BAN 675

Text Mining Project

BREATHE Biomedical Literature Dataset

Team 4:
Dalton Williams
Prathamesh Rajiv Bhople
Subramanian Tirunelveli Padmanabhan

Contents

I. INTRODUCTION

1.1. Dataset Description

"BREATHE is a large-scale biomedical database containing entries from 10 major repositories of biomedical research. Our dataset contains both abstract and full body texts of biomedical papers going back for decades and contains more than 16 million unique papers. This dataset can be used to train language models to better understand outcomes from biomedical research and uncover insights to combat the COVID-19 pandemic." – Google Cloud Platform, BREATHE Description

Included in the ten tables in this database—one for each repository—are the publication dates, abstracts, authors, and full-text of each article provided. Google maintains a weekly expected update frequency for this publicly available database.

1.2. Objective

The objective of a project is to develop topic modeling based on the abstracts of biomedical papers, predict topic distributions, create a similar article recommender, evaluate topic concentration, and compare different variations of the models.

1.3. Techniques & Libraries Used

The NLTK, Gensim, Wordcloud and Matplotlib libraries are used extensively in the project. The packages including Corpus, Tokenize, Stem, Corpora etc. help in cleaning up the dataset, creating models and graphs, and predicting topics.

1.4. Results

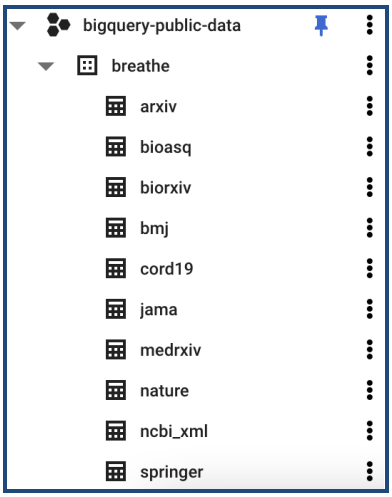
The results obtained from the project helped in understanding:

- How the topics are modeled
- The usage of relevant words in an abstract
- The metrics, such as coherence values and perplexity score, that help to predict topics
- What paves the way to recommend similar article

II. DATA EXTRACTION

2.1. SQL Query

```
EXPORT DATA OPTIONS(
  uri='gs://ntlk-project-healthcare-csu/[tablename]*.csv',
  format='CSV',
  overwrite=true,
  header=true) AS(
SELECT
   *
FROM
   `bigquery-public-data.breathe.[insert table name]`
WHERE
   LENGTH(TRIM(date))>9
   AND LENGTH(TRIM(date))<11 --Prevents mis-entered dates from
   breaking timestamp function
   AND TIMESTAMP(TRIM(date)) > TIMESTAMP('2018-07-01'));
```



2.2. Random Set of Abstracts

The SQL queries for each of the 10 tables exported .CSV files to a Google cloud storage bucket, which were then bulk downloaded. The resultant download was around 20 Gigabytes of data, so an initial pre-processing step was to write a python script to iterate through each file and retrieve rows based on a random uniform distribution. The export's CSV splits were approximately equivalent in size, based on GCP's default exporting settings, so the python code loops through each CSV file from the directory and selects an equivalent number of rows from each file. The number of rows-per-file selected is based on dividing the target number—in this case 1000 rows—by the number of files, and rounding that number up so that the target is the minimum number of retrieved rows. Then, when iterating through each file in the directory, the rows are chosen by a uniform random number with a minimum of the first-row index and maximum of the last row index in each file, checked to ensure it is not a duplicate number. The row with an index matching this uniform random number is added to a dataframe. Once this dataframe is fully constructed by iterating through all these files, it is saved as a new CSV for use in the final model.

III. DATA PRE-PROCESSING

Topic models are statistical models that help to uncover hidden composition in a collection of texts. This project was executed by extracting a huge amount of data from medical journals. Using Python programming, a random set of abstracts i.e., 1000 or more, in total, was obtained to perform the analysis.

3.1. Data Cleaning

To perform a clear analysis, it is essential to clean up the unstructured data using different methods, helping us to increase the accuracy of the prediction, and to reach reasonable inferences.

- Filtering out the stopwords is essential as it removes commonly used words that do not add much information to the text. Using the corpus package, the list of stop words is removed by checking that list against words within the text.
- The words in the text are obtained using the word_tokenize method from the Tokenize package.
- Words in the text can be derived from one another and these inflected words need to be reduced to their root forms. These words may not be a valid word but assist in achieving a more precise understanding of the text. Using the PorterStemmer method from the Stem package, these word stems are created.

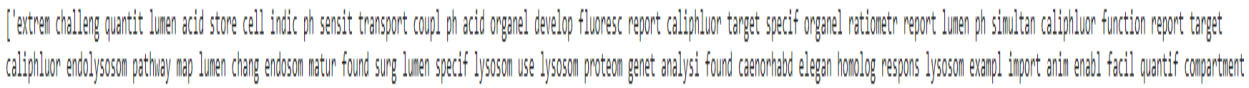


FIGURE 1. Corpus with sentences after data cleaning.

- URLs are the address locators for online resources, but are unnecessary while performing any text analysis. They are removed by using a regular expression (re.sub ()) that replaces URLs with nothing (‘’).

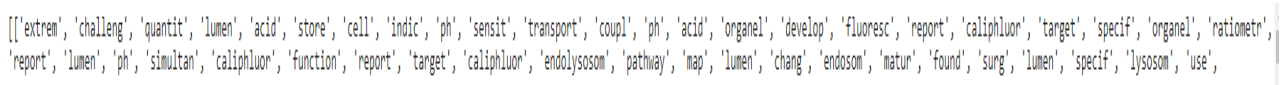


FIGURE 2. Corpus with Tokenized words after data cleaning.

- Characters like full stop and comma do not convey any information that impacts the model building. These characters are removed by checking a binary method.
- A few of the least common words from each document are removed because they may not find a match in other documents and may help produce quicker model runs on a dataset containing many documents.

3.2. Corpora Dictionary

A corpus contains each word’s token’s id, and to work on text documents, Gensim requires the words, i.e., tokens to be converted to their unique ids. Here the list of words is passed to the corpora.Dictionary() object where each word is mapped to its unique integer id.

```
3851 metastas
3852 metastat
3853 micrometastas
3854 neutrophil
3855 surround
3856 xenotransplant
3857 zebrafish
3858 bma
3859 chair
3860 hallett
3861 junior
3862 sarah
```

FIGURE 3. Corpus Dictionary with Index for each unique word

3.3. Document Term Matrix

The tokenized list of words is sent to a bag of words object (Dictionary.doc2bow()) and it results in frequency of each word, thus creating a Document Term Matrix for the complete bag of words

```
[[(0, 3), (1, 1), (2, 2), (3, 1), (4, 5), (5, 1), (6, 1), (7, 1), (8, 1), (9, 1), (10, 1), (11, 1), (12, 2), (13, 1), (14, 1), (15, 1), (16, 1), (17, 1), (18, 1), (19, 2), (20, 2), (21, 1), (22, 1), (23, 1), (24, 3), (25, 1), (26,
4), (27, 3), (28, 1), (29, 1), (30, 3), (31, 1), (32, 4), (33, 1), (34, 1), (35, 2), (36, 2), (37, 4), (38, 1), (39, 1), (40, 1), (41, 2), (42, 1), (43, 1), (44, 1), (45, 2), (46, 1), (47, 1), (48, 2)], [(21, 1), (38, 1), (48, 1),
```

FIGURE 4. Document Term matrix with frequency of a word

IV. LDA Model

Latent Dirichlet Allocation(LDA) is a popular algorithm for topic modeling with excellent implementations in the Python's Gensim package The LDA module is very scalable, robust, and optimized in terms of performance, but its primary LDAModel function only runs on a single thread. Because in recent years CPUs with multiple cores have become standard, the project has a multicore implementation of LDAMulticore that provides a considerable time reduction, especially for very large corpora as in this case. Using all CPU cores to parallelize and speed up model training, the model performance and response time was key to extracting information from these huge texts. We observed a decrease in runtime for the model generating cell down to approximately 25 minutes, from an initial run of over 55 minutes.

V. TOPIC COHERENCE AND PERPLEXITY

The optimum number of topics in text is a challenging task to figure out and Topic Coherence measurement is a widely used metric to evaluate topic models. It uses the latent variable models. Each generated topic has a list of words. The high value of topic coherence score model will be

considered as a good topic model. Using the average/median of pairwise word similarity scores of the words in a topic will help identify optimum number of topics.

```
[0.4930712015438484, 0.4870525695631046, 0.5086349888717144, 0.4892630408615828, 0.45394855207226115, 0.4767163663685559, 0.5070450508084525, 0.5315250999048419]
-------
-7.738539722076859
-------
```

FIGURE 5. Coherence & Perplexity Score To find Optimized Topic

The number of passes and batch size are used to run model several time between start and stop value at each step. At each step, the coherence score is calculated to give the value for each topic. The topic with highest score would then be used in the model to find the topic distribution.

5.1. Coherence Graph

The coherence graph is a plot of coherence score against number of topics, and this visualization will show us the peak score of a particular number of topics. That peak topic number will be the total number of topics that we can use in building an efficient topic model.

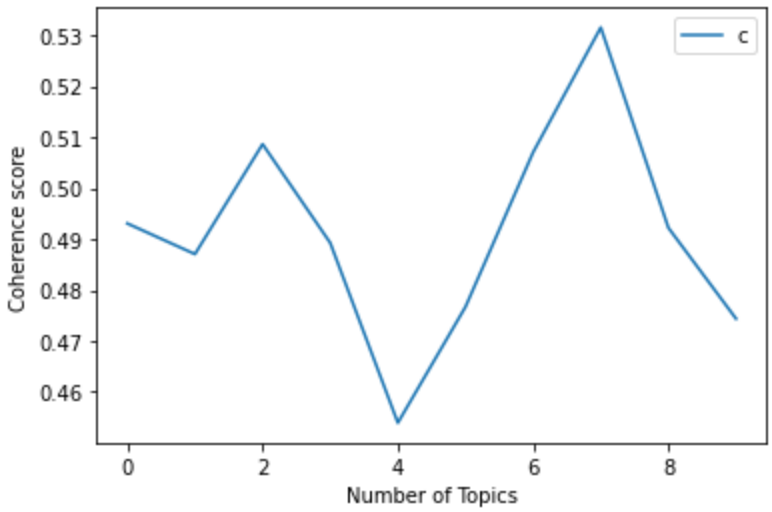


FIGURE 6. Coherence Graph to Find Optimized Topic

5.2. Perplexity Score

The perplexity score is measure of how a model predicts a sample; this is achieved through log_perplexity method of the model where the document term matrix is passed as input. The perplexity score with a low value is the preferred measure to decide to the topics.

```
-------
-7.7446864989181075
-------
```

FIGURE 7. Sample Perplexity score

VI. MODEL EXECUTION AND RESULTS

To evaluate and interpret the model execution the values of a topic printed alongside the extracted weight and word. Here, the most significant 3 topics, and the most important 3 words can be retrieved from the model function.

```
--------------------------------------------------------------------------------
Most Freq. Topics, Most Rep. Words

[(2, '0.020*"ci" + 0.017*"studi" + 0.015*"patient"'), (5, '0.015*"infect" + 0.015*"model" + 0.014*"diseas"'), (0, '0.049*"patient" + 0.015*"sever" + 0.012*"diseas"')]
--------------------------------------------------------------------------------
```

FIGURE 8. Most Frequent Topic & their words

We can also view all the topics and words related to the topics:

```
[(0, '0.049*"patient" + 0.015*"sever" + 0.012*"diseas" + 0.011*"acut" + 0.011*"mortal" + 0.010*"age" + 0.009*"clinic"
```

FIGURE 9. Words Related to topic0

To view only the specific topic and its word with weights:

```
0.049*"patient" + 0.015*"sever" + 0.012*"diseas" + 0.011*"acut" + 0.011*"mortal" + 0.010*"age" + 0.009*"clinic"
```

FIGURE 10. Worsds & Weights for the topic0

All weights of words in a topic add up to a probability close to 1, as shown below:

```
[[1.6633441e-05 1.6665968e-05 1.6634283e-05 ... 1.6629197e-05
  1.6608668e-05 1.6603741e-05]

                    Word Prob.

                    0.9999999938590918
```

FIGURE 11. Word probability

Taking this data into perspective, we used the get_document_topics function in the model to display the highest weighted topics for a particular document. This document can be identified by passing the document term matrix object which has a list of words associated with document and its frequency.

```
Topic Distribution

[(2, 0.48753837), (3, 0.11386986), (4, 0.24616946), (5, 0.14705198)]
```

FIGURE 12. Topic Distribution list

6.1. Word Cloud Comparison

The actual LDA model built for the prediction used the optimized topic value that was inherited using the coherence score. To have a baseline model to compare with to get a better idea of the word behavior, we have created an Arbitrary model, based on the default parameters shown in class, with topics size as 10 as opposed the LDA multicore model with predicted topic value of 8. The Wordclouds below show the 15 most representative words in the top 8 topics in both models.

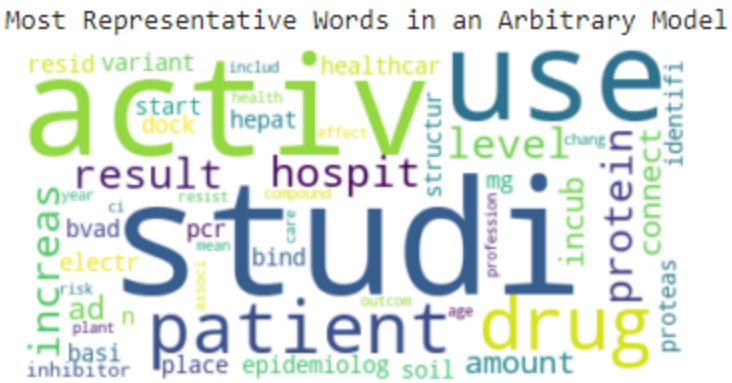


FIGURE 13. Word Cloud for Arbitrary Model

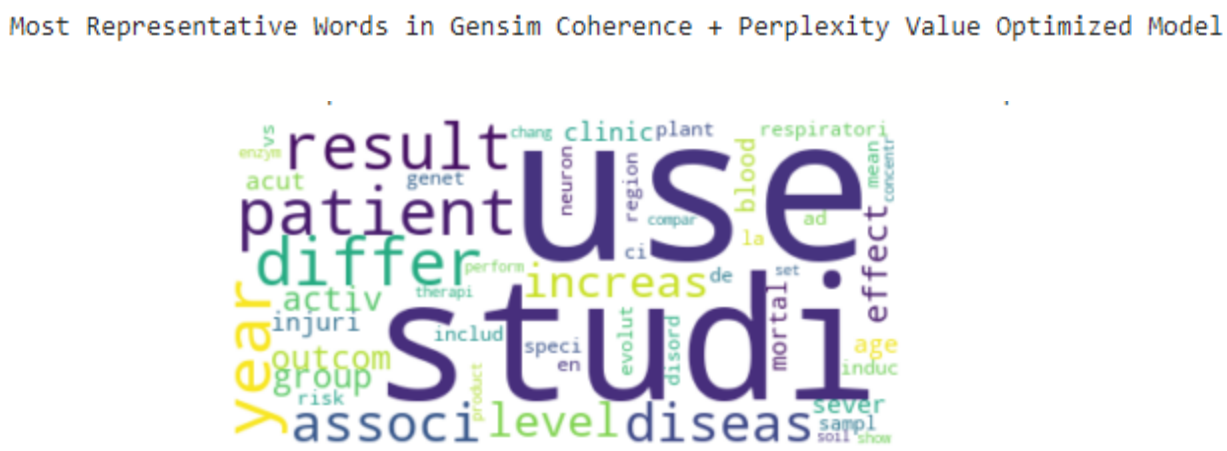


FIGURE 14. Word Cloud for Multicore LDA Model

VII. SIMILAR ARTICLE RECOMMENDER

Once the topics are modeled, we used a metric called Cosine Similarity that determines the similarity of the data objects in a dataset. These data objects are treated as a vector. From the Topic Distribution list, the values of the weight of words are sent as a list of vectors to the Cosine Similarity module which in turn performs the calculation.

With the above values calculated, we then check for the cosine similarity value greater than 0.6 (i.e., a threshold value set within the code to confirm that documents are more than 60% similar and are a better representation of each other). If probability is greater than 0.6, we are then indicating the Topics that are similar to each other, which helps to more accurately interpret which documents are most closely related to each other.

```
def cossimilarity(x,y):
    value1=np.dot(x,y)
    xm=np.sqrt(sum([i**2 for i in x]))
    ym=np.sqrt(sum([i**2 for i in y]))
    value2=xm*ym
    cs=value1/value2
    return cs
```

FIGURE 15. Cosine Similarity Function I Python

7.1. Top Articles with Similarity

With the Cosine Similarity recorded, the list of articles needs to be analyzed to determine the top 5 articles containing the most similar features with other documents, and which topic provided the highest number of similar documents.

- A dictionary was created with an article's details and its similarity count

```
The top 5 articles with highest similarity with another article are  {32: 490, 134: 450, 118: 441, 74: 437, 40: 423}
```

FIGURE 16. Top 5 Articled Similarity Detail

- Another dictionary was created with article and article numbers with which it was most similar
- Articles 32 & 33 were similar and had topic5 as their respective highest weights

```
Article 32 with highest similaritty with other documents: 33 & 242.This to check the impact of topics and words
Article 32's topic distribution:
 {0: 0, 1: 0.11283641, 2: 0.023115102, 3: 0, 4: 0.33451927, 5: 0.42909393, 6: 0.09829514, 7: 0, 8: 0, 9: 0}


Article 33's topic distribution:
 {0: 0, 1: 0, 2: 0, 3: 0.31435052, 4: 0, 5: 0.41714564, 6: 0.24245521, 7: 0, 8: 0, 9: 0}
```

FIGURE 17. Topic weight for Document 32 & 33

- Articles 134 & 705 were similar and had topic 4 as their respective highest weights

```
Article 134 with highest similaritty with other documents: 705 & 146.This to check the impact of topics and words
Article 134's topic distribution:
 {0: 0.030058462, 1: 0.013626705, 2: 0.062027432, 3: 0, 4: 0.43469718, 5: 0.3961067, 6: 0.06289439, 7: 0, 8: 0, 9: 0}


Article 705's topic distribution:
 {0: 0, 1: 0.20519675, 2: 0, 3: 0.0929156, 4: 0.67940974, 5: 0, 6: 0.018414844, 7: 0, 8: 0, 9: 0}
```

FIGURE 18. Topic weight for Document 134 & 705

7.2. Similarity Graph

Having achieved the implementation of the design to get article similarity detail, the top 5 articles with the highest count of their similarities with other articles was plotted to infer more on the article recommendation.

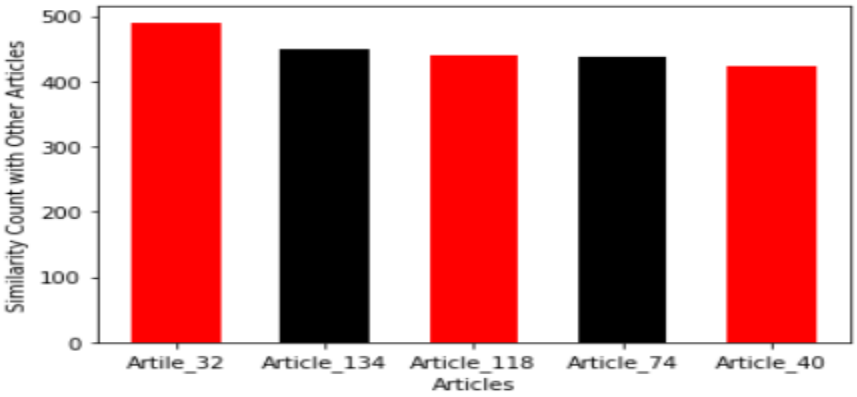


FIGURE 19. Similarity Count For Articles

7.3. Topic Labeling

The model has the method print topics with which we can retrieve the words associated with the topic for a particular article. For this we need to pass the article index to this function. Using the most representative words we can predict a meaningful label manually and associate with similar articles. Topics 4 and 5 had words which were both related among their respective lists.

```
Topic 4 details:
 0.025*"cell" + 0.012*"protein" + 0.011*"express" + 0.010*"gene" + 0.008*"activ" + 0.006*"result" + 0.006*"studi" + 0.006*"cancer" + 0.006*"use" + 0.006*"respons"


Topic 5 details:
 0.015*"infect" + 0.015*"model" + 0.014*"diseas" + 0.013*"case" + 0.011*"coronaviru" + 0.010*"data" + 0.009*"number" + 0.009*"use" + 0.009*"spread" + 0.008*"countri"
```

FIGURE 20. Weight of Words in topic4 & topic5

As Topic 134 had topic4 with highest weight, we can label the topic as “Cell Protein Study”.

As Topic 32 had topic5 with highest weight, we can label the topic as “Coronavirus Infection”

7.4. Arbitrary Abstract Testing

Following the creation of the topic distribution list for the sample data set, an arbitrary abstract which is not part of the sample data was chosen to test the article recommendation feature. The purpose is to recommend similar articles from the 1000+ abstracts in the sample set based on this new arbitrary article. Therefore, after reading and cleansing this new article, we prepare a clean corpus. For this new clean corpus, we generate the word dictionary, doc term matrix and topic distribution list only for this article. Now we have two topic distribution lists, one for all the articles we trained and other for this new article. We compared topic distribution list of this new article with each individual topic distribution list of all the articles and calculated cosine similarity. If the cosine similarity is greater than or equal to 60%,

we can conclude that the new article and an article from dataset are similar, and therefore, we will be recommending that article. If cosine similarity is less than 60%, we would not be recommending that article. For example, as shown in FIGURE 21, article 5 and article 6 from the dataset have cosine similarity greater than 60% when compared with new article, therefore these articles are recommended. On the other hand, article 4 and article 7 are not recommended.

```
--------------------------------------
Topic similarity between New article and article 4
Not recommended
--------------------------------------
Topic similarity between New article and article 5
0.9563822972779038
--------------------------------------
Topic similarity between New article and article 6
0.9416221149503052
--------------------------------------
Topic similarity between New article and article 7
Not recommended
--------------------------------------
```

FIGURE 21. Similarity Count For Articles

VIII. AUTHORS AND AFFILIATIONS

8.1. Authors: Dalton Williams, Prathamesh Rajiv Bhople, Subramanian Padmanabhan

College of Business & Economics

California State University, East Bay

Email: **pbhople@horizon.csueastbay.edu, dwilliams129@horizon.csueastbay.edu spadmanabhan3@horizon.csueastbay.edu**

IX. EQUATIONS

Similarity measure refers to distance with dimensions representing features of the data object. In the project we used cosine similarity, shown by the following equation:

(1). $\cos cos\,(x, y) = \frac{(x \bullet y)}{||x||\ ||y||}$

where,

- $(x \bullet y)$ = product (dot) of the vectors ‘x’ and ‘y’.
- $||x||\ ||y||$ = length of the two vectors ‘x’ and ‘y’.
- $||x||\ ||y||$ = cross product of the two vectors ‘x’ and ‘y’.

Acknowledgment

We would like to express our sincere thanks to our Professor Peng Xie for sharing his knowledge with his students and developing our skills in this subject. We would also like to acknowledge Google as a valuable host of an immense amount of frequently updated information, which they provide to the general public at no charge; without their efforts in providing public access to the Google Cloud Platform, this project would not have been possible. Furthermore, we would like to thank all the people who guided us and gave us an opportunity to present this idea as a project that could lead to more research.

X. REFERENCES

1. Electronic Media

Aravind CR., "Topic Modeling using Gensim-LDA in Python," *Analytics Vidhya*, July 26, 2020,

https://medium.com/analytics-vidhya/topic-modeling-using-gensim-lda-in-python-48eaa2344920

Google. "BREATHE Biomedical Literature Dataset." *Google Cloud Platform*, Google,

https://console.cloud.google.com/marketplace/product/breathe-gcp-public-data/breathe?project=wired-column-330101 .

Navlani, A., "Latent Semantic Analysis using Python," *Data Camp*, October 9, 2018,

https://www.datacamp.com/community/tutorials/discovering-hidden-topics-python

Rao, S., "Cosine Similarity," geeksforgeeks, October 6, 2020,

https://www.geeksforgeeks.org/cosine-similarity/