Document Retrieval Report

1. Description of the implementation of my_retriever.py file

First of all, I change the content form of the dictionary 'index' to another dictionary 'document': {document id: {term: count}} for initialisation in the _init_ scope in order to derive the different terms in a certain document easily and quickly. After that, in the forQuery function, there are three parts divided to compute the similarity through different term weighting schemes as the following.

If the termWeighting variable is 'binary', the sum of square of each query, q^2, is equal to the query length which means how many terms in this query appear because the binary term weighting only concerns about whether a term appears or not (1 or 0). Likewise, the sum of square of each document, d^2, is also equal to its length. Moreover, in the document iteration, I have to check every term which simultaneously appears in the query and specific documents for the q multiplying d.

In addition, in terms of 'tr' term weighting, due to the fact that I have initially built the document dictionary, it is quite straightforward and same procedure as above scheme to derive those three values (q^2, d^2, and q*d). It is noticeable that I must use the count of each term to calculate them instead of the length.

For the 'tfidf' term weighting, I sequentially gain the D, tf, df and tfidf, and then compute the squares of d, q and qd respectively. In particular, I have to create a new dictionary to store the result of idf derived from the index dictionary for calculating q^2 when the term also appears in the query; the idf, otherwise, is assigned to zero.

Finally, the key of 'result' dictionary is document id and value of it is similarity. Before the document id list is returned, it can be sorted by the value of the result in descending order.

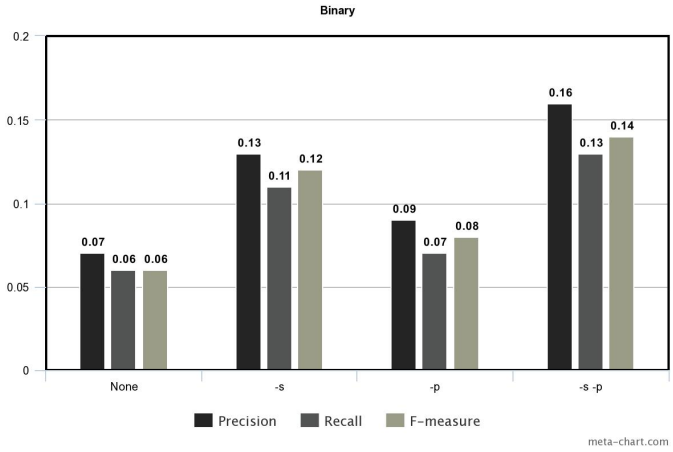
2. The performance results

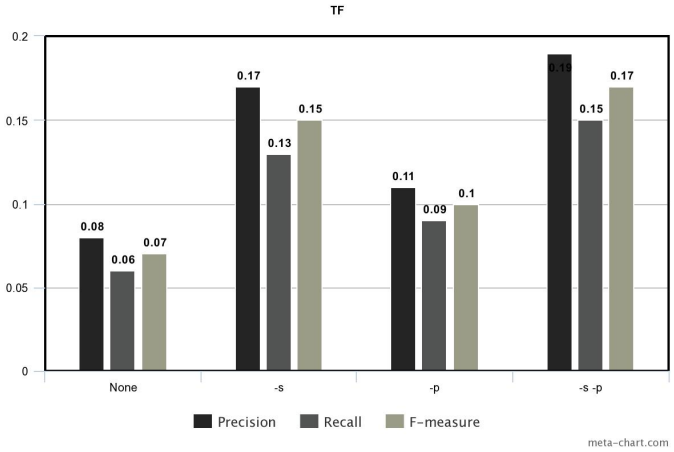
- The execution time of python code under different configurations which are different input file

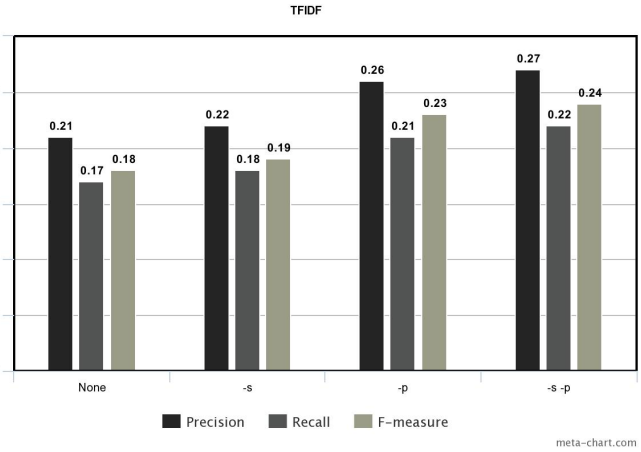
Execution time under different configuration (second)

	none	-s	-p	-s -p
Binary	1.08	0.75	1.06	0.72
TF	4.14	2.63	3.95	2.48
TFIDF	13.39	8.23	12.63	7.70

- The three different values including precise, recall, and f-measure with different configurations derived through ‘gold standard’







3. Conclusion

According to several performance results, those data which have been processed have quite fewer execution time than the raw data. It can be seen that the execution time of the data dealt with stop list is much fewer near to that with both stop list and stemming; however, the execution time of the data only with stemming does not decrease obviously. That is, using the data with stop list is beneficial and efficient. Furthermore, although information retrieval through ‘tfidf’ term weighting scheme takes the most execution time among those three schemes, it has the highest precision and recall. It may be a kind of trade-off between more accurate result and less execution time.