Document Retrieval System

Group 108

Vishnu Vardan and Shreyas Sahu

Dec - 2024

Abstract

Finding relevant data from a database is a time-consuming process. This is especially true for medical records as there are huge databases of patient information. Traditionally, a search engine is built to query for records, but such engines do not understand semantics and do not provide answers, but rather just bring the relevant record to the user's focus. This retrieval system aims to solve this issue by not only finding the relevant record but also answering the query that the user has.

Introduction

Traditionally, when a practitioner needs specific medical records from a database, he/she is presented with a search engine that can query the database for relevant medical records and provide it to the user. This has 2 inefficiencies:

1. The engine searches for exact/similar text in the database, without considering semantics.
2. The retrieved data still requires to be processed manually by the user.

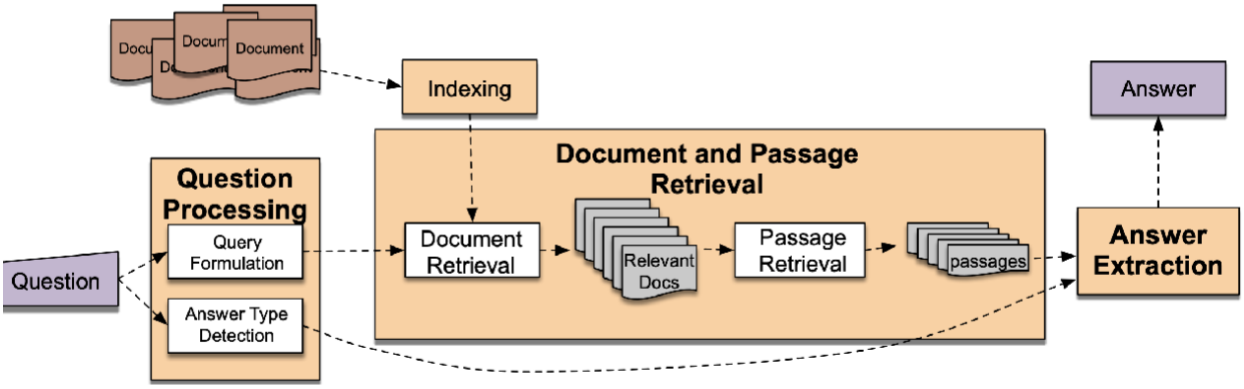
To solve these inefficiencies, a document retrieval and answering system is proposed in this report.

This system (Hereafter referred to as "system") will take in a query entered by the user and a vector-space model (selected by the user). The selected model will retrieve the top n documents relevant to the query, and pass them to a secondary BERT - model, which will formulate an answer to the query, taking into context the documents sent by the vector-space model.

The system has an accuracy of 70% in retrieving the relevant document, and an accuracy of 80% in answering the query correctly. There are a few inefficiencies (read *Discussion*) in the current system, and they can be improved in successive iterations.

Methodology

The system can be broadly split into two distinct layers, the retrieval layer, and the QA layer.



The retrieval layer deals with parsing the query into tokens and with the help of different cosine similarities, it figures out the 10 most relevant documents. To do this, the retrieval layer uses one of three available models:

1. TF-IDF + KNN
2. SBERT + FAISS
3. BM25

In this part of the report, we shall investigate the pros and cons of each model in detail.

TFIDF + KNN

This first part of this model uses TF-IDF to create an embedding of the database, and the second part uses KNN to find the relevant set of documents.

The TF-IDF model is configured as follows:

- `Lowercase – True: Changes all the words to lowercase so that the corpus is case insensitive.`
- `Analyzer – word: Considers complete words in the corpus instead of characters.`

- `stop_words – english`: Removes noise from the text by excluding frequently used grammatical words.
- `Binary – True:` This parameter decides if each word's weight is equal or not. This is set to true as the corpus contains a lot of unique names, hence they should have the same weightage as other more common words.
- `max_df – 0.9:` This filters out 1/10th of the most common words.
- `ngram_range – (1,2):` This is used to calculate both unigrams and bigrams.

The second part of the model uses KNN trained with the embeddings created by TF-IDF to calculate and return the most relevant documents. The configuration is as follows:

- `n_neighbors – 10`: Returns the 10 most relevant documents
- `Metric – cosine`: Calculates the relevance using cosine similarity. Such models are called vector–space models.

SBERT + FAISS

The first part of the approach uses a pretrained Sentence-BERT model (all-MiniLM-L12-v2) to create vector embeddings of the corpus. This specific model was chosen after multiple different trials and for the following reasons:

- Lightweight: It has only 6 layers, 384 dimensions and 23 million parameters. So, the response times are acceptable even on lower end hardware.
- Context length: It has a context length of 512 words. This is sufficient as all the rows of the dataset are less than 500 words.
- Performance: This model, combined with FAISS, can achieve an accuracy of 70%, tying it up in performance with the other methods approached in this project.
- Semantics: This is the only model capable of understanding semantics, so similar words have similar vectors. This helps when the user replaces medical terms with colloquial words.
- Embeddings: There is no need to re-create the word embeddings when a new document is added to the database. Instead, the embedding of the new document can be manually appended to the existing word embeddings. This saves valuable time and computing costs.

BM25

This approach uses BM25kapi to create a ranking system for each document based off the user's query. This approach is distinctly different from the other approaches as the other 2 methods use vector-space models while BM25kapi uses a ranking algorithm.

The BM25 Ranking algorithm is as follows:

$$\text{score}(D,Q) = \sum_{t \in Q} \text{IDF}(t) \cdot \frac{\text{tf}(t,D) \cdot (k_1 + 1)}{\text{tf}(t,D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)}$$

Some points to note from the equation:

- BM25 introduces a saturation effect for term frequency, so the impact of additional occurrences of a term diminishes as the frequency increases.
- Longer documents are penalized to avoid favoring them over shorter, more concise documents.
- k_1 and *b* allow for fine-tuning the algorithm's behavior for specific types of documents or queries.

Question Answering (QA) Layer:

The retrieval layer sends a list of 10 indices of documents that are the most relevant to the user's query. The user query and the most relevant documents as context are passed as parameters to the BioBert language model fine-tuned for Question Answering. The model then returns a span of text from the document that is the most relevant to the query.

BioBert for Question Answering:

BioBert (Bidirectional Encoder Representations from Transformers for Biomedical Text Mining) is a domain-specific BERT-based model pre-trained on large Biomedical corpora. It outperforms BERT in various biomedical text mining tasks. It has the same model architecture as BERT.

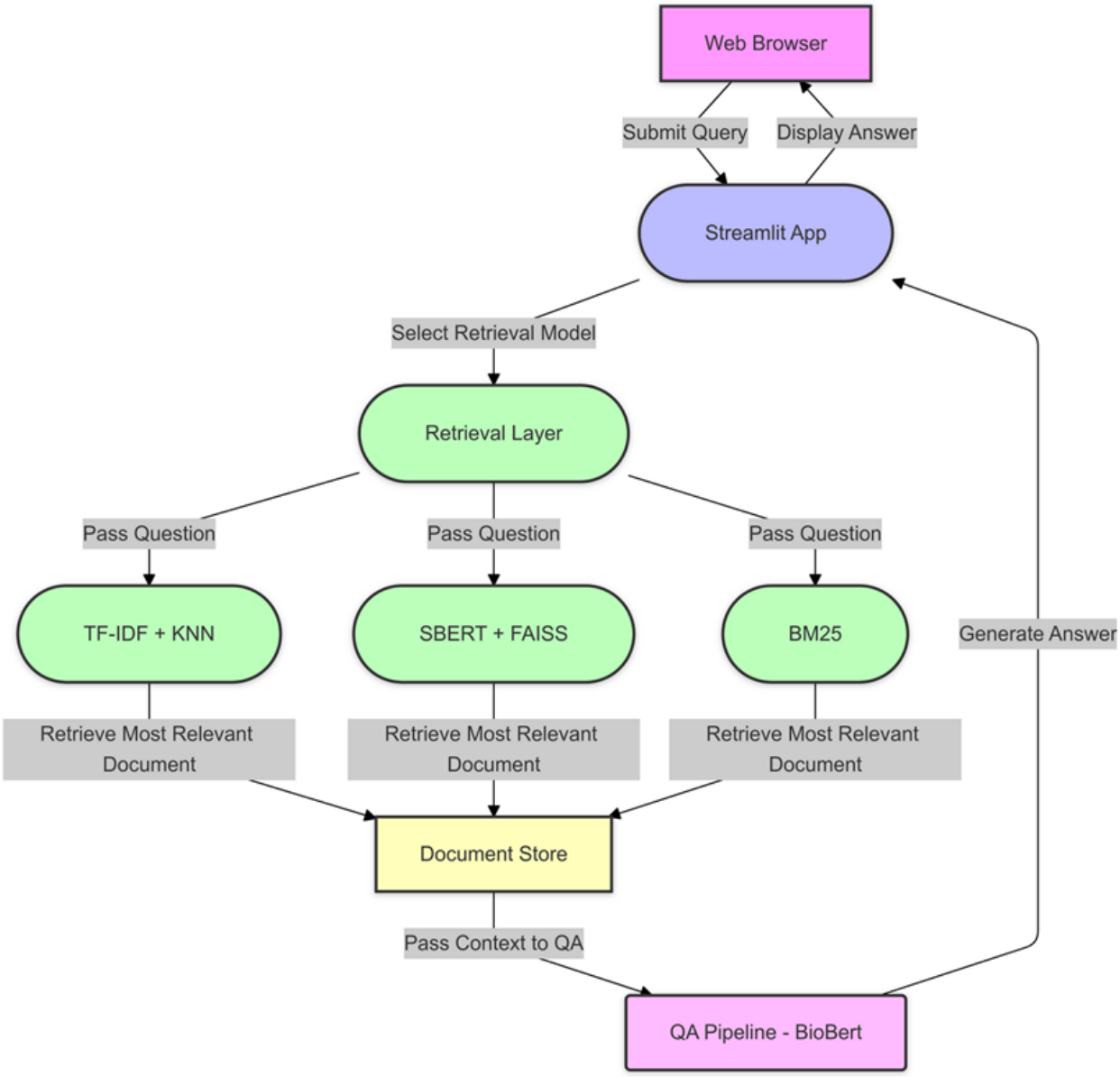
The model we have used for the Question Answering layer is "biobert-large-cased-v1.1-squad", which is a BioBert model fine-tuned on the SQUAD (Stanford Question Answering) dataset.

Steps followed in the Question Answering layer:

1. The question and the most relevant document(context) are obtained from layer 1 (document retrieval) of the system.
2. The tokenizer associated with the model is loaded.
3. The model with pre-trained weights is loaded.
4. A question-answering pipeline is created which is initialized with the loaded tokenizer and model.

5. The question and the context are passed to the pipeline
6. The pipeline tokenizes the inputs and passes them to the model.
7. The model returns the best span of text that is most relevant to the question.

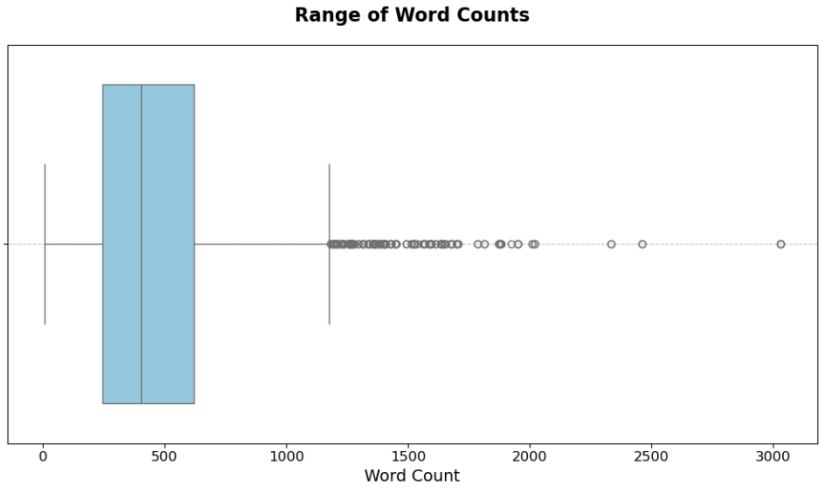
The overall model can be summarized by the image below.



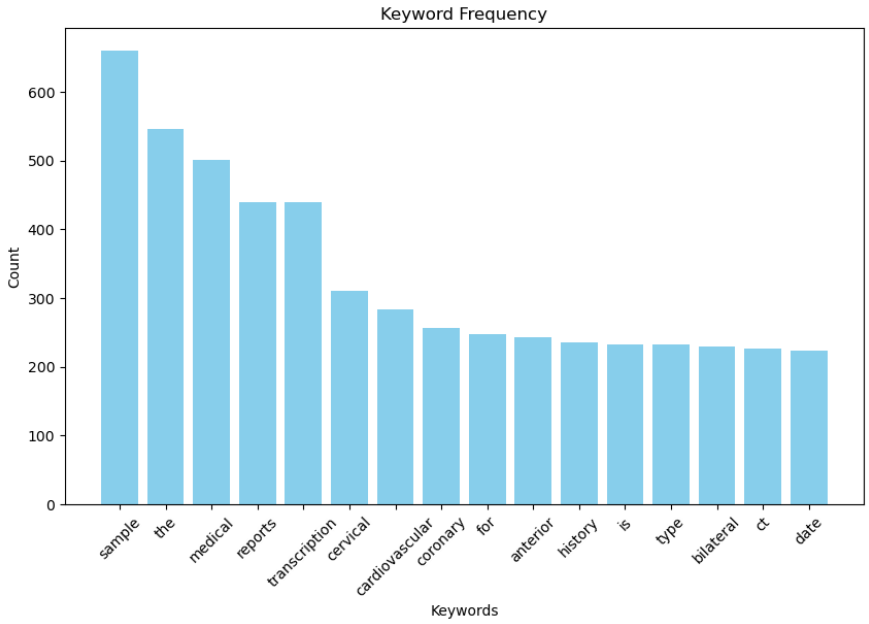
Data

The data for this project was retrieved from mtsamples.com

This is a synthetic dataset consisting of medical reports varying from doctor's notes to Diet/Nutrition and surgery reports. The following plots below provide a visual representation of the salient features of the dataset.



Range of word-counts in the database.



Frequency of most used keywords in the database.

Medical Specialty:
Allergy / Immunology

Sample Name: Allergic Rhinitis

Description: A 23-year-old white female presents with complaint of allergies.
(Medical Transcription Sample Report)

SUBJECTIVE: This 23-year-old white female presents with complaint of allergies. She used to have allergies when she lived in Seattle but she thinks they are worse here. In the past, she has tried Claritin, and Zyrtec. Both worked for short time but then seemed to lose effectiveness. She has used Allegra also. She used that last summer and she began using it again two weeks ago. It does not appear to be working very well. She has used over-the-counter sprays but no prescription nasal sprays. She does have asthma but doest not require daily medication for this and does not think it is flaring up.

MEDICATIONS: Her only medication currently is Ortho Tri-Cyclen and the Allegra.

ALLERGIES: She has no known medicine allergies.

OBJECTIVE:
Vitals: Weight was 130 pounds and blood pressure 124/78.
HEENT: Her throat was mildly erythematous without exudate. Nasal mucosa was erythematous and swollen. Only clear drainage was seen. TMs were clear.
Neck: Supple without adenopathy.

Lungs: Clear.

ASSESSMENT: Allergic rhinitis.

PLAN:
1. She will try Zyrtec instead of Allegra again. Another option will be to use loratadine. She does not think she has prescription coverage so that might be cheaper.
2. Samples of Nasonex two sprays in each nostril given for three weeks. A prescription was written as well.

Dataset Sample

Results

Retrieval Layer:

The layer's performance was assessed using two key metrics:

- **Accuracy**: This measures the proportion of correct predictions made by the model out of all predictions, providing a straightforward evaluation of its overall correctness.
- **Mean Reciprocal Rank (MRR)**: This evaluates the quality of the ranking, focusing on the position of the first relevant result. Higher MRR values indicate better performance in ranking relevant items earlier.

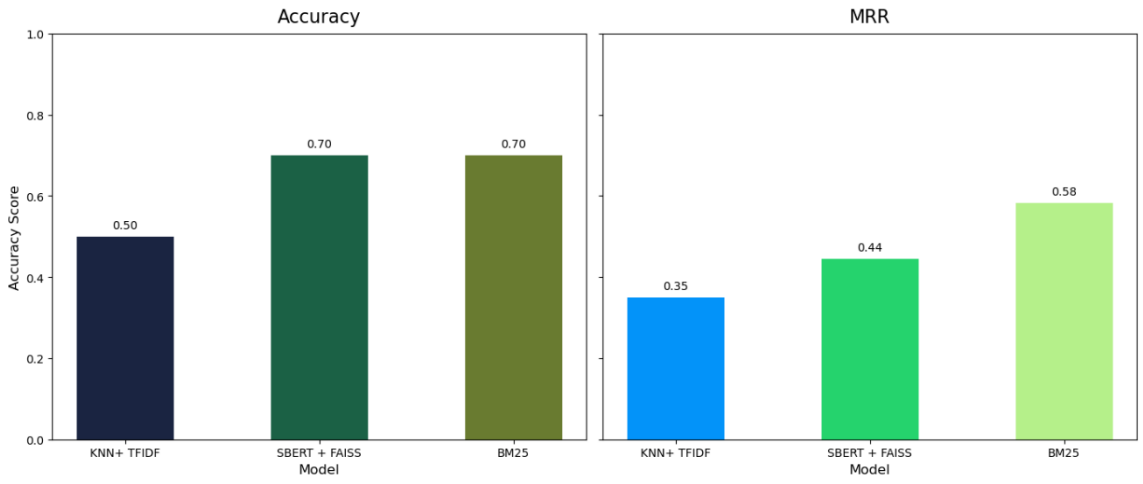
Note: While MRR provides a metric that scores the correct prediction rate and the document's position, it is not relevant for this system. This is because the top 10 documents are sent to the QA layer regardless of their position in the results. Hence, the accuracy metric is a better indicator of the performance of this layer.

An ideal use of MRR would be in search engines, where the position of the relevant document is also as important as retrieving the relevant document.

Finding the recall, and by extension the f1-score of the retrieval layer is not viable, as for every query, there is always only 1 relevant document.

The summary of tests of the retrieval layer are presented below:

Metric	TFIDF + KNN	SBERT + FAISS	BM25
Accuracy	0.5	0.7	0.7
MRR	0.35	0.36	0.58



Testing results visualized.

Question Answering Layer:

For the question answering layer, we have used the metric “average F1 score” to test the performance. For a list of questions, we prepared the ground truth answers and ran the model to predict the answers for the questions. We then calculated the F1 score for each question and computed the average for the complete list.

Formula:

$$recall = (number\_of\_matching\_tokens) \ / \ len(number\_of\_tokens\_predicted\_answer)$$

$$precision = (number\_of\_matching\_tokens) \ / \ len(number\_of\_tokens\_ground\_truth\_answer)$$

$$\mathrm{F1} = 2 * \mathrm{precision} * \mathrm{recall} \ / \ (\mathrm{precision} + \mathrm{recall})$$

Test results:

We achieved an average F1-score of 0.8 for the Question Answering layer of the system.

Discussions

There were multiple approaches to solve this problem statement. The first method involved training existing BERT models on the corpus and evaluating their performance.

This method had the following drawbacks:

- Performance: The model was showing significantly low performance. With an accuracy rate of less than 50%, it was not ideal to proceed with this approach. On close inspection, we believe that the accuracy was extremely low due to the similarity between the documents. Since most documents had the same keywords and were in the same domain, all the different models that were trained on this dataset suffered from the same issues - cross referencing across documents.
- Retraining: Since the weights are trained on the dataset, any new addition would mean that you would have to retrain the model again from scratch. This was not feasible as in an environment where new documents are added constantly.

After moving on from this approach, we explored the currently implemented approach, where the system is split into two layers, the first layer retrieving the relevant documents and the second layer answering the query.

This architecture was not finalized, and we explored the insertion of a middle layer between the retrieval layer and the QA layer. This intermediate layer would find the most relevant sentence from the most relevant document. The intention of creating the intermediate layer was to increase the accuracy of the QA layer by providing it only the singular relevant sentence, but the idea was scrapped as the insertion of the intermediate layer increased the processing time significantly while not providing measurable gains in accuracy.

Another path we explored was using different models for the retrieval layer. One such example is RoBERTa , but it was not suited to create embeddings for semantic tasks. So, we decided to go for SBERT instead.

While the current system provides satisfactory results, it can be improved in the following directions:

- The retrieval layer always sends the 10 most relevant documents. Instead of setting a hard rule on the number of documents, we can set a confidence threshold, where the number of relevant documents can be dynamically decided at runtime based on the model.
- An extension of the first improvement would be to dynamically choose the best model based on the confidence levels of all three models. While this might give accurate results, the processing time will increase significantly as all three models will need to be run for each query. So, a sufficient increase in processing power is required.

References

- Tests - https://zilliz.com/learn/information-retrieval-metrics
- MRR - https://weaviate.io/blog/retrieval-evaluation-metrics
- Models save and load - https://www.tensorflow.org/tutorials/keras/save_and_load
- Deployment - https://blog.streamlit.io/host-your-streamlit-app-for-free/
- SBERT and FAISS:
    - https://github.com/beir-cellar/beir/blob/main/examples/retrieval/evaluation/dense/evaluate_faiss_dense.py
    - https://towardsdatascience.com/billion-scale-semantic-similarity-search-with-faiss-sbert-c845614962e2
    - https://www.analyticsvidhya.com/blog/2022/10/a-gentle-introduction-to-roberta/

- Question Answering Model:
    - https://huggingface.co/models?pipeline_tag=question-answering&sort=downloads&search=biobert
- QA tests:
    - https://gist.github.com/primaryobjects/6b98a8793556659fd207636c53679a1a
    - Question Answering Lecture Slides from class modules.
- Architecture:
    - https://spotintelligence.com/2023/10/18/document-retrieval/
    - https://www.ibm.com/think/topics/information-retrieval
    - https://www.chatbase.co/blog/document-parsing
    - Question Answering Lecture Slides from class modules.
- Dataset:
    - https://physionet.org/content/mimiciii/1.4/
    - https://physionet.org/content/mimic-iv-ed-demo/2.2/
    - https://www.mtsamples.com/
    - https://www.kaggle.com/datasets/tboyle10/medicaltranscriptions

Appendix

Code Structure:

- *StreamlitUI.py* – This file represents the UI deployed. It loads the models, creates the UI, then processes the query that the user enters.
- *preprocess.ipynb* – This file reads the dataset downloaded from Kaggle, fills in the rows that do not have keywords, then adds synthetic names to all the rows.
- *BERT_FAISS.ipynb* - This file reads the preprocessed data, creates the embedding and retrieving models using the dataset, and then dumps them into a .joblib file for the UI to read from disk later.
- *BM25.ipynb* - This file creates the BM25 model using the preprocessed data and dumps the model to a .joblib file.
- *Dataset_EDA.ipynb* - This file does some visualization on the dataset.
- *document_retriever_tests.ipynb* - This file runs tests on all models and provides accuracy, MRR and F1 scores.

- *tests.py* – This python file is identical to document_retriever_tests.ipynb, but provided in python format for easy execution. (**please note that this will take a significant time to run**)
- *requirements.txt* - This file contains all the required external libraries and different modules required to run the program.

User Manual:

- The model has been deployed at https://vv-22-nlp-doc-retrieval-streamlitui-ujoqg0.streamlit.app/ . You can open this link to see the deployed program.
- Alternatively, you can run this program locally. To do so, you must have the external libraries present in requirements.txt.
    - go to the root directory and run this command with terminal:

      streamlit run StreamlitUI.py