The task was to complete the document retriever class that is used by the information retrieval algorithm. The required weighting schemes were such that relevant documents could be returned ordered by Term Frequency (TF), Term Frequency Inverse Document Frequency (TFIDF) and Binary.

The retriever class has fully working methods for all the required weighting schemes, the methods have been created with attention to maximum code efficiency as not only is precision important in document retrieval but results must be returned in a timely manner as to not frustrate the user.

Binary Scheme:

In Binary configuration, the frequency of how often a term appears in the query or document is irrelevant, all that is considered is how many of the query terms appear in each document. A document that contains many of the query terms is more similar. So when finding candidate documents, using the *getCandidateDocuments* method, a list is generated that contains document ID's which contain the query term. The list can contain for example the same document ID twice, this is because that document contains two of the query terms.

The cosine similarity is then calculated for each document. The document length is calculated by a separate method called *getDocumentLengths* when the first query is received by the IR engine. Therefore the summation of the term squares is not required in the cosine similarity equation' as this is calculated within the *getDocumentLengt*hs method and the value is stored within a dictionary called *docLengths*.

The calculated cosine similarity and document ID are added to a dictionary called *similarityScores* which is then sorted with the ranked document ID's being returned to the IR engine.

Term Frequency Scheme:

In TF configuration the variables required are the query term occurrences in each candidate document and the document lengths. Document lengths are determined using the same method mentioned in Binary. The term occurrences are found using the *getCandidateDocuments* method a dictionary called *candidates* is generated which contains both document ID's but also term occurrences for each document. Iterating through each term in the query and accessing the candidate documents for this term, the TF is equal to:

$$Term\ Frequency = Term\ Frequency\ in\ Query \times Term\ Frequency\ in\ Document$$

Term frequency is then summed for each document and cosine similarity can be calculated.

Term Frequency, Inverse Document Frequency Scheme:

In TFIDF configuration, the IDF value for every term in the index is calculated by the *getIDFValues* method when the first query is sent by the IR engine in TFIDF scheme. With IDF being:

$$IDF = \log\left(\frac{Num\ of\ documents\ in\ corpus = 3204}{Num\ of\ documents\ that\ contain\ term}\right)$$

The IDF values are then stored in a dictionary called *IDFValues*. The process of calculating cosine similarity for each candidate document is the same as the TF method other than TF is multiplied by the IDF for the term. This has the effect of weighting down terms that are common and increasing the weight of terms that are rare. So documents that contain rare query terms are ranked as more similar.

Evaluation of the Performance of System over CACM Test Collection

Computer Specs: Intel Core i7-8700 CPU @ 3.20GHz. 15.8GB Usable Installed RAM. Windows 10.

Weighting Scheme	Configuration S = Stop words P = Stemming	Relevant Docs Returned	Precision	Recall	F-Measure	Time to Run (Seconds)
Binary	**N/A**	21	0.03	0.03	0.03	0.17
	P	25	0.04	0.03	0.03	0.18
	S	48	0.07	0.06	0.07	0.05
	P and S	60	0.09	0.08	0.08	0.07
TF	**N/A**	49	0.08	0.06	0.07	0.19
	P	72	0.11	0.09	0.10	0.21
	S	106	0.17	0.13	0.15	0.06
	P and S	123	0.19	0.15	0.17	0.08
TFIDF	**N/A**	114	0.18	0.14	0.16	0.21
	P	146	0.23	0.18	0.20	0.24
	S	132	0.21	0.17	0.18	0.06
	P and S	170	0.27	0.21	0.24	0.08

For all methods when the configuration of stopwords is used whether by itself or in combination with stemming the run time is dramatically reduced. This is because the number of terms in the index is reduced from 10769 to 6940, this combined with reduced terms in the query means there are far less terms to iterate through and so faster run times. For both Binary and TF schemes using the stopword configuration also benefits precision more than the stemming configuration.

Stopwords add double the improvement to precision than stemming does. Stemming improves precision by 0.01 and 0.03 for binary and TF, whereas stopwords improve precision by 0.04 and 0.09 respectively. This is not the case for TFIDF as TFIDF already accounts for common words which would be removed by stopwords. Therefore, in TFIDF stopwords are used to reduced run times rather than improve precision.

With both stopwords and stemming configured, the TFIDF scheme has far better characteristics than TF and Binary. TFIDF has an improvement of 0.08 in precision over TF and 0.18 over Binary with similar increases for both Recall and F-Measure but most importantly there is no real change in run time. This improved precision is because TFIDF takes into account how important a word is in the corpus, neither of the other schemes consider the importance of words relative to the corpus as a whole.

TFIDF is the optimum scheme from those considered in this assignment. It offers better precision and returns more relevant documents for all configurations whilst maintaining similar run times as other schemes.

During research delta TFIDF was found whereby terms are also classified by subject, so the similarity of a document also includes whether the subject of the document is similar to the subject of the query. This enhanced method could further improve precision but with the effect of increasing run times. It would be interesting to see whether the increased precision of delta TFIDF would outweigh the increased run times and whether delta TFIDF would out perform the basic TFIDF model.