Assignment: Document Retrieval

The code execution is as follows:

self.compute_cosine_score(query):

This function first retrieves all unique words for each query.

Computes a *term_document_map* which is a 2d matrix of all unique words * number of documents

self.compute_term_weights (term, document_frequency, row):

It calculates the vector space model for all 3 cases:

a. *binary*: It stores the values as 1 if word is present in document, else 0
b. *tf*: It stores the frequency of that word in the document
c. *tfidf*: It stores the tf.idf value by calculating the cosine value for each and every term in the query. A *term_idf_map* is a dictionary which maintains term as keys and *idf* as value which is used to calculate *tf.idf*

self.compute_query_vector(self, term, query):

It calculates the query vector for all the terms present in query.

a. *binary*: It appends the value 1 to the list if word is present in query, else 0
b. *tf*: It appends the frequency of that word in the query to the list
c. *tfidf*: It fetches the *tf.idf* value stored in *term_idf_map* and appends it to the list

By iterating over the term_document_map the numerator and denominator are calculated and the score is computed using the following formula. These scores are computed for all candidate documents and stored in dictionary *docid_score.*

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

self.fetch_top_matching_docs(docid_score):

It sorts the dictionary based on descending order of the computed scores for all relevant documents. It then retrieves the top 10 ranked scores and returns the document id list for each query.

Following are the results obtained for all binary, tf and tfidf approach:

1. **Binary**

BINARY	No Stoplist and Stemming	Stoplist	Stemming	With Stoplist and Stemming
Relevant Retrieved document	64	79	72	92
Precision	0.10	0.12	0.11	0.14
Recall	0.08	0.10	0.09	0.12
F-measure	0.09	0.11	0.10	0.13

It is observed that the number of relevant documents retrieved for binary approach is less compare toother two approaches. The reason being it results in selecting the frequent and unnecessary words. It doesn't capture those words that occur frequently in document which would rather be more relevant. Precision recall and f-measure are highest for configuration which handles stemming and removes stop words.

2. Term Frequency

TF	No Stoplist and Stemming	Stoplist	Stemming	With Stoplist and Stemming
Relevant Retrieved document	61	90	74	107
Precision	0.10	0.14	0.12	0.17
Recall	0.08	0.11	0.09	0.13
F-measure	0.08	0.13	0.10	0.15

It is observed that the term frequency approach has performed relatively better as compared to the binary approach. This approach takes into consideration the frequency of the word in the document. The drawback here is if the frequency of the common words is higher, the weights of these words would be higher. Hence, the relevant documents won't be fetched. Precision recall and f-measure are highest for the configuration where stemming is handled and stop words are removed.

3. TFIDF

TFIDF	No Stoplist and Stemming	Stoplist	Stemming	With Stoplist and Stemming
Relevant Retrieved document	95	124	108	119
Precision	0.15	0.19	0.17	0.19
Recall	0.12	0.16	0.14	0.15
F-measure	0.13	0.17	0.15	0.17

The TFIDF approach works best amongst the other two approaches. The reason being the frequently occurring words are taken care by the logarithm which ensures they aren’t heavily weighted. In other words, common words will be weighted lower. As seen in the table above, the configuration where stop words are ignores and the other one where stop words are ignored and stemming is handled seems to perform well.

To conclude, since the f-measure for all configurations in TFIDF approach is higher than the other two approaches we could say that TFIDF is the best approach of the lot.