Jiacong Wu

COM3110 Document Retrieval Assignment Report

Implementation description

Below are brief descriptions of how the different retrieval models were implemented in the `Retriever` class. Before any of the retrieval functions are called in `for_query`, the query is preprocessed to remove all terms which don't occur in the index, so we won't have to check for existence of a term in the index everytime we want to access it. The query is then transformed into a dictionary mapping query terms to its frequency in the query, which will be helpful for the vector space retrieval models, when we want to represent the query as a vector. From Python 3.6 onwards, dictionary key insertion order is retained, so we won't have to use any specialised dictionary classes.

Binary model

The binary retrieval model calls a function `get_tfs` which only looks at documents with any of the query terms present, this can be thought of as a disjunctive query of all the search terms. `get_tfs` returns a dictionary mapping document ids to query terms to their frequencies. All the query term frequencies 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 term frequencies in the query or document. The query vector is compared to each document vector and assigned a similarity score via a cosine function

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

The function has a normalising effect, so large term frequency values of, say 1 term, won't have too much of an impact on the similarity score. The top most relevant document being the ones with the highest similarity scores are returned.

Vector space model - term frequency x inverted document frequency

Mostly the same implementation as the term frequency approach, but takes into consideration the fact that terms with high frequencies in the collection overall are less helpful and so they are weighted less, but the opposite is also true. We call the `get_idfs` function 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 the term w occurs in, in the collection. The term frequencies are then multiplied by their respective idfs to obtain a final term weighting to rank the relevant documents based off.

Jiacong Wu

Results

* -s = stopping list, -p = stemming

No raw frequency moderation

	binary	tf	tfidf
no -s or -p	Rel_Retr: 21 F-measure: 0.03	Rel_Retr: 62 F-measure: 0.09	Rel_Retr: 120 F-measure: 0.17
only -s	Rel_Retr: 80 F-measure: 0.11	Rel_Retr: 87 F-measure: 0.12	Rel_Retr: 119 F-measure: 0.17
only -p	Rel_Retr: 24 F-measure: 0.03	Rel_Retr: 77 F-measure: 0.11	Rel_Retr: 129 F-measure: 0.18
both -s and -p	Rel_Retr: 73 F-measure: 0.10	Rel_Retr: 102 F-measure: 0.14	Rel_Retr: 121 F-measure: 0.17

binary performance with a stopping list is better than with stemming, 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, and hence bloat up the score. tf performance with a stopping list although provided better performance than without one. wasn’t as big of a gap as with binary with one, which is expected because of the normalising effect of the cosine function, mitigating the effect of high frequency terms has on the similarity score. tfidf approach outperformed previous approaches and gave similar performance across the different configurations, however using stemming only, provided slightly better performance.

Log of raw frequency moderation

	binary	tf	tfidf
no -s or -p	Rel_Retr: 55 F-measure: 0.08	Rel_Retr: 73 F-measure: 0.10	Rel_Retr: 123 F-measure: 0.17
only -s	Rel_Retr: 102 F-measure: 0.14	Rel_Retr: 92 F-measure: 0.13	Rel_Retr: 122 F-measure: 0.17
only -p	Rel_Retr: 65 F-measure: 0.09	Rel_Retr: 94 F-measure: 0.13	Rel_Retr: 126 F-measure: 0.18
both -s and -p	Rel_Retr: 115 F-measure: 0.16	Rel_Retr: 113 F-measure: 0.16	Rel_Retr: 128 F-measure: 0.18

Further moderating the raw term frequency count in the `get_tfs` function, by replacing it with $1 + \log(tf_{wd})$ has shown to be quite effective in all three approaches, with the binary approach receiving the most benefits and tfidf the least.