CORD-19 Article Clustering

1st Sanket Mehrotra
Department of Computer Science
Colorado State University
Fort Collins, USA
sanketm@colostate.edu

2nd Hwankook Lee
Department of Computer Science
Colorado State University
Fort Collins, USA
hwankook@colostate.edu

3rd Aniket Tomar
Department of Computer Science
Colorado State University
Fort Collins, USA
aniket.tomar@colostate.edu

***Abstract*—Given the CORD19 dataset of over 50000 research articles, we attempt to cluster them and narrow the given sea of literature down to a more reasonable cluster of papers using both a non-distributed and distributed fashion.**

***Index Terms*—Document Clustering, Document Analysis, Search Engine, Natural Language Processing, Clustering, Classification, Deep Learning, Topic Modelling.**

I. Introduction

Background: 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). CORD-19 is a resource of over 59,000 scholarly articles, including over 47,000 with full text(7-8GB), 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. There is a growing urgency for these approaches because of the rapid acceleration in new coronavirus literature, making it difficult for the medical research community to keep up. It is now also possible to develop text and data mining tools that can help the medical community develop answers to high priority scientific questions. [1] In this paper, we present a distributed analysis approach that covers over 50000 paper abstracts. The tool clusters the papers according to their abstracts in a scalable distributed manner using Apache Spark running on a HDFS (Apache Hadoop). We also perform LDA topic-modelling to describe the article abstracts and return the topics that best describe a paper.

II. Problem

A. Problem Definition

This problem statement was part of a Kaggle challenge released on the 3rd of April 2020 as the *"COVID-19 Open Research Dataset Challenge (CORD-19)"*. It provided a dataset of articles and papers from a variety of sources based on coronavirus and related studies.Participants in the challenge had to run data mining and information retrieval algorithms to extract useful forms of information to answer high level general questions related to the pandemic.The organizers had framed some interesting questions that we could tackle such as: *"What can we learn from this dataset about COVID-19 transmission?"* we decided to cluster the documents after exploring the structure of the dataset and noting the similarity of the abstracts of similar papers.

TABLE I
Metadata Schema with computed word counts

Column	Value
paper_id	0015023cc06b5362d332b3baf348d11567ca2fbb
title	'The RNA pseudoknots in foot-and-mouth...
abstract	The positive stranded RNA genomes...
authors	Joseph C. Ward. Lidia Lasecka-Dykes...
journal	biorXiv
doi	10.1101/2020.01.10.901801
body_text	VP3, and VP0 (which is further ...
abstract_word_count	194
body_word_count	1709
body_unique_words	704

B. Why is it interesting as a Big Data problem?

Our team noticed the large volume of documents whose full text had been provided, totaling up to a whole 8GB of text. The scraped text of this dataset is not perfect(verified) and required loads of cleaning to be able to process properly. Although there was no variety in the dataset (it was text throughout), we could see a velocity factor, since the size of the dataset doubled in the 15 days since we first saw it on the Kaggle competition website.

C. Cluster Information

We ran most of our analysis on an Apache Spark cluster of 14 worker machines. Our data was stored on a HDFS cluster of 9 machines. One of our machines: 'Nashville' went down at some point, and we had to re-upload the data on our cluster to accommodate the lost and corrupt partitions. We faced some issues in setting up the cluster and had to tinker with some of the memory settings, changing the spark.driver.memory to 15 Gb and the spark.kryoserializer.buffer.max setting to the maximum possible value of 2047 Mb. Getting the spark to work with jupyter notebooks also required some setup: exporting the PYSPARK_PYTHON, PYSPARK_DRIVER_PYTHON, and PYSPARK_DRIVER_PYTHON_OPTS system variables.

III. Methodology

A. Dataset and Cleaning

The dataset consists of papers and articles with their full text scraped from different Journals such as 'BiorXiv', 'MedrXiv',

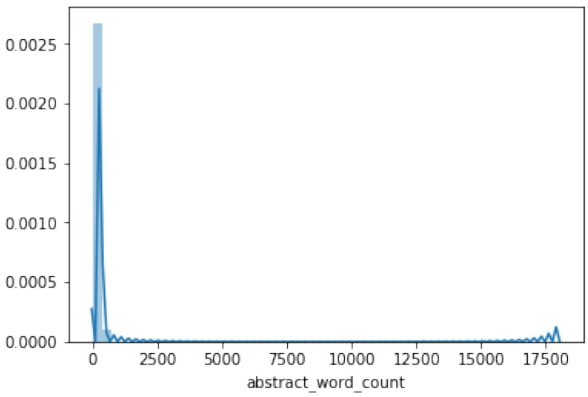

Fig. 1. Distribution of unique words in Abstracts.

TABLE II
ABSTRACT WORD COUNT STATISTIC

count	9,258
mean	221.1857
std	243.289
min	1
25%	167
50%	214
75%	269
max	18,000

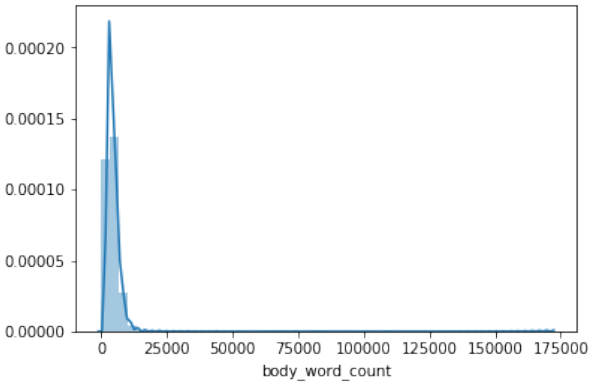

Fig. 2. Body Word count distribution.

TABLE III
BODY WORD COUNT STATISTIC

count	9,258
mean	4,276.256
std	3,443.385
min	25
25%	2,661
50%	3,824
75%	5,376.5
max	171,326

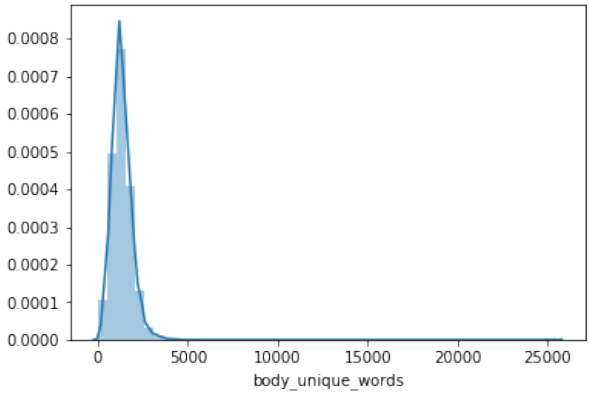

Fig. 3. Distribution of unique words in Body texts.

TABLE IV
UNIQUE BODY WORD COUNT STATISTIC

count	9,258.00
mean	1,342.41
std	684.32
min	23.00
25%	961.00
50%	1,275.00
75%	1,641.00
max	25,516.00

'PMC', 'Elsevier', 'CZI' and 'WHO'. There are several useful features present in the form of metadata such as the authors of the paper, it's title and abstract and the link to its DOI. Its references are also available. Extracting the data consisted of loading in a json file of the paper and exploring its structure for tags of interest, such as 'title', 'publishing date', and different sections of the body text.

There were several steps in cleaning the data that we performed. Firstly, we discarded attributes that were mostly null, NaN or not of particular use to us such as 'pmcid', 'pubmed_id', 'Microsoft Academic Paper ID' and 'WHO #Covidence'. We then wrote a class to represent an article, so that we could directly extract useful fields from the json structure of a file. We tried removing all documents whose titles or abstracts were null. We also dropped duplicate rows,

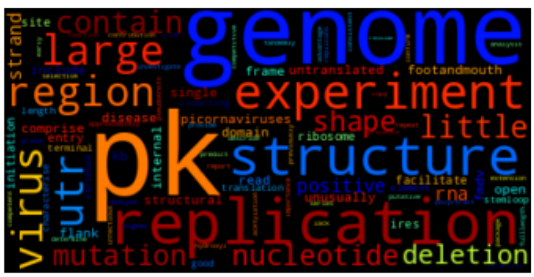

Fig. 4. Word Cloud of one abstract

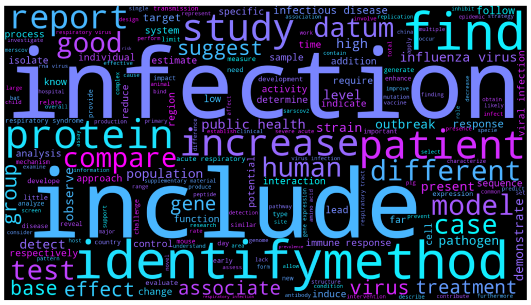

Fig. 5. Word Cloud of around 10000 abstracts

cleaned up dirty abstracts and then started the preprocessing.

B. Data Preprocessing

After the cleaning, we had to prepare the abstracts for processing. This involved mapping them to lower case, removing punctuation and more importantly, removing abstracts not in English. We used the langdetect library run over the whole cleaned set of abstracts; Its output showed a couple of papers in Italian and Taglog(graph). We removed these instead of translating them because 1. It would add unnecessary complexity to our project, and 2. There is high possibility that translation may not convey the right meaning of the whole paper. Correctly modelling the topics also required that we removed stopwords from the abstracts, otherwise their frequency may know other more relevant keywords from being counted. For this we used the spacy library and its eng_sci_bio parser built specifically for biological and clinical data. We also plotted a wordcloud of some abstracts individually, and then of all the abstracts together. Observing these allowed us to notice that the wordcloud of the whole corpus' abstracts was far too general to learn anything useful.

C. Data Statistics

While exploring the dataset, we computed two statistics that helped us understand the size of the text we were going to analyse: The word count of the abstracts(Table II) and body texts(Table 3) of each paper. Looking at the distribution charts, we understood that most of the articles had abstracts around 220 words in length and body texts in the range of 4000-4500 words in length, with some outliers being up to 17000 words in length. Looking at this figure, we felt that it was more reasonable to process the abstracts than the body texts. Larger texts will lead to too many features that will impact runtime. We also calculated the number of unique words(Table IV) present in the body, to estimate the size of the vocabulary we were dealing with.

D. Summary of Approach

1) Parse the text from the body of each document using Natural Language Processing (NLP).
2) Turn each document instance d_i into a feature vector X_i using Term Frequency Inverse Document Frequency (TF-IDF).
3) Apply Dimensionality Reduction to each feature vector X_i using t-Distributed Stochastic Neighbor Embedding (t-SNE) to cluster similar research articles in the two dimensional plane X embedding Y_1.
4) Use Principal Component Analysis (PCA) to project down the dimensions of X to a number of dimensions that will remove noise and outliers in embedding Y2.
5) Apply k-means clustering on Y_2, where k is 20, to label each cluster on Y_1.
6) Apply Topic Modeling on X using Latent Dirichlet Allocation (LDA) to discover keywords from each cluster.
7) Investigate the clusters visually on the plot, zooming down to specific articles as needed.

E. Algorithms and techniques

1) Clustering: We decided to use k-means++ clustering algorithm to find the clusters based on the similarity between document abstracts. We chose k-means++ because it was familiar to us as k-means, k-means++, and k-means|| had been covered in class. The algorithm would take inputs as vectors representing a document and its output would be the corresponding cluster that each document belonged to. This required getting a vector representation of document text for each document using a text embedding technique to pass to the k means algorithm. K-means initializes k centroid points that define a cluster and assigns each data point to a cluster based on its distance from the centroid. It then improves this clustering iteratively. Both scikit-learn and PySpark ML provide implementations of k-means algorithm. Scikit-learn's implementation has k-means++ as the default initialization while PySpark ML implements a parallelized variant of k-means++ called k-means||.This allowed the pyspark implementation to be much more scalable and faster since it was deployable across a cluster.

2) Text Embedding: There are many techniques for converting text documents to a vectors called text embeddings. We initially chose to use auto-encoders based on LSTMs as they have recently been used very successfully with NLP tasks. However, we couldn't implement an LSTM autoencoder to get document level embeddings instead of word level embeddings. So, we decided to use the widely popular TF-IDF (Term Frequency Inverse Document Frequency) embedding technique instead. In TF-IDF embeddings, for each word/term in the document given, a score is calculated by taking the frequency of that term within the document which provides a measure of the significance of the word within the document and multiplying this value with the inverse of the frequency of that term in the entire corpus which captures how informative that term is. Thus, TF-IDF score for a term captures the significance of it within that document. Both scikit-learn and PySpark ML provide implementations of TF-IDF. PySpark provides two vectorizers for Tf calculation CountVectorizer and HashingTF. CountVectorizer converts text documents to vectors of term counts while in HashingTF, a raw feature is mapped into an index (term) by applying a hash function. Then term frequencies are calculated based on the mapped

indices. This approach avoids the need to compute a global term-to-index map, which can be expensive.

3) Dimensionality Reduction: TF-IDF embeddings are extremely sensitive to occurrences of words rare in the corpus. Therefore, we limited the number of words to the 4096 most frequent words and ignored words that were found in less than 2 documents. TF-IDF gives a sparse encoding for each document with length equal to the number of unique words in the corpus which can be very large. This if passed as is to our k-means algorithm would make it much slower. So, to reduce this high dimensionality we used the PCA (Principal Component Analysis) dimensionality reduction technique to reduce the length of these input embeddings to just the number of important dimensions that capture 95% of the variance which led to the number of dimensions reducing to about half. PCA is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. Both scikit-learn and PySpark ML provide implementations of PCA.

4) Validating Clustering: To find the number of clusters best suited for clustering our dataset i.e. the k value, we used the elbow method. Here, k-means is run for each k-value in a range and the corresponding loss is plotted. We find that the loss initially decreases and then increases after reaching a minima. The k value corresponding to this point of minima is the correct k value.

5) Evaluating Clustering using silhouette method: To evaluate the performance of our k-means algorithm based on the intra-cluster cohesion and the inter-cluster separation we used the Silhouette method. PySpark ML's k-means implementation provides the evaluate() method to evaluate the clusters using silhouette method.

6) Evaluating Clustering using a classifier: If the k-means algorithm used meaningful information to cluster the documents we could implement a classifier that would be able to capture this information given the document-cluster pairs and be able to predict the cluster an unseen document belonged to. The performance of such a classifier can be used to evaluate the clustering. So, to evaluate our k-means, we implemented a FeedForward Neural Network Classifier to classify documents into their clusters and it performed well based on metrics like accuracy, precision, recall. Both scikit-learn and PySpark ML provide implementations of MLPCLassifier. We experimented with different hyperparameter values locally using scikit-learn's implementation on a quarter of the dataset and found the number these values to work best:(insert hyperparameter values). We then tried using Dask for distributing the classifier over a yarn-cluster but were facing difficulties setting up the cluster, so, we used PySpark ML's distributed implementation instead.

7) Topic Modelling: It would be helpful for researchers working on covid19 to be able to search for documents from this particular dataset for any required covid19 related topic.

To enable this, we used the topic modelling technique- LDA (Latent Dirichlet allocation) to assign top 10 topics to each

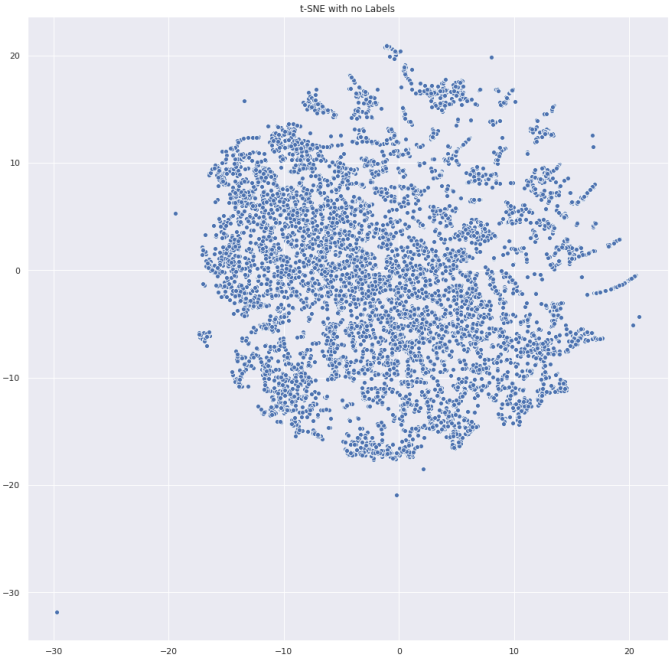

Fig. 6. Scatter plot of all the abstracts' TF-IDF representation using t-SNE

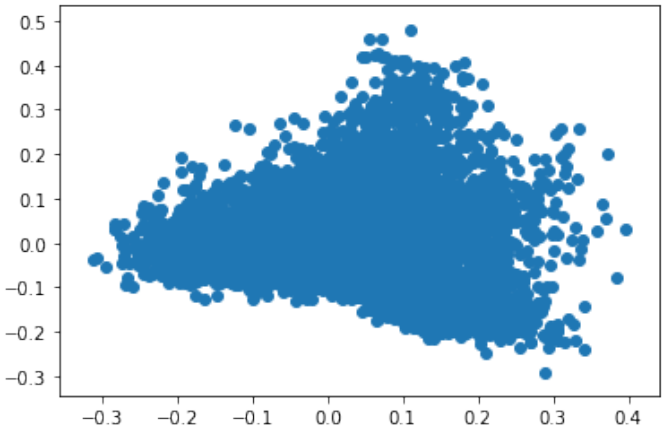

Fig. 7. Scatter plot of all the abstracts' TF-IDF representation using PCA

document and implemented a search functionality to make the documents searchable by topics. Both scikit-learn and PySpark ML provide implementations of LDA.

8) Visualization: Each cluster of documents might capture important connections between documents that are unapparent to humans. So, beyond having the ability to search for documents based on topics, researchers might like to see what other documents are similar in some sense (according to our clustering algorithm) to a particular document they searched for. To enable this, we decided to use t-SNE to reduce the dimensionality of the documents in our data to 2D and visualize them as points on a scatter-plot first without cluster labels and then with cluster labels indicated by different colors. We used t-SNE (Fig. 6) instead of PCA (Fig. 7) for visualization because unlike PCA, it reduces dimensionality

while trying to keep similar instances close and dissimilar instances apart and is therefore good for visualizing clusters of data. We find that the document data points are plotted in a close resemblance to their corresponding clusters. t-SNE is extremely sensitive to parameters such as perplexity, learning rate, number of iterations and initial stochasticity, so, we had to run it multiple times to get the desired results. PySpark ML doesn't provide a t-SNE implementation, so, we used scikit-learn's implementation locally. This was thus the most expensive computation in our project.

After this, we used Bokeh to make the plot interactive so that the researchers can search for a topic and find relevant articles belonging to a cluster indicated by a color and labelled by a topic. They can zoom into a cluster to see articles very similar to each other or zoom out of a cluster to see a high-level overview. They can also hover over data points in the plot to find details of the article which allows them to see if it is relevant and gives them easy access to the articles they need.

IV. Results and Evaluation

A. Clustering

Clustering is an unsupervised algorithm and does not have a ground truth value to evaluate its performance. Moreover, k-means requires k as an input to partition n observations into k clusters. This results in the number of clusters, but there is no proper answer to partition the data space into Voronoi cells in any problem [7]. Therefore, how can we evaluate the models are well partitioned based on different k? Here are three methods.

1) Elbow Analysis: The Sum of square distances (SSE) of each point from the centroid of the cluster suggests the proper number of clusters(k). We evaluated SSE for different values of k from 2 to 49 and chose a good k where the curve might form an elbow and flatten out. However, this method does not always give a good k value as the elbow is sometimes ambiguous, so it is not clear to find a spot where the curve starts flattening out.

To find optimal k value, we investigated k values from 2 to 49. In this graph (Fig. 8), we used Spark's computeCost function to calculate SSE. The elbow plot illustrated the point starting at 6 flatten out and forming an elbow despite the cost at 8 spiked up once. However, it was still not possible to point out the exact elbow spot because it had some fluctuations, so we looked into other metrics to find the appropriate k value. Here Silhouette Analysis can be considered.

2) Silhouette Analysis: Silhouette metric can be used to determine the degree of separation between clusters. This computes coefficients with all points in the same cluster and closest cluster, a range of this measure is [-1, 1]. The coefficients near 1 indicate that the sample is far away from the neighboring clusters. On the other hands, close to 0 indicates that the sample is close to the neighboring clusters. Lastly, a value of negative indicates that the sample is assigned to the wrong clusters. Therefore, we should find silhouette score as close to 1 as possible.

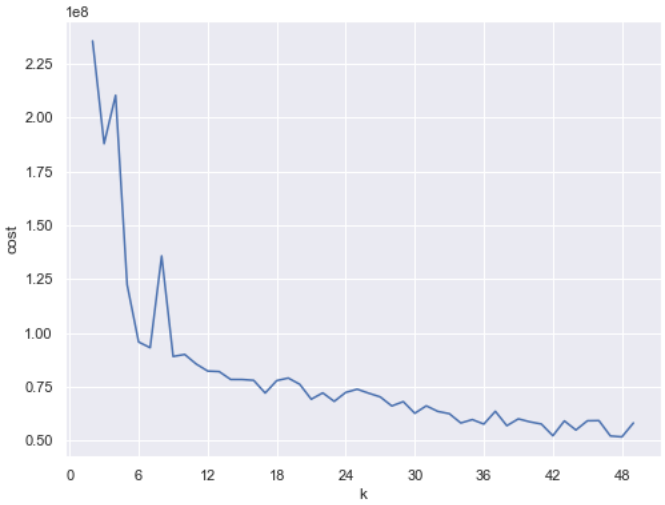

Fig. 8. Elbow method for cluster evaluation

TABLE V
Spark's Elbow Result

K	Cost
2	235513759.6185776
3	187826735.0279221
4	210305139.8575913
...	...
47	52089160.26292235
48	51680466.1513065
49	58141344.33466771

In this project, Spark and scikit-learn used to analyze silhouette metric for an optimal value of clusters. To begin with, we computed Silhouette coefficients with Spark. Tab. V shows the average of coefficients with the number of clusters by descending order. In this silhouette analysis, the number of clusters at 2 has the closest value to 1, following the number of clusters at 7. Can we choose 2 for the optimal cluster? To make sure, we further explored silhouette analysis with scikit-learn. The Fig. 9 of the right cluster at the cluster of 2, it seemed plausible, yet the score was 0.023285... (Tab.VII). However, Scikit-learn's results had been increased by the number of cluster at 47 after at 2 (Fig. 10), unlike the previous result of Spark, and decreased by at the end. The cluster of 2 was reasonable at the first time, but now 47 was more credible.

TABLE VI
Spark's Silhouette Result

K	Silhouette Score
2	0.97007909
7	0.66487722
3	0.63605217
...	...
44	-0.06245410
38	-0.07805334
35	-0.08465259

TABLE VII
SCIKIT'S SILHOUETTE RESULT

K	Silhouette Score
2	0.02328579064453245
3	0.01897346866166880
4	0.01831956972521928
...	...
47	0.04457166654170685
48	0.04156933429654094
49	0.04394185708606362

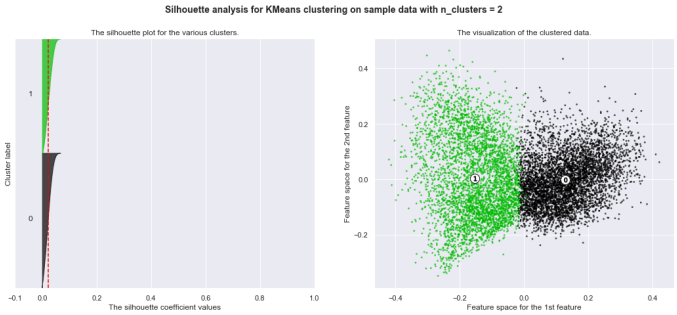

Fig. 9. Silhouette 1.

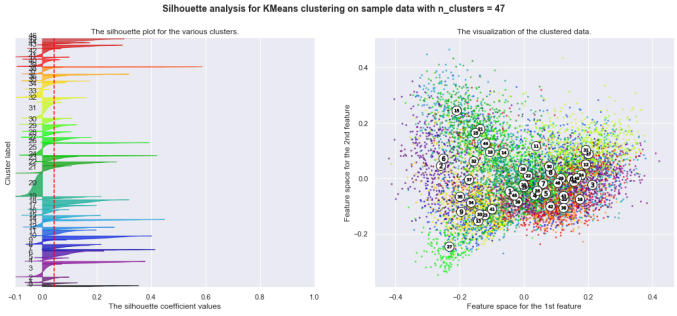

Fig. 10. Silhouette 2.

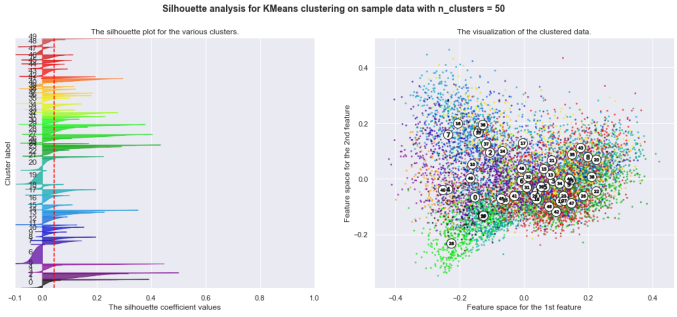

Fig. 11. Silhouette 3.

3) MLPClassifier: We ran scikit-learn's MLPClassifier on a quarter of the dataset and experimented with various hyperparameters like the optimizer, hidden layer configurations, number of iterations before distributing the computation for the best classifier. We found that the classifier converges faster using the optimizer 'lbfgs' than 'adam'. We experimented with different number of neurons in the hidden layers and also the number of hidden layers. We got best results when there were a large number of neurons in each individual hidden layer but the number of hidden layers were low. We experimented with the number of neurons in each hidden layer using the following numbers (40, 80, 100, 200, 400) and with the number of hidden layers ranging from 1 to 5. We took 60% of the data as our training set and 40% as the test set. We used scikit-learn's cross validation with 5 fold validation to evaluate the hyperparameters. Initially, we got the best results on the test set for 2 hidden layers with 400 and 100 neurons. The accuracy, precision, recall were all over 90% which would indicate that the clustering was good. Surprisingly, the results got worse with an increase in the number of hidden layers. However, rerunning the k-means algorithm for a different number of k as suggested by other evaluation metrics and using its results as the new data for classification using the same hyperparameters gave us bad results with an accuracy of 53% which should indicate a bad clustering. So, here again it would appear that there is a contradiction between our metrics for cluster evaluation. However, this may be because some clusters had too few members for the classifier to be able to train well.

Fig. 12. K-means with k=20 run on the t-SNE representation in Fig. 6

B. t-SNE

Evaluating the plots resulting from t-SNE is crucial because t-SNE is very sensitive to a number of its parameters such

as perplexity and learning rate. Especially for scikit-learn's implementation the value of learning rate is crucial. If the learning rate is too high, the data may look like a 'ball' with any point approximately equidistant from its nearest neighbors. If the learning rate is too low, most points may look compressed in a dense cloud with few outliers. So, we tried the following values of learning rate (10, 50, 100, 500, 1000) and picked the best one:600.

The t-SNE plot keeps changing with every iteration, so, to choose an appropriate value for the number of iterations we ran t-SNE until the plot stabilized and chose the corresponding value for the number of iterations.

The Perplexity parameter indicates the number of neighbouring data points whose distance t-SNE tries to preserve. Different perplexity values can give very different results and an inappropriate perplexity value can give different results in different rins. To decide the perplexity value, we ran t-SNE multiple times for each of the perplexity values (2, 5, 10, 20, 50, 100) and picked the one that gave similar results in each run.

We managed to get some good resulting plots like Fig. 12.

V. Contributions

Team Member	Contribution
Sanket M.	Data cleaning and preprocessing, Statistic analysis, LDA, Merging, Report
Hwankook L.	PCA, t-SNE, Clustering, Clustering Evaluation, Report
Aniket T.	Classifier, Classifier Evaluation, Report

References

[1] COVID-19 Open Research Dataset Challenge (CORD-19) [Online]. Available: kaggle, https://www.kaggle.com/allen-institute-for-ai/CORD-19-research-challenge/.

[2] W. Buntine et al., "A Scalable Topic-Based Open Source Search Engine", IEEE/WIC/ACM International Conference on Web Intelligence (WI'04), Beijing, China, 2004, pp. 228-234.

[3] C. Grant et al. "A topic-based search, visualization, and exploration system." The Twenty-Eighth International Flairs Conference. 2015.

[4] How to build a topic-based search engine. [Online]. Available: https://www.smithinst.co.uk/insights/build-topic-based-search-engine/

[5] Semantic Topic Modeling for Search Queries at Google. [Online]. Available: https://gofishdigital.com/semantic-topic-modeling/.

[6] The COVID-19 Literature Clustering project. [Online]. Available: github, https://github.com/MaksimEkin/COVID19-Literature-Clustering.

[7] K-means Clustering: Algorithm, Applications, Evaluation Methods, and Drawbacks [Online]. Available: https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a.

[8] Spark Document (https://spark.apache.org/docs/latest/index.html).

[9] Selecting the number of clusters with silhouette analysis on KMeans clustering. [Online]. Available: https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html.