## **Covid-19 Question-Answering System**

**Team:** Shuyan Yu, Zhengjia Mao, Xiaoyu Liu **Project Mentor TA:** Sihao Chen

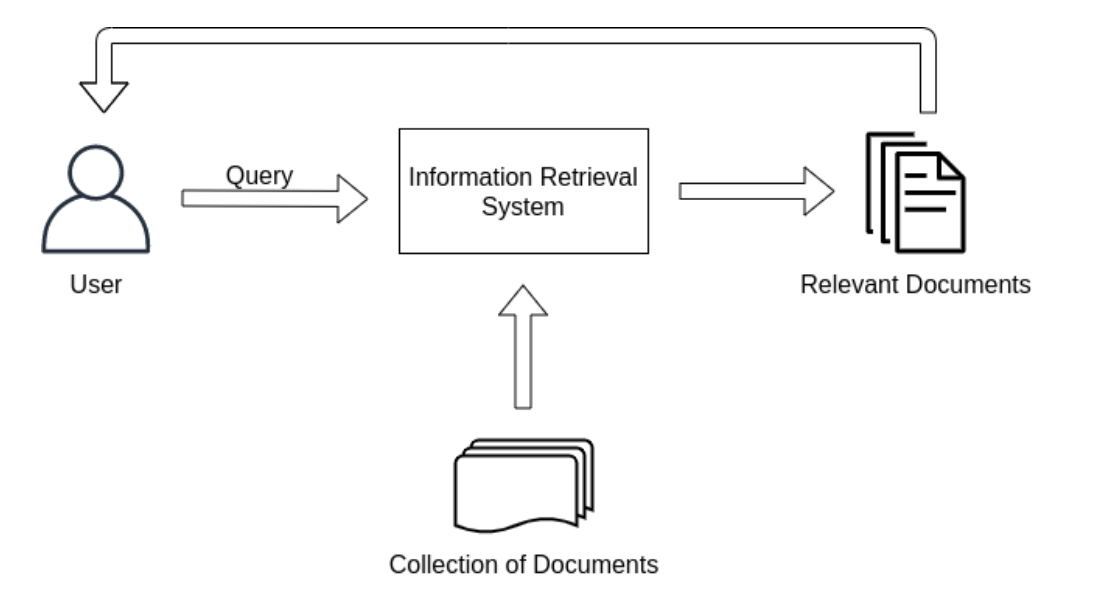
#### **1. Abstract**

In this project, we dived into one of the hottest Natural Language Processing (NLP) topics, a document ranking task specifically, and built a Covid question-answering (QA) system based on the Retrieve-and-Rerank framework. This task is important for efficiently extracting relevant information from a large amount of documents. Our primary target contribution is to optimize traditional QA systems that generally rely on a single NLP model and require expensive computing resources. We, instead, experimented document retrieval models, TF-IDF and BM25, with different tokenization approaches to narrow down the ranges of target documents and then utilized strong learner BERT to rerank the retrieved data to improve relevantivity of the recommended documents. This optimized QA system can be simple while effective even with limited computational resources. Given the untokenized ranking results by TF-IDF as a benchmark, we found that the combination of BM25 and BERT performed fairly well in recommending documents relevant to a given query.

#### **2. Introduction**

When a large amount of documents is presented, researchers and scientists usually get discombobulated searching for relevant information on a specific topic. Our project aims to make this task easier with less time and higher precision. As the user inputs a natural language query, our system is expected to recommend a list of documents that are semantically related to the query in the descending order of relevance.

Our QA system was established based on the open dataset published on Kaggle’s COVID-19 Open Research Dataset Challenge. The original dataset consisted of 335k JSON files totaling 33.91 GB. Each file included a paper ID, a title, an abstract, body texts, bibliography entities, reference entities, and back matters. Due to computational limitations on Google Colab, we shuffled and drew 10000 random samples from the database for training. The project pipeline involved two stages: retrieving and reranking. In the retrieving phase, TF-IDF and BM25 were implemented to return a list of 500 files relevant to the given query. And then during the reranking phase, the retrieved files were passed to BERT which returns the top k files based on reranking scores (here k is less than or equal to 10). Note that all the models were trained on title, abstract, and body text sections.



*Figure 1: Question-Answering System Diagram*

**3. Background**

**A**. Ubiquitous Knowledge Processing Lab, “Retrieve & Re-Rank”, 2021, GitHub. Retrieved 4/1/2022 from <https://github.com/UKPLab/sentence-transformers/tree/master/examples/applications/retrieve_rerank>

This reference guided us through the idea of retrieve & re-rank and how to construct a BM25 model. We built our BM25 retrieving part with inspiration from this code and modified our model to fit the CORD-19 dataset.

**B**. Chris McCormick, “How To Build Your Own Question Answering System”, 2021, McCormickML. Retrieved 4/1/2022 from ​​<https://mccormickml.com/2021/05/27/question-answering-system-tf-idf/>

This reference constructed a BERT model for a different dataset based on a library that we are able to reproduce. We built our BERT reranking part with reference to this code.

**4. Summary of Our Contributions**

**Contribution(s) in Code:** Basically, we customized the entire project by ourselves with the adoption of the “retrieve & rerank” framework from PyGaggle [Castorini, 2021]. We first implemented the data preprocessing procedure by multiple tokenization approaches, then built a TF-IDF embedding from scratch to narrow down the range of relevant documents. We also modified the BM25 model to make it compatible with the CORD-19 context. On top of that, we applied BERT with tuned hyperparameters to rerank the retrieved articles.

**Contribution(s) in Algorithm:** Our main contribution in algorithm was to manually construct a TF-IDF and a BM25 retrieval model from scratch. As the vectorizer in Scikit-learn does not allow customization in terms of how to separate punctuations, we defined several rules for the algorithm better handling the CORD-19 text data.

**Contribution(s) in Analysis:** In addition to the traditional evaluation metrics such as recall, precision etc., we measured the macro-averaged performance of the reranked list of documents regarding each training query with the intention of treating all classes equally.

**5. Detailed Description of Contributions**

**1) Contributions in Code:**

Initially, we proposed to utilize the neural ranking algorithm from PyGaggle as a baseline for our CovidQA. Nonetheless due to package dependency issues and the limitation of computational resources, we failed to reproduce the results from PyGaggle. Thus instead, we adopted its underlying “retrieve & rerank'' idea and constructed the whole QA framework by ourselves. As the training set is stored in JSON format, we first created a data preprocessing pipeline to load and separate the text body in terms of queries and potential lists of relevant articles. The following preprocessing steps involved stopwords removal and documents indexing. Our initial experiment was implemented on the preprocessed corpus without tokenization, which was identified as a benchmark for our ranking models. Considering the existing vectorizers from Scikit-learn do not allow customization of the ways to handle different types of punctuations, we ended up designing our own tokenizers and document ranking models from scratch. The retrieving process mainly relied on TF-IDF and BM25.

The first tokenization approach consisted of removing common English stopwords, removing standard generalized markup language, regularizing date/time data, handling acronyms, abbreviations, numbers, possessive nouns, hyphenated phrases. On the contrary, the second method was hard-coded based on observations. On top of these two approaches, we manually computed term frequencies and document frequencies, then built a TF-IDF model, outputting scores that indicate the relavantivity of each article and the given query. Furthermore, with the intention of experimenting on different retrieval methods, we repeated the embedding process by the BM25 framework from scratch. Upon the computation of document frequency and index frequency, we parsed the corpus and queries, then calculated the BM25 scores to rank each document for the first time.

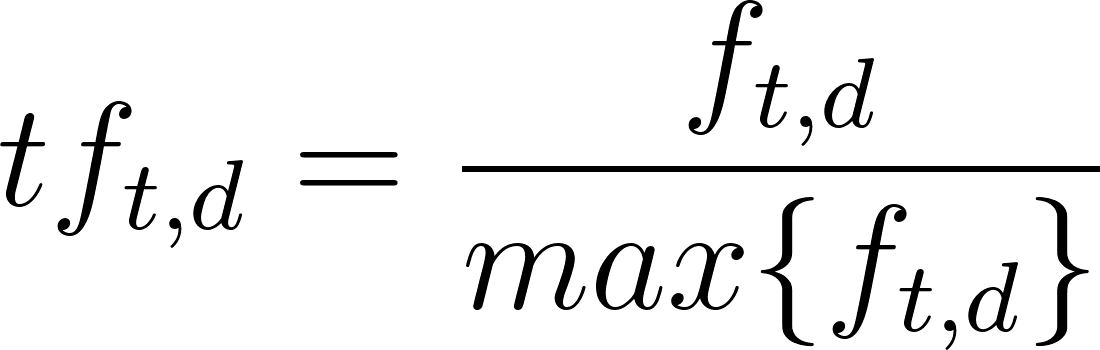
**2) Contributions in Algorithm:**

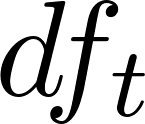
One of the most important steps before NLP modeling tasks is to tokenize data. Considering that text data is usually skewed and not well-organized, we needed to standardize both input documents and queries to make sure there was no mismatch between words that refered to the same thing. In hopes of designing a specific preprocessing solution for the CORD-19 data context and going above and beyond the tokenization approaches in existing libraries such as NLTK, we set up a basic preprocessing pipeline to manipulate the training documents from scratch mainly using regular expressions. For each file, we extracted title, abstract, and body text sections from the JSON data source and stored them as a list of sentences. The corpus list was then fed into a customized tokenizer which separated punctuations among words and normalized the cleaned corpus. In respect of punctuation, we detailed multiple edge cases in natural language and created some principles as following:

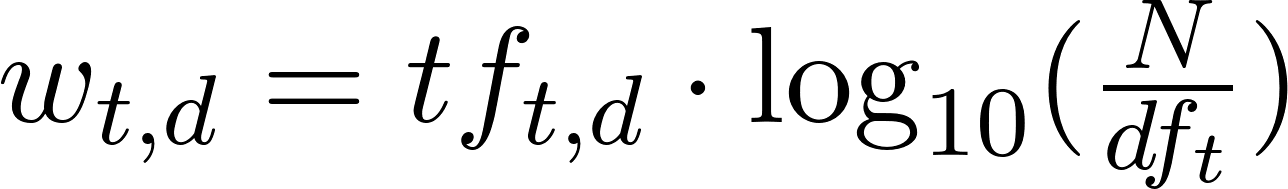
| **.** | Acronyms, abbreviations, numbers are not tokenized |
| --- | --- |
| **‘** | Expand while needed (e.g. can’t -> can not, Monday’s -> Monday ‘s) |
| **-** | Group the phrases separated by - together |
| **,** | Numbers are not tokenized |

For normalization, we integrated different formats of the same word (e.g. jan, Jan. → January; de-accent words such as resume) and unified the date/time data to ensure them to be displayed in a consistent manner.

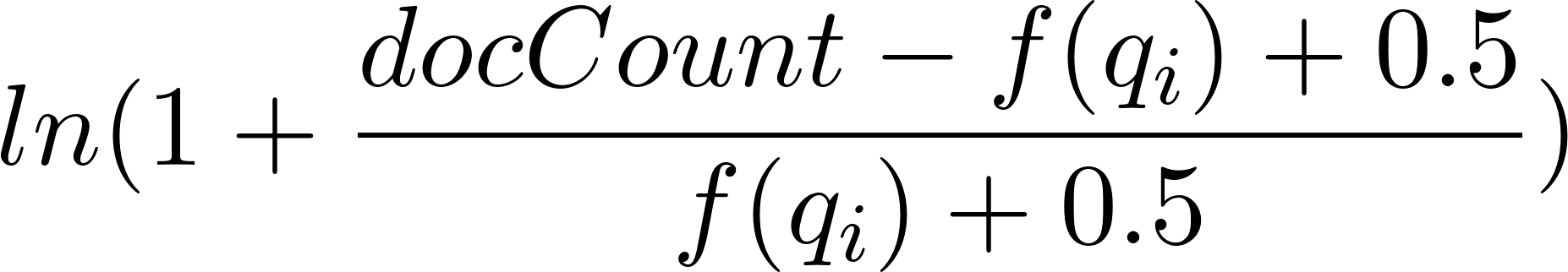
In addition, we built a TF-IDF model from scratch and performed the embedding on a sample set of 10,099 files from CORD-19. This TF-IDF solution involved tokenization described above as well as the process of constructing Vector Space Models through TF-IDF weighting scheme. We first computed the inverted index dictionary of all documents’ tokens using term frequency as shown in Eq. (1):

 (1)

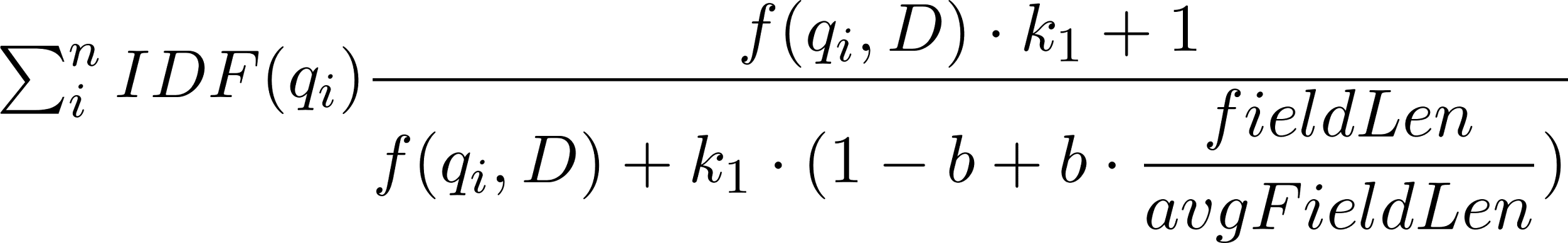
Then we combined the term frequency with inverse document frequency to obtain the TF-IDF weight of each term through Eq. (2), where [](https://www.codecogs.com/eqnedit.php?latex=df_t#0) stands for the number of documents that contain *t*. To initialize the retrieving process, we also computed the weights of both queries and documents then compared the cosine similarity between them.

[](https://www.codecogs.com/eqnedit.php?latex=w_%7Bt%2Cd%7D%20%3D%20tf_%7Bt%2Cd%7D%20%5Ccdot%20%5Clog_%7B10%7D(%5Cfrac%7BN%7D%7Bdf_t%7D)#0) (2)

Another innovation in algorithm was the BM25 document retrieval model we built from scratch as well. Similarly, we computed the query term *q* and the inverse document frequency of the term *IDF(q)*. Note that the IDF component of BM25 measures how often a term occurs in all of the documents and penalizes terms that are common, which is slightly different from the IDF formula in TF-IDF shown in Eq. (3):

 (3)

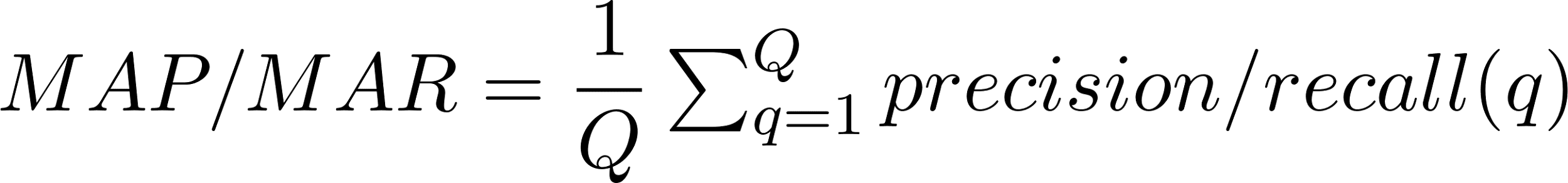
Intuitively, the queries containing rarer terms generally result in greater multipliers, thus they contribute more to the final score. Then after the queries and document texts being parsed, the BM25 score of each article is obtained by Eq. (4):

 (4)

where the term *b* indicates the proportional effect of the length of the document compared to the average length, and the term frequency saturation characteristic *k1* limits how much a single query term can affect the score of a given document.

**3) Contributions in Analysis:**

The major evaluation metrics employed for our system are macro-averaged precision and macro-averaged recall. In general, a macro-average computes the metric independently for each query and then takes the average, thus weighing all queries/classes equally. Mathematically, precision and recall are both computed over individual documents in the prediction against those in the true answer. Precision is the ratio of the number of shared documents to the total number of documents in the prediction, while recall is the ratio of the number of shared documents to the total number of documents in the ground truth. The macro-averaged performance is simply averaging the scores by the number of queries,

[](https://www.codecogs.com/eqnedit.php?latex=MAP%2FMAR%20%3D%20%5Cdfrac%7B1%7D%7BQ%7D%5Csum_%7Bq%3D1%7D%5EQ%20precision%2Frecall(q)#0) (5)

where *Q* is the total number of queries, *AP(q)* is the average precision for query *q*.

**5.1 Method**

**Retrieve-and-Rerank:** The tasks “retrieve” and “rerank” are different only on how large the candidate pool is. It refers to using simple models to retrieve a narrowed list of candidate documents related to a query from a larger pool for optimization purposes, then using more sophisticated models to rerank the selected candidates with advanced scoring schemes.

**TF-IDF**: Known as Term Frequency Inverse Document Frequency, it is an algorithm that transforms natural language text into a numerical representation as indexed values. The main idea is to give more weights to frequent words: “tf” (term frequency) refers to the frequency of a term in a document, and “idf” (inverse document frequency) refers to the frequency of documents that contain the term [Mukesh Chaudhary, 2020].

**BM 25**: It can be seen as an improved version of TF-IDF, which uses the TF-IDF structure but adds more complexity to it. BM25 adds term saturation parameter, multi-term evaluation, document length parameter [Rudi Seitz, 2020].

**BERT**: Known as Bidirectional Encoder Representations from Transformers, it is a more sophisticated embedding model developed by Google. After sentences are tokenized (BERT has its own tokenizer), the words are converted into high-dimensional tensors through a multi-layer network [Chris McCormick, 2019]. Unlike sequential models, BERT is bi-directional and pretrained with two NLP tasks: Masked Language Modeling (MLM) and Next Sentence Prediction (NSP). MLM randomly masks a word in a sentence and predicts it using transformers from both sides at the same time, and NSP takes two adjacent sentences each time to learn the contextual relations between them. More intuitively, BERT differs from Word2vec in that one word can have different embedding representations in different contexts [Pavan Sanagapati, 2021].

**5.2 Experiments and Results**

**Experiment Problem Setup**

Our project aims to build a QA system with experiments on comparing multiple models (TF-IDF, BM25, BERT) that can help return a list of documents relevant to a given query. BERT is known for one of the most advanced NLP models for building a QA system [Dai & Callan, 2019]. As the limited computing resources were not able to support BERT to retrieve documents from the entire mega pool with 300k documents, we decided to implement the Retrieve & Rerank method for optimization purposes. We aimed to validate the hypothesis that the Retrieve & Rerank method is advantaged in consuming less computing resources while achieving satisfying results that are good enough to beat the baseline.

Notice all three models basically do the same task of selecting the most relevant documents from a pool. We differentiated the stages“retrieve” and “rerank” only based on how large the pool is. First, we manually selected 10k random JSON files from a 300k-document pool. Our baseline solution is to retrieve a narrowed down the candidate documents related to a query from the 10k pool simply using TF-IDF without any tokenization. To beat the baseline solution, we used TF-IDF and BM25 with tokenization to narrow down the candidate documents for the reranking stage, we retrieved top 10, 50, 100, and 500 files for each query and calculated the metrics to compare and contrast performances of different models. During the reranking stage, we performed BERT on the retrieved files from the previous step and zoomed in to top 1, 5, and 10 files to evaluate model performance.

**Standards and Metrics**

Given its best results produced by human specialists’ literature review, we compared our performance metrics with respect to it. The evaluation metrics we employed include macro averaged precision (details discussed in the contribution section). The answers/standards we used here were the literature review results used in PyGaggle models. There were 27 queries in total along with 155 document IDs as the standard answers to queries. Notice that the distribution of given answers was imbalanced as query 1 had 26 marked answers but query 27 only had 1 document. We utilized the metadata file to construct links between the file IDs retrieved by standard datasets and the filenames of training data, which prepared us for future analysis.

**4) Results**

A. Retrieving Stage

| **Model** | **Metric** | **Top 10** | **Top 50** | **Top 100** | **Top 500** |
| --- | --- | --- | --- | --- | --- |
| TF-IDF  (no tokenization) ~30min | Precision | 3.7% | 1.78% | 1.37% | 0.39% |
| Recall | 10.93% | 20.25% | 36.75% | 42.82% |
| # Files | 172 | 272 | 523 | 2172 |
| TF-IDF (tokenization 1)  ~5min | Precision | 4.81% | 1.85% | 1.30% | 0.41% |
| Recall | 8.75% | 15.50% | 28.22% | 38.63% |
| # Files | 49 | 211 | 418 | 1770 |
| TF-IDF (tokenization 2)  ~5min | Precision | 4.07% | 1.85% | 1.37% | 0.47% |
| Recall | 14.63% | 20.78% | 37.73% | 47.37% |
| # Files | 72 | 274 | 525 | 2192 |
| BM25  ~3min | Precision | 7.04% | 2.30% | 1.30% | 0.41% |
| Recall | 13.08% | 23.32% | 26.23% | 44.99% |
| # Files | 164 | 691 | 1211 | 4021 |

To locate down on a range of candidates, we chose TF-IDF with tokenization 2 as the final retrieved model since it balanced the performance with running time and also returned a reasonable amount of unique files. Notice that the TFIDF without any tokenization will take too long to execute but still with less recall and BM25, although good in performance, retrieved too many documents (4021 compared to 2192).

B. Reranking Stage

| **Model** | **Metric** | **Top 1** | **Top 3** | **Top 5** |
| --- | --- | --- | --- | --- |
| TF-IDF (tokenization 2) | Precision | 0.0% | 1.23% | 3.7% |
| Recall | 0.0% | 0.15% | 4.93% |
| # Files | 7 | 20 | 38 |
| TF-IDF (tokenization 2) + BERT | Precision | 14.81% | 11.11% | 7.41% |
| Recall | 3.42% | 9.16% | 10.09% |
| # Files | 22 | 40 | 57 |

In the reranking stage, while retrieving and targeting final answers, adding BERT as the reranking stage showed a clear improvement compared to simply TF-IDF (tokenization 2) to limit answers within range of 5.

**6. Compute/Other Resources Used**

We mainly used Colab Pro for this project. Given the limited computational capacity and flexibility, we manually selected 10,000 random samples from the 300k documents pool to optimize the process.

**7. Conclusion**

Our final results have reached the benchmark achieved by the untokenized TF-IDF baseline model, and gone above and beyond in implementing multiple tokenizing and retrieving techniques, with the best performing combination selected for our QA system. We did find this learning process fruitful and challenging. Our initial intention at the beginning of the semester was to build a QA system by virtue of a Knowledge Graph embedding, while as the project progressed, we became aware that the lack of referenceable resources and the lack of our own in-depth knowledge towards this advanced topic would hinder us from moving forward. Meanwhile, we noticed the results of one of the only few available resources from Github was unable to be reproduced on our end given the limited computational resources and package dependency issues on Google Colab. Thanks to our mentor, Sihao, who has been absolutely responsive and helpful throughout the semester, we were guided to refine our focus on the retrieve-and-rerank framework and build the algorithms from scratch.

For future work, we would consider experimenting on more robust ranking models such as monoT5, to improve our system performance. We would also test on more different combinations of models to set up a good pipeline of document retrieval. In terms of computational limits, we would look into other dedicated servers such as AWS, and other database frameworks that support fast compilation with NLP models, like ElasticSearch and Hugging Face. So far, our QA system has succeeded in recommending COVID articles for input queries. As we integrate more advanced algorithms and feed more training samples into the system, we believe our project would be useful as a COVID document searching tool one day.

**References (Exempted from page limit)**

1. Pavan Sanagapati, “Knowledge Graph & NLP Tutorial-(BERT,spaCy,NLTK)”, 2021, *Kaggle*. Retrieved 2/15/2022 from <https://www.kaggle.com/pavansanagapati/knowledge-graph-nlp-tutorial-bert-spacy-nltk>

2. Sandyvarma, “Covid-19: BERT + MeSH Enabled Knowledge Graph”, 2020, *Kaggle*. Retrieved 2/15/2022 from <https://www.kaggle.com/sandyvarma/covid-19-bert-mesh-enabled-knowledge-graph#Abstract>

3. Shahules, “CORD : Tools and Knowledge graphs”, 2020, *Kaggle*. Retrieved 2/15/2022 from <https://www.kaggle.com/shahules/cord-tools-and-knowledge-graphs/notebook>

4. Mukesh Chaudhary, “TF-IDF Vectorizer scikit-learn”, 2020, *Medium*. Retrieved 3/20/2022 from <https://medium.com/@cmukesh8688/tf-idf-vectorizer-scikit-learn-dbc0244a911a>

5. Ragini Patil, “Question Answering System using A TF-IDF Method”, 2020, *International Journal of Science, Engineering and Technology*. Retrieved 3/20/2022 from <https://www.ijset.in/wp-content/uploads/IJSET_V8_issue4_257.pdf>

6. Dhilip Subramanian, “Content-Based Recommendation System using Word Embeddings”, 2020, *KDnuggets*. Retrieved 3/20/2022 from <https://www.kdnuggets.com/2020/08/content-based-recommendation-system-word-embeddings.html>

7. Chris McCormick, “BERT Word Embeddings Tutorial”, 2019, *McCormickML*. Retrieved 3/20/2022 from <https://mccormickml.com/2019/05/14/BERT-word-embeddings-tutorial/#what-is-bert>

8. Ubiquitous Knowledge Processing Lab, “Retrieve & Re-Rank”, 2021, *GitHub*. Retrieved 4/1/2022 from <https://github.com/UKPLab/sentence-transformers/tree/master/examples/applications/retrieve_rerank>

9. Chris McCormick, “How To Build Your Own Question Answering System”, 2021, *McCormickML*. Retrieved 4/1/2022 from ​​<https://mccormickml.com/2021/05/27/question-answering-system-tf-idf/>

10. Rudi Seitz, “Understanding TF-IDF and BM-25”, 2020, *KMW Technology*. Retrieved 4/1/2022 from <https://kmwllc.com/index.php/2020/03/20/understanding-tf-idf-and-bm-25/>

11. Zhuyun Dai & Jamie Callan, “ Deeper Text Understanding for IR with Contextual Neural Language Modeling”, 2019, *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. Retrieved 4/1/2022 from <https://arxiv.org/abs/1905.09217>

12. Castorini, “PyGaggle: Neural Ranking Baselines on CovidQA”, 2021, GitHub, Retrieved 3/20/2022 from <https://github.com/castorini/pygaggle>

**Broader Dissemination Information:**

Your report title and the list of team members will be published on the class website. Would you also like your pdf report to be published?

* YES

If your answer to the above question is yes, are there any other links to github / youtube / blog post / project website that you would like to publish alongside the report? If so, list them here.

* PyGaggle: Neural Ranking Baselines on CovidQA: <https://github.com/castorini/pygaggle/blob/master/docs/experiments-covidqa.md>

##### 

##### **Proposed Schedule (Exempted from page limit)**

| **PERSON (S)** | **TASK (S)** | **Wk5** | | | | **Wk6** | | | | **Wk7** | | | | **Wk8** | | | | **Wk9** | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **MAR** | | | | **APR** | | | | | | | | | | | | | | | |
| S25 | M27 | W28 | F30 | S3 | M4 | W6 | F8 | S10 | M11 | W13 | F15 | S17 | M18 | W20 | F22 | S24 | M25 | W27 |  |
| **All** | Midpoint Report |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Zhengjia, Shuyan** | Reproducing Baseline |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Zhengjia** | Trouble Shooting |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Xiaoyu, Shuyan** | Data Preprocessing |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Zhengjia, Xiaoyu** | TF-IDF |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **All** | Checkpoint with Mentor |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Xiaoyu** | BM25 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Zhengjia** | BERT |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Shuyan** | Model Tuning |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **All** | Final Report Wrapup |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

#### **Midterm Report (Exempted from page limit)**

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### **Supplementary Materials (Exempted from page limit)**

#### Main Work Repository: <https://drive.google.com/file/d/1KgzpsEGf63kI7F2BhRXu7QgZpSrMHSxe/view?usp=sharing>

#### Experiment Results:

