Manuel Worlali Ziwu - 180128251

COM 6115: Text Processing

Assignment 1: Document Retrieval Report

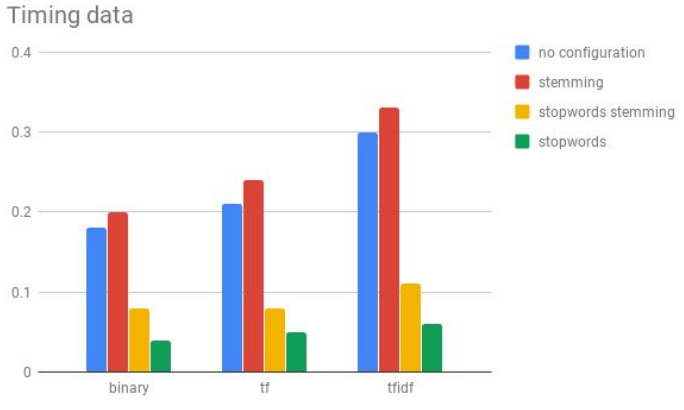
Implementation

To improve system performance, values that would be reused across multiple queries were computed in the constructor of the **Retrieve** class and assigned as class variables to prevent them from being computed for every query. There were also as many implementations of list and dictionary comprehension to reduce code length and improve overall computational speed.

Code Implementation

- List of all document IDs - A double for-loop in a list comprehension to return all document ids from a term. The list is then transformed into a set to get rid of all duplicates in it.
- $\sum_{i}^{n} d_i^2$ - A nested for-loop is used to obtain the count values of a term in a document where needed (TF and TF-IDF) or perform a basic increment(in the case of binary), and based on the weighting scheme used the values are computed accordingly. A dictionary containing as keys document IDs and as values, the weighting scheme values stores the computed value for every term in the dictionary. To improve the efficiency of the program, this operation was done in the constructor to prevent it from being computed multiple times where not needed.
- $\sum_{i}^{n} q_i^2$ - For computing this value for the binary weighting, it was simply the number of terms in the query, this value wasn't needed for the TF weighting scheme. A list comprehension was used to compute this value for the TF-IDF weighting scheme, after which the list was passed to a sum function to get the final value.
- Idf_w - The IDF values for every term in the collection was computed in the constructor and assigned as a class variable. It was computed using dictionary comprehension to go over the terms in the index and the **log** function in Python's default **math** library was used to compute the final value. The number of documents in the collection was obtained by finding the size of the entire document ID array.
- $\sum_{i}^{n} q_i d_i$ - A nested for-loop was used to obtain or compute the value for each document that contained the terms of the query. These values are kept in a dictionary that has the document IDs as its keys and the dot product of the query value and document as its values.
- Cosine similarity - A dictionary comprehension is used to finally obtain a key-value pairs of the document IDs with their cosine similarities computed using the results from the dictionary of the dot product and the dictionary that contains the values for $\sum_{i}^{n} d_i^2$ as well as the result for computation for $\sum_{i}^{n} q_i^2$.
- Ranking documents - Using Python's **sorted** function, a list of the document IDs from the resulting dictionary were returned in descending order by setting the **reverse** parameter to True and performing the sort by the values in the dictionary(i.e. Cosine similarity values).

Result Analysis



The program was executed on a laptop with an 8GB RAM and an Intel Core i5 @ 1.60GHz.

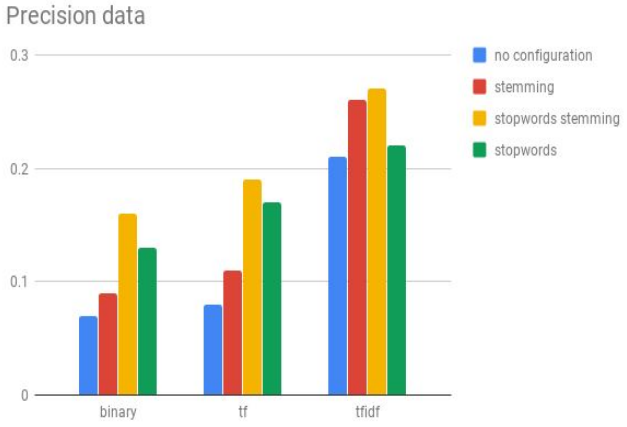
It can be observed that, in all of the weighting schemes used, performing a document retrieval with stoplists yields the fastest results.

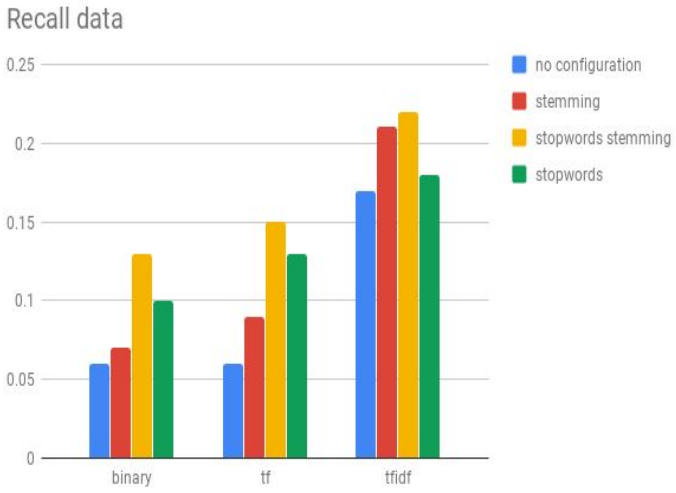
This can be attributed to the fact that, stop words usually amount to majority of terms in the collection.

Using an index and query with stemming could take a while because the term frequency increases significantly for every term.

Using no configuration at all also takes as long as a configuration with stemming because no preprocessing has been performed on the data.

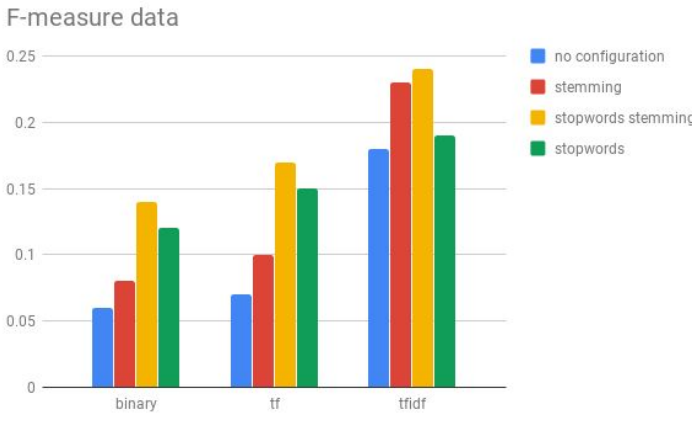
Precision is the proportion of retrieved documents that are relevant.
Though under all weighting schemes, the value of precision is generally low, the configuration that employs the use of stopwords and stemming appears to perform better than all the other configuration methods.
Retrieval of documents with a configuration that involves stoplists gives a relatively high precision value in TF and binary weighting schemes, but in TF-IDF weighting scheme, the retrieval of documents with the stemming configuration appears to outperform that of stoplist. The use of stemming for TF-IDF weighting scheme reduces the collection frequency of a term and provides a good IDF value that assists in retrieving documents closely matched to the query.





Recall is the proportion of relevant documents retrieved.
It can also be observed that using stopwords and stemming configuration has bigger influence on the recall value that all other configurations.
With the TF and binary, it can be observed that using the the stoplist configuration has a better impact for the recall value than the 2 other configurations. However, with the TF-IDF weighting scheme, using the stemming configuration provides a better recall value done with the stoplist configuration.
This can be attributed to the fact that, words in the stoplist though included with just the stemming configuration have a small impact on the document retrieval process because of their IDF values that make them quite insignificant in the collection as a whole as compared to the individual documents.

F-measure is the combination of both the the precision and recall values. It gives equal weight to both values and penalizes low performance in one value more than the arithmetic mean.
It can be observed that there is a close relation between the F-measure values and the Recall values.
The graph shows us that overall, the TF-IDF weighting scheme gives the best retrieval of documents with the stoplist and stemmig configurations.



Conclusion

Based on the graphs displayed, it can be concluded that the TF-IDF is the best weighting scheme when it comes to document retrieval as it outperforms both TF and binary weighting schemes in all configurations. It must also be noted that, using the retrieval process with both stoplist and stemming produces the best results even with any type of weighting scheme whatsoever. In regards to the TF and binary weighting scheme and besides the stoplist and stemming configurations, we observe that the second best configuration to use is the configuration with stoplist as their method of evaluating the similarity between queries and documents do not consider the collection as a whole but just the document on its own and as such the elimination of stopwords from the query and/or documents as a preprocessing task increases its ability to retrieve documents of relevance with greater precision than the other configurations.