University of
Sheffield

Document Retrieval Assignment

Jonathan Lupu

A report submitted in fulfilment of the requirements
for the degree of MComp in Computer Science

in the

School of Computer Science

November 19, 2025

Chapter 1 - Implementation

1.1 Initialisation

When initialising the Retriever class, the Doc-Term matrix is computed using the inputted weights method (binary, tf, tfidf). This matrix is available throughout the class. The matrix computation function is called with the respective term_weighting value.

1.2 Doc-Term Vector Computation

The Doc-Term matrix, an $m \times n$ sparse matrix, stores the weightings between documents and terms. Implemented as an array with dictionaries, it allows easy iteration and storing of term-weighting pairs. Since arrays are 0-based, accessing a document with id i requires $i - 1$ as the index. These vectors are used to compute Cosine Similarity in subsequent stages.

1.2.1 Computing Binary Weighting

Binary weighting is the most basic form of weightings to compute by marking with True or 1 that a term is present in the document. This was done by iterating through the reverse index to access the dictionary that stores the document id and the number of times the term appears. This information was stored in the matrix at index $id - 1$ (as it is 0-based array) and then marking term x present with a 1 e.g. $\{x : 1\}$. This continues until all terms in the reverse index are computed.

1.2.2 Computing TF Weightings

Term frequency (TF) weightings use the count of terms to weigh them. The most basic implementation used at the start was iterating through the terms in the reverse index, and storing the raw count of the term given as the value to each document id key. However, an improvement was implemented by dampening the effect of high-frequency terms. This was done to ensure that terms with lower frequency will still have a positive impact. The value was smoothened using $1 + \log(count)$

1.2.3 Computing TF-IDF Weightings

Computing TF-IDF weightings involves a similar approach to TF but with additional steps. The function iterates through each unique term in the reverse index, calculating df_w (the number of documents the term appears in) by accessing the dictionary using the term as a key. Next, IDF is calculated as $1 + \log \frac{NumberDocs+1}{df_w+1}$. Adding the constant 1 to the numerator and denominator introduces a smoothing factor, preventing zero divisions. Adding the constant 1 to the IDF equation is so that terms with 0 IDF will not be ignored [1]. After computing IDF it is combined with TF by multiplying each other and subsequently stored in the Doc-Term matrix.

1.3 Query Vector Computation

Queries were computed similarly to Doc-Term matrix. However, the only difference was using the terms in the given query rather than reverse index. Additionally, TF-IDF and the TF functions rely on a two pass method. Where the first pass goes through the list of terms and counts the occurrence, and the second pass does the calculation. They use the same formulas (with the smoothening) just on different data.

1.4 Cosine Similarity

Cosine similarity was implemented to calculate the similarity between a query and the documents. Similarity scores were calculated for each document and stored in a dictionary in the form {doc_id : score}. The function goes through the regular cosine similarity formula, computing $\sum q_i d_i$ for each term in the document vector. This is possible using the .get method, which will use 0 by default if the term in the query doesn't exist. Next, the magnitude of the documents, the sum of weightings squared, and then the root, are calculated. The two are then divided and saved into the dictionary.

Chapter 2 - Results

The results from this information retrieval system vary depending on the weighting used and pre-processing applied to the terms. As mentioned in the previous chapter, there are three types of weighting: binary (default), term frequency (TF), and term frequency-inverse document frequency (TF-IDF). Stemming can combine morphological variants of a term, and a stop list can exclude non-content terms (e.g., a, and, but, also). These terms are often frequent and don't provide useful information, so they're discarded when computing the vector. Applying both stemming and a stop list can increase the impact of the score.

2.1 Scores

The tables below show the difference between the different processes applied for a given weighting method.

Criteria	None	Stemming	Stoplist	Both
Relevant retrieved	44	60	81	105
Precision	0.07	0.09	0.13	0.16
Recall	0.06	0.08	0.1	0.13
F-measure	0.06	0.08	0.11	0.15

Table 2.1: Table showing score when evaluating Binary weighting with different pre-processing methods

Criteria	None	Stemming	Stoplist	Both
Relevant retrieved	68	85	97	117
Precision	0.11	0.13	0.15	0.18
Recall	0.09	0.11	0.12	0.15
F-measure	0.09	0.12	0.14	0.16

Table 2.2: Table showing score when evaluating TF weighting with different pre-processing methods

Criteria	None	Stemming	Stoplist	Both
Relevant retrieved	117	159	121	170
Precision	0.18	0.25	0.19	0.27
Recall	0.15	0.2	0.15	0.21
F-measure	0.16	0.22	0.17	0.24

Table 2.3: Table showing score when evaluating TF-IDF weighting with different pre-processing methods

2.2 Conclusion

The tables compare the performance of different weightings and pre-processing functions. Table 2.1 shows that binary weighting performs the worst, even with stemming and stoplist. It achieves a precision of 0.16, the lowest compared to TF and TF-IDF. This is because binary weighting doesn't consider term frequency, which in turn reduces similarity measures. However, using stoplist with binary weighting increases precision by 85% from 0.07 to 0.13. Stemming only slightly improves the scores across all three criteria. Table 2.2 shows how TF outperforms Binary even without any pre-processing, achieving a score of 0.11 for precision and an F-measure of 0.09, which is better than Binary with none and with just stemming. However, their scores when using both are very similar, with only a 12.5% difference. Finally, comparing with TF-IDF, this method outperforms the other weightings by a significant margin. The highest precision is 0.27, which is 0.09 more than TF with stemming and stoplist and 0.11 more than Binary with stemming and stoplist. Furthermore, TF-IDF has the highest F-measure indicating a higher level of overall effectiveness. One key observation is how using just stoplist with TF-IDF doesn't have much of an impact. TF-IDF already penalises common words because high document frequency lowers their IDF component, and when combined in the equation, $tf \times idf$, the term will receive a lower weighting.

Bibliography

[1] SCIKIT LEARN. sklearn feature extraction — scikit-learn 0.23.1 documentation.