COM3110 Text Processing
Assignment: Document Retrieval
Catalin Mares

Implementation

For each of the term weightings I have used the cosine similarity formula provided in the lectures. I've ignored the query vector size calculation due to the fact that the query will remain constant for each article and it would not impact the score obtained by each article. This will make computing the score of each article for each term weighting easier. The formula used is shown under Figure 1. After computing the cosine similarity scores between each article and the query the program will return the top 10 articles that score highest.

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

Figure 1

- Binary term weighting

The way this has been implemented was by computing the set intersection between the set of terms in each article and the set of terms in the query assigning the value 1 for each of the elements that were found in the intersection of the two sets specified. As a result of that if a term existed both in the query and document that term will be assigned the weight 1 or otherwise if the term exists only in the document but not in the query the weight of the term will be 0. Doing this for every article will compute a cosine similarity score for each one of the articles in the library.

The last step would be to compute the vector size for each of the vectors constructed for each article and hence divide the article score by its vector magnitude in order to normalise score received. This is done because a term may appear in an article more often than in another article due to the simple fact that an article contains many more terms than the article we are comparing it to.

- TF weighting

The way this has been implemented is similar to the binary term weighting however rather than using the term weighting of 1 for the terms that are present both in the article and the query the weight assigned to the term is the (frequency of the term in the article * frequency of the term in the query). By doing this the terms which appear more often in an article will be considered to be more important than other less frequent words.

- TFIDF weighting

Lastly the TFIDF term weighting is similar to TF term weighting but with the addition of an inverse document frequency element that is being taken into consideration for each of the terms. This IDF is computed for each of the terms by calculating the inverse of (how many articles the term appears in / how many articles are in the collection). This will ensure that the fact which states that rarer terms are more important than common terms is taken into consideration as it will adjust the scores of the terms accordingly.

The remaining of the calculation will be pretty much the same as the one for TF apart from applying the IDF weight to the TF weight by multiplying the two together to produce a TFIDF term weight.

Results analysis

In this assignment I have managed to successfully implement all the three different term weighting algorithms for document retrieval. Figure 2 will show the F-Scores obtained for each of the 12 configurations.

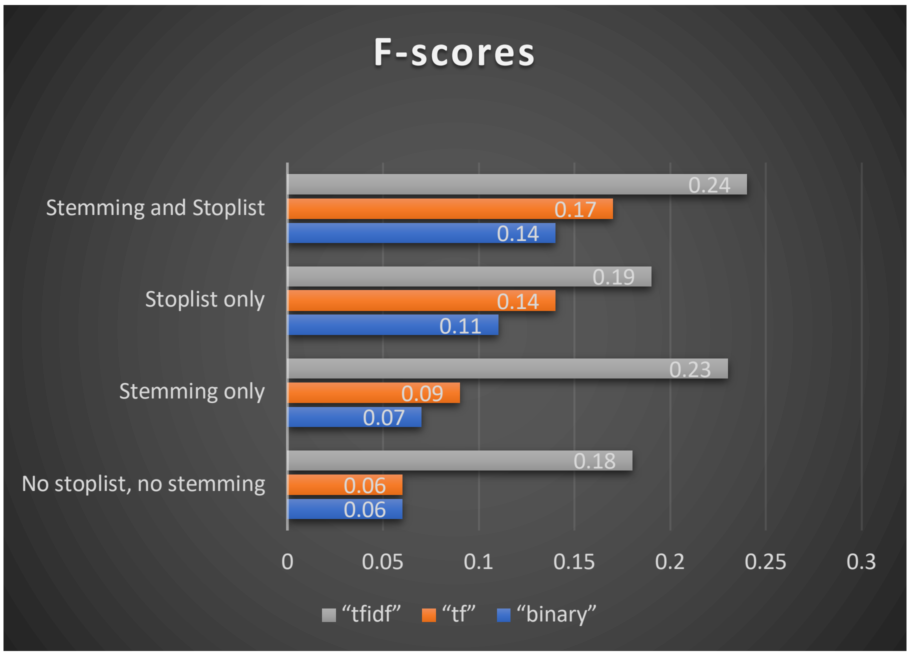

Figure 2

From these results we can draw the conclusion that the "binary" term weighting is the worst out of the three and that "tfidf" is the best due to their computed F-scores.

This is to be expected as the "binary" term weighting is the simplest method as it assigns 0 or 1 to each term depending on whether or not it exists in the query and in the document.

On the other hand, TF will not only assign a binary weight to each term but will assign the value of the frequency of that term in the document meaning that terms which appear more frequently in a document can have a higher impact upon the final cosine similarity score of that document.

Finally, I have concluded that the TFIDF term weighting is the best due to the fact that rather that looking just at the amount of times a specific term appears in an article it will also take into consideration how important that word is within the collection of documents which are being queried. This will adjust the weights of the words which appear more rarely within the collection by considering them more important. An example of words that appear many times in documents are ("an", "the", "is", "it", etc.). These words are evidently not very useful in identifying relevant documents as perhaps other words such as "algorithm" for example. By multiplying the TF by the IDF score of each term this importance factor will be taken into consideration and the weights of the terms will be adjusted accordingly.