TTS - Practical 2 - Image Search
Ruaridh Thomson - s0786036

Implementation

For both word overlap and tf.idf the documents and queries are stored in two types of lists: one storing each document/query as a list element and one storing each document/query as a list of the individual words (a list of lists). To calculate word overlap we iterate over each query, then for each document we count how many of each query word appears in each document. The count is stored alongside the query number and document number (almost identically to the search results format) and written out. To calculate the tf.idf we similarly iterate over each query, then for each document we calculate the tf.idf for each word in the query and sum these together - again storing them in the output format and writing to file. An expensive (efficiency) step was calculating the number of docs containing each word in a query, having to calculate:

(num queries)(num documents)*(num words in query)*(num documents containing word)*

However, we know that if tf_wd=0 then the weight for that particular word will be 0. Therefore we only check the number of documents containing the query word if tf_wd is not 0.

TF.IDF

We find the default algorithm to perform as expected. Attempts to increase the performance (Avg. Precision) of the algorithm are shown in Table 1.

Initially, stemming was employed via a python stemming implementation [2], and was found to perform significantly worse than the default. Both documents and queries were stemmed, though the results suggest that stemming alone does not increase the performance of the tf.idf algorithm. Beyond the assignment it would be worth combining stemming with a database such as WordNet (word meanings and lexical relationships), and when applied to both documents and queries we would expect an increase over the default tf.idf.

Increasing the value of k also resulted in a slightly reduced performance, however reducing the value of k to 1 showed a slight increase in performance. Setting k to 0.8 showed the same performance as k=1 though with a decreased R-Precision, and we expect tuning k around 1 would slightly increase the performance further. From the table we observe that changing the base of the log used when calculating the tf.idf weight of a word has no effect on the overall performance or R-Precision. We expected the log to have an effect for rare words leading the results to suggest that there are little to no rare words or that they did not affect the overall tf.idf weighting for words in a query.

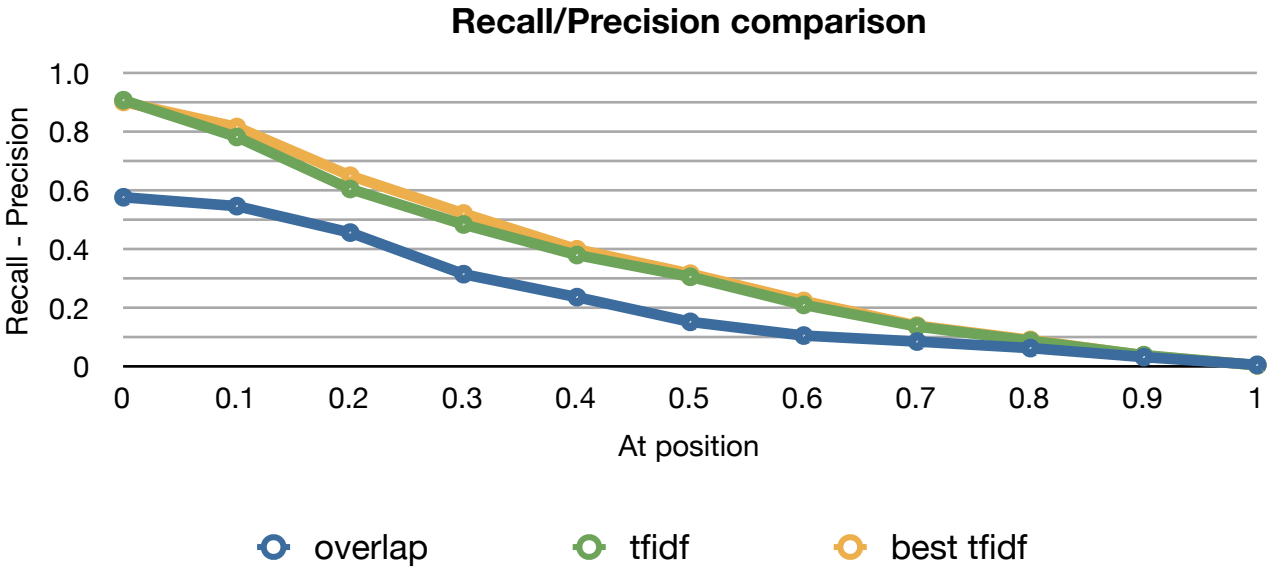
Performance is consistent over the altered algorithms, where expected that tuning k would improve performance and the results show this. We expected stemming to increase the performance, though we were presented with results that showed we found more documents that were not relevant to the query; each document was effectively generalised and returned.

Table 1

Algorithm	Avg. Precision	R-Precision	Time*
Default**	0.3353	0.3697	1m21s
Default + Stemming documents & queries	0.1888	0.1973	2m19s
Default & k=4	0.2953	0.3247	1m26s
Default & k=1	0.3479	0.3910	1m32s
Default & k=1 & $\log_{10}$	0.3479	0.3910	1m44s

Algorithm	Avg. Precision	R-Precision	Time*
Default & k=1 & $\log_2$	0.3479	0.3910	1m34s
Default & k=0.8	0.3479	0.3811	2m3s

**tests were performed on the same machine, minutes and seconds*
***no alterations to the tf.idf calculation, k=2, $\log_e$*



References

[1] TTS Slides
[2] Porter2, python stemming implementations, http://pypi.python.org/pypi/stemming/1.0, 20th October 2011.