# Document Retrieval using Embedding Search

Samhit Chowdary Bhogavalli (sb4845), Mooizz Abdul (ma4496).

November 2024

## 1 Overview

In the final project, We aim to tackle the problem of parallelizing document retrieval using embedding search. Document retrieval is a specific form of text retrieval used in many applications in medical, legal, and a lot of other industries. The problem becomes interesting when we have a large number of documents to search for, and the need for parallelization arises. The retrieval of k similar documents given a query mainly involves two stages, we believe that parallelization helps in these two stages, ie

- Efficient pre-processing of a large number of documents to create embedding vectors.
- Efficient Retrieval of k best documents by comparing the similarity between these embedding vectors.

## 2 Background

To perform Document Retrieval using embeddings, Given a large set of documents, the process is divided into three stages.

### 2.1 Text Normalization

In this step, the text in the documents is normalized, for example converting the text to lowercase, stop-word deletion, and optionally tokenization.

### 2.2 Pre-processing the Normalized Documents: Creating Embeddings for the dataset

#### 2.2.1 TF-IDF

TF-IDF [3] [2] is a statistical measure that evaluates the importance of words in a document relative to a collection of documents.

Term Frequency (TF) measures how frequently a term occurs in a document, calculated as the ratio of a term's occurrence to the total number of terms in the document.

Inverse Document Frequency (IDF) measures the term's importance across all documents, computed as the logarithm of the ratio of total documents to the number of documents containing the term.

#### 2.2.2 Creating TF-IDF vectors for each Document

The final TF-IDF score for each term is the product of its TF and IDF values, resulting in a numerical vector representation of each document. This transformation converts text documents into sparse numerical embedding vectors.

### 2.3 Retrieving best k documents: Embedding Search using TF-IDF

To retrieve the k most relevant documents for a given query, the system first converts the query into a TF-IDF vector using the same vocabulary and IDF values computed during the preprocessing stage.

Then, the similarity between the query vector and each document's TF-IDF vector is computed using cosine similarity, which measures the cosine of the angle between two vectors, effectively comparing their directional similarity regardless of magnitude. This operation yields a similarity score between 0 and 1 for each document, where a higher score indicates greater relevance to the query.

Finally, the documents are ranked based on these similarity scores, and the top k documents with the highest scores are returned as the search results.

Both these stages process documents and embeddings akin to the Map-Reduce [1] framework. In the first stage, the mapper handles term frequency calculation and the reducer handles the tf-idf vector creation. in the second stage, the mapper handles the embedding similarity calculation and the reducer handles the similarity sorting and getting the best k documents.

## 3 Parallelization Overview

There is scope for parallelization in both the pre-processing stage and retrieval stage.

### 3.1 Pre-processing Stage

- Given K cores, the number of documents can be split into k subsets
- For each subset of documents, we calculate term frequency.
- Using these term frequency maps, we get a vocabulary of words and also IDF values for each term.
- We form TF-IDF vectors for each document

### 3.2 Retrieval Stage

- Given K cores, we divide the task of computing embedding similarities into K sets. Each Set contains a list of embeddings
- For each subset, we calculate the cosine similarity of query embedding with the list of embeddings in a subset.
- Finally, we rank the embeddings according to cosine similarity and get the best k.

We expect challenges in the pre-processing stage with file I/O and text normalization.

Input data: We plan to use a document dataset akin to the Cranfield medical document dataset Parallelization Strategies: We mostly plan to use Par monad but we will experiment with Accelerate Library.

## References


[1] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, January 2008.

[2] Juan Ramos. Using tf-idf to determine word relevance in document queries, 1999.

[3] Jake Tae. https://jaketae.github.io/study/tf-idf/, 2017.