Jiacong Wu

COM3110 Document Retrieval Assignment Report

Implementation description

When the `Retriever` class is instantiated, a method `get_tfs` is called to obtain a dictionary which maps all document IDs in the collection, to terms, to the terms' frequency (tf) in the document. `for_query` is called for each query, each time the query is preprocessed to remove all terms which don't occur in the index. The query is then transformed into a dictionary mapping query terms to tf in the query, as with document tfs, to allow for convenient simultaneous access to the dictionaries when computing a similarity score between a query and document. It's further processed depending on what raw tf moderation method is chosen. `for_query` also updates a set of relevant document IDs for each query to reduce the search space and runtime of the retrieval model. It then proceeds to call a function corresponding to the desired weighting scheme, or model to use.

Binary model

The binary retrieval model calls `get_relevant_tf_wds` to obtain a dictionary of relevant documents, with the criteria of having any of the query terms present in them. All the query tfs are summed together for each document, and the top most relevant documents are returned as the ones with the most occurrences of all the query terms.

Vector space model - term frequency

This model also identifies relevant documents as having any of the query terms present in it, like the binary model. But this vector space model represents the query and documents as vectors of tfs in the query or document. The query vector is compared to each document vector and assigned a similarity score via the following cosine function

$$\cos(\overrightarrow{q}, \overrightarrow{d}) = \frac{\sum_{i=1}^{n} q_i d_i}{\sqrt{\sum_{j=1}^{m} d_j^2}}$$

The numerator multiplies together that occur in both the query and document, but the denominator sums the square of all terms in the document, including ones not in the query. Using a cosine function gives a normalising effect, so high occurrences of a few, potentially, uninformative words like "the" won't stagnate the similarity between potentially relevant documents. The top most relevant documents are ranked as the ones with the highest similarity scores are returned.

Vector space model - term frequency $\times$ inverted document frequency

Similar to the term frequency approach, but takes into consideration the fact that terms with high frequencies in the collection overall are less informative and so, are weighed less, but the opposite is also true. `get_idfs` is called during instantiation to obtain a dictionary mapping query terms to their inverse document frequencies (idfs) by the following equation

$$idf_{w,D} = \log \frac{|D|}{df_w}$$

where $|D|$ is the number of documents in the collection (a constant) and df_w is the document frequency, the number of documents a term w occurs in, in the collection. tfs are then multiplied by their respective idfs to obtain a final weighting vector to compare and rank relevant documents.

Jiacong Wu

Results

* -s = stopping list, -p = stemming

No raw term frequency moderation

	no -s or -p	only -s	only -p	both -s and -p
binary	Rel_Retr: 21 Precision: 0.03 Recall: 0.03 F-measure: 0.03	Rel_Retr: 80 Precision: 0.12 Recall: 0.10 F-measure: 0.11	Rel_Retr: 24 Precision: 0.04 Recall: 0.03 F-measure: 0.03	Rel_Retr: 73 Precision: 0.11 Recall: 0.09 F-measure: 0.10
tf	Rel_Retr: 49 Precision: 0.08 Recall: 0.06 F-measure: 0.07	Rel_Retr: 106 Precision: 0.17 Recall: 0.13 F-measure: 0.15	Rel_Retr: 73 Precision: 0.11 Recall: 0.09 F-measure: 0.10	Rel_Retr: 122 Precision: 0.19 Recall: 0.15 F-measure: 0.17
tfidf	Rel_Retr: 132 Precision: 0.21 Recall: 0.17 F-measure: 0.18	Rel_Retr: 140 Precision: 0.22 Recall: 0.18 F-measure: 0.19	Rel_Retr: 166 Precision: 0.26 Recall: 0.21 F-measure: 0.23	Rel_Retr: 172 Precision: 0.27 Recall: 0.22 F-measure: 0.24

Binary model performance is drastically improved with -s than -p, which makes sense since we rank documents by total query frequency, “non-content” words which are quite common, will contribute a lot to the total frequency to bloat up the score. The tfidf model seems to benefit from using both -s or -p, but even more significant with -p. Without -s or -p, tfidf significantly outperforms both binary and tf models, showing how informative and useful, less occurring words in the collection, are. tf benefited the most from using -s, compared with tfidf, suggesting a pure vector space model is not sufficient to reduce the effect of non-content words.

Log of raw term frequency

	no -s or -p	only -s	only -p	both -s and -p
binary	Rel_Retr: 55 Precision: 0.09 Recall: 0.07 F-measure: 0.08	Rel_Retr: 102 Precision: 0.16 Recall: 0.13 F-measure: 0.14	Rel_Retr: 65 Precision: 0.10 Recall: 0.08 F-measure: 0.09	Rel_Retr: 115 Precision: 0.18 Recall: 0.14 F-measure: 0.16
tf	Rel_Retr: 71 Precision: 0.11 Recall: 0.09 F-measure: 0.10	Rel_Retr: 112 Precision: 0.17 Recall: 0.14 F-measure: 0.16	Rel_Retr: 100 Precision: 0.16 Recall: 0.13 F-measure: 0.14	Rel_Retr: 136 Precision: 0.21 Recall: 0.17 F-measure: 0.19
tfidf	Rel_Retr: 128 Precision: 0.20 Recall: 0.16 F-measure: 0.18	Rel_Retr: 139 Precision: 0.22 Recall: 0.17 F-measure: 0.19	Rel_Retr: 171 Precision: 0.27 Recall: 0.21 F-measure: 0.24	Rel_Retr: 176 Precision: 0.28 Recall: 0.22 F-measure: 0.25

Further moderating the raw tf count in the `get_tfs` function, by replacing the count with $1 + \log(tf_{wd})$, further reduces the stagnating effect of non-content terms in documents. This has shown to be quite effective in all three approaches, with the binary model receiving the most benefits on average across different configurations, which is to be expected since it’s not receiving the same benefits, as with tf and tfidf, from the cosine function which already moderates this.