Overview

I have implemented an information retriever which uses Binary, Term frequency (TF) and TFIDF weighting schemes.

The function **filter_documents(query)->filtered_doc_ids** returns a list of document IDs which contain at least one of the words in the query. This negates the need to include documents which would just return 0 as a similarity score in **calculate_cosine_distance(filtered_doc_ids, query_vector)** (mentioned later on), improving efficiency.

More on efficiency, the document vectors for each of the respected weights are calculated upon instantiation, so they are calculated only once in a preprocessing step. The function **weighted_document_vectors()->document_vectors** uses conditions to create a vector weighted on the method set in the program's arguments. For TFIDF where TF is needed before IDF, on the first run of the function a TF vector is returned and assigned to a class variable. The second run uses that class variable to finally create an index containing the TFIDF weights. TFIDF makes use of **create_df_index()** and **create_idf_index()**. These indexes are preprocessed upon instantiation when TFIDF is selected, and aids in the calculation of TFIDF.

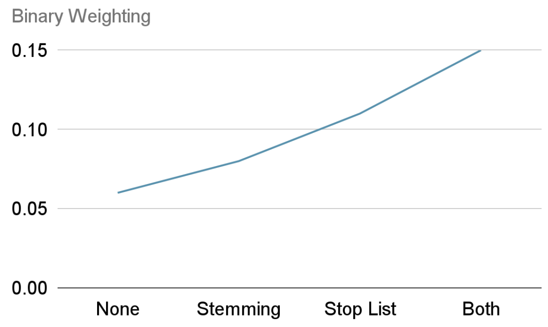
Individual query vector functions create standardised dictionaries which are weighted accordingly. I couldn't find a way to create one standardised function for this so there are three.

calculate_cosine_distance(filtered_doc_ids, query_vector) takes the query and filtered doc ids, returning a ranked list of document ids to then be returned back to the IR_engine. This one function works for every weighting scheme.

Results

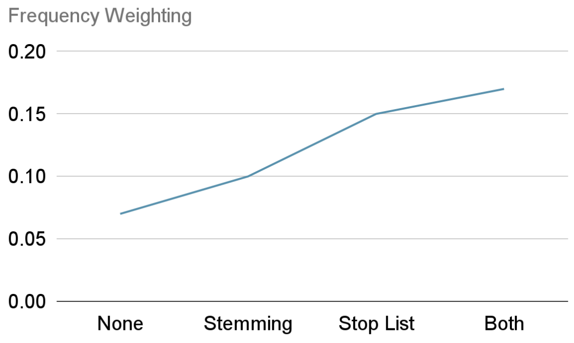
Binary weighting (% assume none as baseline)

	None	Stemming	Stop List	Both
Precision	0.07	0.09	0.13	0.16
Recall	0.06	0.08	0.10	0.13
F-Measure	0.06	0.08 **(+33.3%)**	0.11 **(+83.3%)**	0.15 **(+150%)**



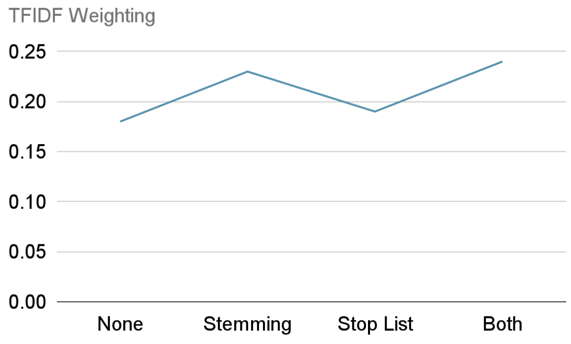
TF Weighting (% assume none as baseline)

	None	Stemming	Stop List	Both
Precision	0.08	0.11	0.17	0.19
Recall	0.06	0.09	0.13	0.15
F-Measure	0.07	0.10 **(+42.9%)**	0.15 **(+114.3%)**	0.17 **(+142.9%)**



TFIDF Weighting (% assume none as baseline)

	None	Stemming	Stop List	Both
Precision	0.21	0.26	0.22	0.27
Recall	0.17	0.21	0.18	0.22
F-Measure	0.18	0.23 **(+27.8%)**	0.19 **(+5.6%)**	0.24 **(+33.3%)**



Summary

Percentages are difference over the previous results.

FMessure	**None**	**Stemming**	**Stop List**	**Both**
Binary	0.06	0.08	0.11	0.15
FMeasure TF	0.07 **(+16.7%)**	0.10 **(+25%)**	0.15 **(+36.4%)**	0.17 **(+13.3%)**
FMeasure TFIDF	0.18 **(+157.1%)**	0.23 **(+130%)**	0.19 **(+26.7%)**	0.24 **(+41.6%)**

Processing Time for all queries. (average of 3)

FMeasure	**None**	**Stemming**	**Stop List**	**Both**
Binary	4.68(s)	3.72(s)	2.66(s)	2.42(s)
TF	4.54(s)	3.73(s)	2.66(s)	2.14(s)
TFIDF	6.58(s)	4.74(s)	4.62(s)	3.38(s)

Evaluation of Results

Some interesting results are shown. Binary weighting is shown to be largely ineffective without the use of any term manipulation. The use of a stop list improved f-measure by 83.3% over just binary weighting alone. Stemming, albeit an improvement, is much less effective on its own compared to stop list, only improving f-measure by 33.3%. There is a linear relationship in f-measure from the addition of stemming to stoplist to both, leaving both with a 150% increase.

TF weightings results are comparable to Binary weighting, with small improvements. TF weighting alone only has a 16.7% increase in performance over Binary in the same category, with stemming and stop list also having marginal increases. This shows that TF is almost equally as ineffective as Binary when term manipulation techniques are not applied. Across the board there is on average a 22.9% increase over Binary weighting, and a relatively linear increase is seen here too over the 4 categories.

TFIDF has a 157.1% increase in f-measure alone over the previous TF. This clearly shows that taking document frequency into account results in an increase in the effectiveness of the information retrieval system. Interestingly, this is the first weighting scheme that saw the stop list being less effective than stemming when applied on their own. I feel this could be because common words affect this weighting scheme less, as the frequency over documents is taken into account. This metric can be skewed if different variants of words are not taken into account properly, which stemming seeks to resolve. This results in stemming having a greater effect over the corpus than stop list alone. All term manipulation schemes saw an increase over the previous though, so I conclude that this weighting scheme is superior over the previous and thus overall.

Time Efficiency

To make my IR system more efficient, I calculated the document vector upon instantiation of the Retrieve class. I also found time savings in only including present terms in the vectors. Term manipulation schemes reduced processing time, with the highest processing time being in the weighting schemes alone. Unsurprisingly, stop list reduces the time by nearly half on all, due to it reducing the number of terms processed significantly. TFIDF saw the highest processing time. This is because multiple indexes need to be created, including a TF index. Notably, TFIDF does not have the same effect with stop list as the other two. This is because of the aforementioned indexes that produce an overhead.