a tag, cleaning punctuation marks and emojis, correcting misspells, changing word contractions, lemmatizing the data and removing stop words like is, a, of etc. which adds no meaning to a sentence when we are training a basic model. So now my data was ready to be used for training purpose, next step was to learn and investigate about few models mostly about the theoretical working and parameters that they used instead of complex mathematical formulas. Next step was to train few different models and evaluate, then compare their performance.

2 Introduction

This project involves creating a NLP model for searching similar sentences from the Covid-19 dataset called cord. This dataset can be found on Kaggle and it contains millions of articles related to Covid-19 or similar kind of disease. We are to get a query from user and then traverse through our data to find the most similar sentences and return top-10 results.

Since the dataset was too large, I was not able to load it on my machine (Jupyter notebook). So I came up with an idea to create Kaggle notebook on which I would read just around 7500 articles from millions of them. In this notebook I also did some preprocessing like removing all the null value rows, since we just needed a subset of the data. I also dropped unnecessary columns, since the main dataset consists of many columns which are not relevant to our project. The main reason to choose the subset of the articles was computational power constraint (I have 16gb ram and 8gb graphics) due to this too large data would take hours if not days to train Doc2Vec and Word2Vec models.

Now, since I got the csv file which is way easier to deal with, I loaded the data and started with EDA, in which I found out my data did not have any null or duplicate values but since it was text data it had a lot of misspells, contractions, punctuation marks, emojis, mathematical formulas, HTML tags, and urls. So my next step involved creating different functions to handle above problems and get a clean data that can be fed to my model. For training and testing part I tried few different models and used cosine similarity as validation method since we needed to find top 10 most similar sentences with the user's input.

3 Methodology

3.1 EDA

Understanding the data or getting the feel of data by exploring the data set is an important part of project, because it helps us to find errors or mistakes in our data sets and gain valuable insight to achieve clean data which is ready to be used by our models. So the more we understand our data, the more features we can extract from it.

I plotted few graphs which indicated how many articles were published on which dates. Then I used Wordcloud library to show which words are most prevalent in title, abstract, body_text fields. After these wordcloud images, I created few functions to get n-gram graphs (frequency plots), which shows most frequent words in whole corpus of data.

3.2 Data Preprocessing

This step is the most important step of all, if this is not executed properly we might still have junk values which would decrease the accuracy or might make our model over fit. This task is all about cleaning the data of unnecessary information (punctuation marks, emojis, formula etc.). This happens because data is gathered from different users or multiple platforms and it is in raw format, so this data is not viable for analysis.

To begin this process, I created sentence and sentence_embedding columns using nltk library from body_text column. Since once these columns gets preprocessed it would have removed all new line characters as well as ".", which are needed in order to separate each sentences.

Then I created a few functions to clean:

- Math's Equations, URL's
- Punctuation marks
- Contractions
- Lemmatization

3.3 Research Models

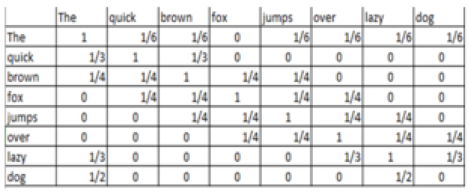
Now we have clean data to be used for training purpose, but we still need to figure out which models to use. Since there are a lot of models out there today, so the best way is to first know what kind of problem we are to solve. Since in this case we want to find similarity between sentences, we can use Doc2Vec, Word2vec, Sentence Transformer and there are few more.

3.3.1 Word 2 Vector (Word2Vec):

Word2vec is a technique for natural language processing (NLP) published in 2013. It was created by a Google research team led by Tomas Mikolov. Word2Vec is based on neural networks like most of deep learning algorithms, it is also unsupervised learning algorithm. Word2Vec is used to convert words into vector representation. This algo works by recognizing linear substructures or patterns in word vector spaces, to gain valuable insights it needs to be trained on large corpus or size of text data.

In this model, we start by selecting a fixed window size(how many words before or after the word we want to consider) and then getting a probability matrix of words or data with respect to each word appearing in that window size. For example:
"The quick brown fox jumps over the lazy dog."

Now let‘s assume that window size is 4 then while calculating the probability matix, we would consider 4 words on left and 4 words on right side of that particular word. Let's start with "The", by calculating the probability of all other words appearing around it, we created the probability vector for that word. Doing same for other words in the corpus.[1]



	The	quick	brown	fox	jumps	over	lazy	dog
The	1	1/6	1/6	0	1/6	1/6	1/6	1/6
quick	1/3	1	1/3	0	0	0	0	0
brown	1/4	1/4	1	1/4	1/4	0	0	0
fox	0	1/4	1/4	1	1/4	1/4	0	0
jumps	0	0	1/4	1/4	1	1/4	1/4	0
over	0	0	0	1/4	1/4	1	1/4	1/4
lazy	1/3	0	0	0	0	1/3	1	1/3
dog	1/2	0	0	0	0	0	1/2	0

[1]

Given a sequence of training words, the objective of the word vector model is to maximize the average log probability.

$$\frac{1}{T}\sum_{t=k}^{T-k}\log p(w_t|w_{t-k},...,w_{t+k})$$

[2]

The prediction task is typically done via a multiclass classifier, such as SoftMax.

$$p(w_t|w_{t-k},...,w_{t+k}) = \frac{e^{y_{w_t}}}{\sum_i e^{y_i}}$$

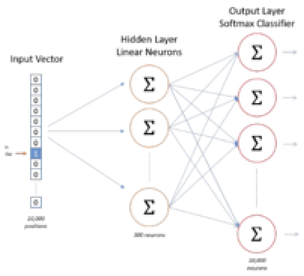
[2]

In the above equation, each yi can be calculated by taking the log probability for each output words (un-normalized) and softmax parameters are U;b. As well as h can be derived by concatenation of average of word vectors extracted from Words.

$$y = b + Uh(w_{t-k},...,w_{t+k};W)$$

[2]

This probability vector for each word in the corpus becomes the target while training a neural network and the one hot embedding vector becomes the input layer in the mode [2]. Check below image to see the architecture of Word 2 Vec:

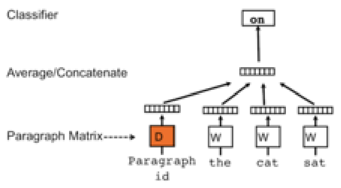


[1] Architecture of Word2Vec Model

The Word2Vec algo works by creating distributed semantic representation of words. And there are mainly 2 methods to achieve that result, 1) Continuous Bag of Words (CBoW) and 2) The skip gram model. The first methods works by predicting the context words using center word, while second is a direct contradiction of first. The second method works by predicting the words using the context words. So, the other variant of Word2Vec is Word2Vec -CBOW (Continuous bag of words) in this case instead of only using 1 word's one hot encoding we use 2-3 words as the name suggests. You can read about it in much detail in Mikolov's paper [3]

3.3.2 Document 2 Vector (Doc2Vec):

Doc2vec is same as word2vec, it is also a natural language processing (NLP) tool, which is used to represent documents as a vector and works as a generalized copy of word2vec model. Doc2vec models' main goal or objective is to create a numeric representation of documents in the corpus, regardless of its length. But the main problem or issue with this is that documents doesn't have any logical structures like words, so they needed to come up with another method. And the concept that Mikolov and Le used was simple, they added a new vector called paragraph id to word2vec model.



Now we have a field that can hold what is missing from the current context – or as the topic/title of the paragraph. Now 2 vectors, word vector and document vector has different info, one represents the concepts of a word while the other represents the concept of a document. If we compare the 2 algo's/models Doc2Vec is faster and saves usage of memory as well compared to Word2Vec, since there is no need to save each and every word vectors.

The idea of creating this model came from the fact that each word contribute some semantics information to the next words. For example: crane- it can be a bird or a heavy machinery, if lets say crane was flying, crane is doing an excavation are 2 sentences then the next words add meaning to other words as well.

To train a doc2vec model you're required to give a set of documents as input and from that document corpus, a word vector W is generated for each word and document vector D is generated for each documents. Model also needs to train weights for its softmax activation layer. For prediction stage, a new document needs to be inputted and then all weights are fixed to calculate the document vector.

3.3.3 Sentence Transformation (SBERT):

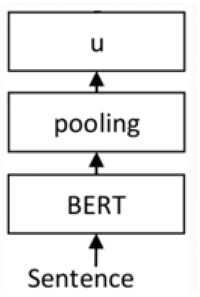
In a recent paper published by researchers at Google AI put forward a model named BERT (Bidirectional Encoder Representations from Transformers), this paper put out quite a stir in Machine Learning community since this was a state of art model whose results in wide variety of Natural Language Processing(NLP) tasks, including natural language inference, sentence similarity, question answering and others are quite higher than other conventional models. If you wish to learn more about this model, please check this link[4].

We can use BERT to measure similarity between 2 sentences, we could achieve that by concatenating both the sentences together and putting [SEP] token in between, then feed them to BERT. We can then use a simple classifier or regressor on the other end with [CLS] token to tell us how those sentences were related. But there are 2 main scenarios and problems:

- In a dataset of n sentences, find 2 most similar sentences: Now we would have to create n(n-1)/2 pairs of sentences and then feed each of them into BERT to find its similarity scores and then compare/sort based on those scores. For example, lets say there are 15,000 sentences in the dataset then it would take 60+ hours to do the computation with good GPU.
- Doing a semantic search: In this case lets say user provides a query and we need to find most similar sentences from our corpus of data. Ideally, this should be done in n time, since all we have to do is compare the query will all existing sentences. But this is also not practically possible to deploy on raspberry pi or phone.

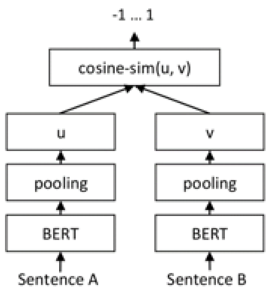
Now let's assume you could pre-compute embeddings for all the sentences in the corpus separate from others. These embeddings could be used to measure similarity in the next step like using cosine similarity function thus we wouldn't need to use BERT to perform these task each and every time. This is what SBERT does in a nut shell. Now let's see in a little detail how it works.

Fine tuning the sentences/text embeddings is way easier in Sentence Transformers than other models. Just like we did in BERT, we feed the input sentences into the transformer. This gives us contextualized word embeddings for all input sentence tokens in out texts. And pooling layer is required since we want a fixed sized output vectors(u). There are many pooling layer options available to choose from, the most basic one is mean-pooling. In this layer we simply get the average of all the contextualized word embeddings that BERT is providing us. This way we can reduce the input text however long to a fixed sized output.[4]



[5] Most simple architecture for sentence transformer

Last but not the least, loss function layer which is very crucial in fine tuning the model. This layer determines how well our embeddings and model would work for a specific downstream task or application. As we know to fine tune our model, we need to tell our model which sentence pairs are actually similar and should be close in vector space. Similarly, which pairs are dissimilar and should be far apart from each other in vector space.



[5] SBert architecture to find similar sentences

3.3.4 Evaluation

Since our main objective in this project was to find most similar sentences and articles we are to use cosine similarity as evaluation metric, it measures how similar the sentences/documents are irrespective of the document size. It is measured by finding cosine angle between the 2 given vectors represented in a multi-dimensional vector space.

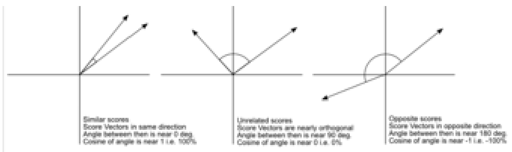
The most common approach during earlier times was to count the maximum number of common words between the documents. But there was a big flaw in this approach, that is as size of the document increased, the number of common words would obviously increase even if the document are based on different topics. To overcome this challenge in "count-the-common-words" or "Euclidean distance approach", we use cosine similarity approach.

This metric of the angle is a measurement of orientation (as it would suggest by angle) and not magnitude. Since we are not only taking TF-IDF (magnitude of each word count) of each document, we are doing a comparison between documents on a normalized space. The equation to solve and build the cosine similarity equation as shown below:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

[6] Cosine Function to find similarity

By checking the angle between the 2 vectors, cosine similarity will generate a metric that says how related the 2 documents are, you can check the examples shown below:


Similar scores
Score Vectors in same direction
Angle between then is near 0 deg.
Cosine of angle is near 1 i.e. 100%
Unrelated scores
Score Vectors are nearly orthogonal
Angle between then is near 90 deg.
Cosine of angle is near 0 i.e. 0%
Opposite scores
Score Vectors in opposite direction
Angle between then is near 180 deg.
Cosine of angle is near -1 i.e. -100%

The Cosine Similarity values for different documents, 1 (same direction), 0 (90 deg.), -1 (opposite directions).

[7]

4 Project Implementation

In this step will be listing all the steps or implementation involved to complete that process. And whole description can be seen on jupyter notebook. As I mentioned earlier I created 2 notebooks:

- First notebook was to read metadata file then select top 10,000 rows and read the corresponding articles present as Jason file. Then I dropped all those articles that had null values in any columns, since I wanted to work on other tasks as well after completion of this project. This way I was left with 7984 articles, which I saved as a csv file.
- Second notebook, This is where all the magic happens, I read the csv file which we generated from 1st notebook activities are:

4.1 Activities in Exploratory Data Analysis (notebook 2nd)

- Importing Library and Dataset
- Checking for null or duplicate values
- Checking basic stat like size, column names and actual data
- Checking sample distribution of articles based on the dates
- Plotting word collage of most frequent words in both sincere and insincere class from training data using wordcloud library.
- Plotting frequency plot for 1-gram(single word) , 2-gram(double word), 3-gram(triple word) for title, abstract, body_text columns
- Created a function to see which different punctuation are present in the data

4.2 Activities for Pre-processing data

Creating functions to clean the data of the anomalies found from EDA

- Replacing math equations and URL's with common abbreviation like a MATH EQUATION tag or URL tag.
- Cleaning contractions. Like "ISN'T" is replaced to "Is not", "Would've" is replaced to "Would have".

- Spelling Correction. Like “oragnization” is corrected to “organization”, “Litecoin” is changed to ”bitcoin”.
- Removing punctuations. All punctuations including emojis, symbols etc.
- Removing Stopwords. Words that occur too frequently in English language and adds a very little meaning to the sentence like A, An, The, is etc.
- Using WordNet Lemmatizer, Lemmatization means converting the word to its root word like me can be changed to meus meaning “my”, changing to change etc.
- Also created functions to separate sentences from the articles and then tokenize them

After Clean the data I again visualize it using n-gram plots and then I printed out 1 article’s abstract and body_text from before preprocessing and after preprocessing for easy comparison.

4.3 Modelling

- First used a pretrained word2vec model “word2vec-google-news-300” to find the most relevant sentence for all articles based on input from user and all the sentences we separated in preprocessing task. Also created many functions to keep the code clean and complete this functionality.
- Then using preprocessed data I trained Doc2Vec model, used the input from user to get the top 10 most similar articles and we already had the most similar sentence available for those articles based on cosine similarity scores.
- After successfully complete the task for our project, I tried to experiment with sentence transformations (sbert model). In this case also I used a pretrained model “bert-base-nli-mean-tokens” and was successful in finding the top 10 most similar articles but then didn’t had enough time to integrate the word2vec model to get the similar sentences for those articles.

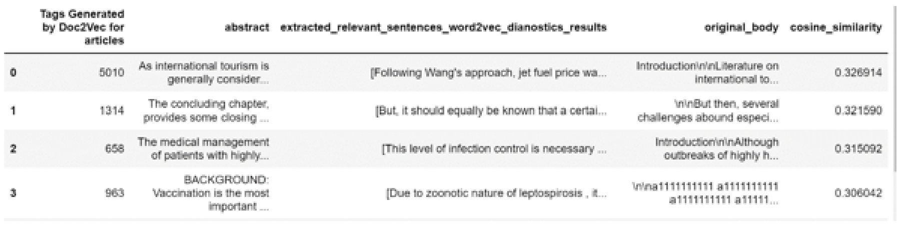
5 Analysis and Solutions

- After EDA it was clear that data had a lot of misspells, math’s equations, URLs, punctuation marks and small preprocessing issues.
- First problem was addressed appropriately in the preprocessing task.
- Next, I tried to create a doc2vec model to get the most similar articles, I was successful in that but then was stuck on finding the similar sentence form those articles.
- So I came up with using word2vec to find the similar sentences for all those articles and then using Doc2vec or sentence transformation to get the top 10 most similar sentences.
- I know this is not the most computational friendly solution but since there was a time crunch I had to settle for this.

6 Conclusion

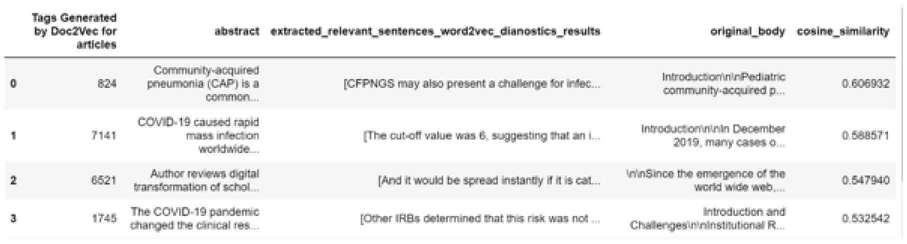
We were successfully able to complete the given task at hand by using mix of word2vec and doc2vec models or we could’ve used sentence transformation technique(sbert) and word2vec models to get the required output just as shown below:

- First we take input from User, he or she is free to ask as many questions as they want. All of those questions will be considered when giving the last output. And we also ask user about which model they want to use among the model trained on abstract or body_text.
- Then using these input we run our corresponding models and get this last output as shown below which has the article’s abstract, body_text, most_similar_sentence from that article and cosine scores of the articles and the tags that were generated while passing in doc2vec.



	Tags Generated by Doc2Vec for articles	abstract	extracted_relevant_sentences_word2vec_dianostics_results	original_body	cosine_similarity
0	5010	As international tourism is generally consider...	[Following Wang's approach, jet fuel price wa...	Introduction\n\nLiterature on international to...	0.326914
1	1314	The concluding chapter, provides some closing ...	[But, it should equally be known that a certai...	\n\nBut then, several challenges abound especi...	0.321590
2	658	The medical management of patients with highly...	[This level of infection control is necessary ...	Introduction\n\nAlthough outbreaks of highly h...	0.315092
3	963	BACKGROUND: Vaccination is the most important ...	[Due to zoonotic nature of leptospirosis , it...	\n\na1111111111 a1111111111 a1111111111 a11111...	0.306042

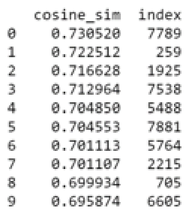
Model trained on Abstract



	Tags Generated by Doc2Vec for articles	abstract	extracted_relevant_sentences_word2vec_dianostics_results	original_body	cosine_similarity
0	824	Community-acquired pneumonia (CAP) is a common...	[CFPNGS may also present a challenge for infec...	Introduction\n\nPediatric community-acquired p...	0.606932
1	7141	COVID-19 caused rapid mass infection worldwide...	[The cut-off value was 6, suggesting that an i...	Introduction\n\nIn December 2019, many cases o...	0.588571
2	6521	Author reviews digital transformation of schol...	[And it would be spread instantly if it is cat...	\n\nSince the emergence of the world wide web,...	0.547940
3	1745	The COVID-19 pandemic changed the clinical res...	[Other IRBs determined that this risk was not ...	Introduction and Challenges\n\nInstitutional R...	0.532542

Model trained on Body_text

- Using the same input, we pass it in sentence transformation (SBERT model) and here is the cosine similarity for that model.



```
   cosine_sim  index
0    0.730520   7789
1    0.722512    259
2    0.716628   1925
3    0.712964   7538
4    0.704850   5488
5    0.704553   7881
6    0.701113   5764
7    0.701107   2215
8    0.699934    705
9    0.695874   6605
```

See the difference in the cosine similarity, it is due to the fact that abstract has way less number of data compared to body_text and due to that model is not able to find similar sentences.

Data Preprocessing was the most crucial part of this project. Passing text data, as it was provided, was not viable since I would have not been able to change formulas and URL into tokens/vectors. Plus bad input data might have led to bad output or false predictions.

Plus, there are few articles which were in different language so to make a better model we could have used multilingual models which would convert these articles into English and then we could increase the accuracy that way. But this part increases computational overhead of the project.

We can also increase accuracies by taking in more articles, right now we just used around 7900 articles but we still have thousands of other articles available to us. This would increase the computational overhead but give a better cosine score.

We also can try to use different pretrained models, since each model is trained on different dataset's and each has different vector sizes. So if we can find a pretrained model on hospital's or virus research data, or have high vector sizes we can increase the cosine similarities even though higher vector size may increase the computational overheads as well.

Another way to improve on accuracy is to use more larger model with better accuracies and finding a way to just use sentence transformation (sbert) or doc2vec to also get sentences similar form the articles. That way we could stop using word2vec model, since it has a large computational overhead then doc2vec or sentence transformation.

References

[1] https://medium.com/analytics-vidhya/theory-behind-word-embeddings-in-word2vec-858b9350870b

[2] https://cs.stanford.edu/ quocle/paragraph_vector.pdf

[3] http://web2.cs.columbia.edu/ blei/seminar/2016_discrete_data/readings/MikolovSutskeverChenCorradoDean2013.pdf

[4] https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270

[5] https://www.sbert.net/docs/training/overview.html

[6] https://en.wikipedia.org/wiki/Cosine_similarity

[7] https://blog.christianperone.com/2013/09/machine-learning-cosine-similarity-for-vector-space-models-part-iii/

Google Drive Link

This link contains whole "a1838670_code" folder, among which few files and folders I have already uploaded in the submission. Only thing missing is "Saved_models" folder which you can download using this link and make sure to keep it in "a1838670_code" folder or you can choose to download this whole folder. Yash's Google Drive Link