University of Sheffield

Document Retrieval Report

Andreas Evripidou

COM3110: Text Processing

Department of Computer Science

November 6, 2023

1 Introduction

This report is part of the first assignment of the COM3110 Text Processing module. The assignment task was to finish the implementation of a given Information Retrieval (IR) engine. The partial IR engine could already extract and tokenise the terms of documents and queries and create an inverted index for the whole collection. The IR engine also had options for using a stoplist to ignore certain terms that add little to no meaning in the retrieval process. The last feature the given IR engine already had was stemming, which takes words and reduces them to their stem so that the size of the inverted index is minimised. To finish the implementation of the IR engine, we had to add a comparison of the given queries with the documents in the collection. The comparison needed to be done by creating vectors for both the documents and the queries and measuring their similarity using the Cosine Similarity.

The cosine similarity between a query vector $\mathbf{q}$ and a document vector $\mathbf{d}$ is calculated as follows:

$$\text{Cosine Similarity}(\mathbf{q}, \mathbf{d}) = \frac{\sum_{i=1}^{n} q_i \cdot d_i}{\sqrt{\sum_{i=1}^{n} q_i^2} \cdot \sqrt{\sum_{i=1}^{n} d_i^2}} \tag{1}$$

Where:

$$\begin{aligned} \mathbf{q} &= [q_1, q_2, \ldots, q_n] \text{ (Query vector)} \\ \mathbf{d} &= [d_1, d_2, \ldots, d_n] \text{ (Document vector)} \\ \mathbf{q_i}/\mathbf{d_i} &= \text{number of words in the query or document multiplied by their weight} \\ n &= \text{number of terms in the query} \end{aligned}$$

In addition, three methods had to be implemented to weigh the terms. The three methods were Binary, Term Frequency (TF) and Term Frequency-Inverse Document Frequency (TF-IDF). The binary method states that a term in a document/query is either relevant and valued with **1** or irrelevant and valued with **0**. The term frequency method, as the name suggests, weights each term based on the number of appearances of a term in the specific document/query. The last weighting scheme, TF-IDF, weights each term by multiplying the Term Frequency of a term in the document/query by the Inverse Document Frequency, which moderates the effect of extremely common words.

2 Implementation

The implementation found in my_retriver.py uses the class Retrieve to process the queries. When you create a Retriever object with an inverted index (a dictionary of {term: {doc_id: term_count}}) and an option of the weighting scheme (a string of either "binary", "tf" or "tfidf"), both are saved into the state of the object. After the state is saved, the number of documents along with the length of each document vector (a dictionary of {doc_id: length}) are calculated and saved in the object's state. To process a query, the function for_query is used. The function takes a query (a list of tokenised terms) as a parameter and filters out terms not found in the collection of documents. After that, a subset of the documents is computed only to include documents that have at least one of the query terms. Those two steps are done as an optimisation to only deal with relevant terms and documents. Once the query terms and documents are filtered, the compute_vectors function is called with the query and relevant documents. The compute_vectors function computes the vector space model by taking into consideration the selcted weighting scheme. There are two helper functions for deciding the weighting scheme, get_query_weighting and get_weighting and each one of them is using sub-functions for calculating a term's TF or TFIDF weights. Once the vector space model is compute, the for_query function loops through the relevant documents and computes the cosine similarity of the query and a document using the cosine_similarity function and stores them in a dictionary named scores ({doc_id: score}). Finaly the score's dictionary is passed to the rank_documents function to sort them by the cosine similarity scores and a sorted list with the document_id is returned. This implementation heavily uses dictionaries which are recommended for text processing since you can faster access their values by the dictionary key and avoid searching through a list. In terms of time complexity, the worst case of this implementation is $\mathcal{O}(td)$, where t is the number of terms in the collection and d is the number of documents.

3 Performance

In Table 1, P stands for Precision, R for Recall and F for F-measure.

	Binary	TF	TF-IDF
	P: 0.07 / R: 0.06 / F: 0.06	P: 0.08 / R: 0.06 / F: 0.07	P: 0.21 / R: 0.17 / F: 0.18
Stoplist	P: 0.13 / R: 0.10 / F: 0.12	P: 0.17 / R: 0.13 / F: 0.15	P: 0.22 / R: 0.18 / F: 0.19
Stemming	P: 0.10 / R: 0.08 / F: 0.08	P: 0.11 / R: 0.09 / F: 0.10	P: 0.26 / R: 0.21 / F: 0.23
Stemming/Stoplist	P: 0.16 / R: 0.13 / F: 0.15	P: 0.19 / R: 0.15 / F: 0.17	P: 0.27 / R: 0.22 / F: 0.24

Table 1: Performance Results of each weighting method

3.1 Results of weighting schemes

From Table 1, the results of our test are as expected. We can see that overall, the Binary method for term writing produces the lowest marks. The Term Frequency method increases the retrieval efficiency slightly in terms of Recall and F-measure but more in Precision. Finally, the TF-IDF method returns the best results of all three weighting schemes in all aspects of Precision, Recall and F-measure.

3.2 Results without Stoplist or Stemming

In the test without stemming or stoplist where applied, we can see that we had the lowest results. Another observation encountered during the test was that the retriever was taking longer than the rest; that is due to the fact without stemming and stoplist, the inverted index dictionary contains terms irrelevant to the retrieval and terms with the same stem.

3.3 Results with Stoplist

When using the stoplist option, results are mixed. For the Binary and TF weight schemes, we can see a good increase in all three measures. However, for the TF-IDF scheme, there is only a slight improvement. This must be because Binary and TF schemes rely a lot on the occurrences of words in a document. In contrast, the TF-IDF weighting scheme moderates the relevance of widespread terms using the inverse document frequency (IDF). This makes the effect of irrelevant words bigger in the algorithms that rely heavily on the word count.

3.4 Results with Stemming

Like using a stoplist, stemming has a positive effect but differs for each algorithm. This time, the Binary and TF algorithms have a slight increase in performance, but the TF-IDF algorithm gets a massive boost. Once words with the same stem are combined into one term, the effect of Term Frequency increases the accuracy of the retrieval. We can see that from the results of the TF-IDF algorithm. However, for the binary and TF schemes, the result is moderate since the lack of a stoplist makes irrelevant terms weigh more.

3.5 Results with Stoplist and Stemming

From the results, it is clear that using both stemming and stoplist produces the best retrieval accuracy and time efficiency. The results are better in all three weighting methods and efficiency measurements.

4 Conclusion

In conclusion, the TF-IDF algorithm combined with stemming and a stoplist stands out as the most optimal choice. As expected, this is due to the notable differences in complexity compared to other schemes, which considers both the term frequency and the inverse document frequency. The use of stemming and a stoplist significantly enhances the efficiency of the information retrieval process by reducing the dimensionality of the terms and eliminating common stop words, resulting in more accurate and relevant search results.