**A Comparative report on performance of TF-IDF and BM25**

Raunak Nandkumar More

*Department of Computer Science*

*University of South Dakota*

**ABSTRACT:**

This report tries to compare performance of term frequency-inverse document frequency (TF-IDF) and best matching 25 (BM25) algorithms for fetching of relevant results given different queries. We attempt to find the similarities and differences between the 2 algorithms. It is clear that BM25 has overall better performance on account of the issues that plague TF-IDF, but a comparative, same query scoring using cosine similarity will give us some estimate of how better BM25 works.

**INTRODUCTION:**

We perform text analysis on Shakespeare’s Macbeth, a tragedy play written by William Shakespeare in the early 17th century. While general Information Retrieval involves querying multiple documents for query results, since in theory the same could be applied for individual sentences and paragraphs an attempt to query and get individual sentences that are relevant is an approach that has been tried. The report aims to compare the similarity scores (that are calculated for each sentence) and rankings of different queries to the documents in the play, using two ranking functions: term frequency-inverse document frequency (TF-IDF) and best matching 25 (BM25). The report also aims to evaluate the performance and effectiveness of the two ranking functions, and to provide some insights and recommendations for future text analysis projects. The terms of reference for this report are what are the TF-IDF and BM25 ranking functions, how are they implemented in Python, and how do they rank the documents in Macbeth based on different queries. The scope of this report is limited to the text of Macbeth, obtained from the NLTK package, and the queries provided by the author. The report does not cover other ranking functions, other texts, or other queries. The methodology of this report consists of four steps: preprocessing the text, creating the TF-IDF and BM25 matrices, computing the similarity scores, and ranking the documents using cosine similarity.

**LITERATURE REVIEW:**

Text analysis is the process of extracting meaningful information from natural language texts, using various techniques and tools. Text analysis can be used for various purposes, such as information retrieval, sentiment analysis, topic modeling, text summarization, and text classification. One of the common tasks in text analysis is to measure the similarity or relevance of a query to a document, or a document to another document. This can be done by using ranking functions, which assign scores to the documents based on certain criteria. Two of the most widely used ranking functions are TF-IDF and BM25.

Since for this experiment we use sentences as smallest unit instead of we also change the formulae to reflect that. TF-IDF is a statistical measure that reflects how important a word is to a document in a collection of documents. In our case it reflects how important a word is to a query in a collection of sentences from Macbeth. It is calculated by multiplying the term frequency (TF), which is the number of times a word appears in a sentence, by the inverse document frequency (IDF), which is the logarithm of the total number of sentences divided by the number of sentences that contain the word., TF-IDF assigns higher scores to words that are more specific and distinctive to a sentence, and lower scores to words that are more common and generic. BM25 is an extension of TF-IDF that incorporates the document length and the query length into the calculation. BM25 is based on the probabilistic retrieval model, which assumes that the relevance of a sentence to a query is proportional to the probability of observing the query terms in the collection of sentences. BM25 uses two parameters, k1 and b, to adjust the term frequency and the document length factors, respectively. These factors are in place to solve 2 main issues that are seen in TF-IDF.1st issue being the frequency spamming that can lead to higher scores for results that have higher frequency of the queried word.2nd issue being the fact that having smaller sentences and longer sentences that have the same number of queried words, there needs to be a parameter that prefer the shorter sentence over the other.k1 and b parameters take care of these issues respectively. BM25 also adds a smoothing factor, epsilon, to avoid zero scores for terms that do not appear in the document. BM25 assigns higher scores to documents that contain more query terms, and lower scores to documents that are too long or too short. Several studies have compared the performance and effectiveness of TF-IDF and BM25 for different text analysis tasks, such as information retrieval, text summarization, and text classification. The results of these studies are not conclusive, as they depend on various factors, such as the type and size of the text collection, the type and length of the queries, the choice of the parameters, and the evaluation metrics. However, some general trends can be observed, such as: BM25 tends to outperform TF-IDF for short and simple queries, while TF-IDF tends to outperform BM25 for long and complex queries; BM25 tends to be more robust and consistent than TF-IDF for different text collections and domains; and BM25 tends to be more sensitive and responsive than TF-IDF to the changes of the parameters and the smoothing factor. Despite the extensive research on TF-IDF and BM25

**METHODOLOGY:**

The data for this research was obtained from the NLTK package, which provides access to various corpora and resources for natural language processing. The data consisted of the text of Macbeth, which is part of Gutenberg corpus. We import necessary libraries as seen below.

A screen shot of a computer

Description automatically generated

We use the gutenberg.raw("shakespeare-macbeth.txt") to get the text from corpus and assign it to a document. We then use document.lower() to convert the document into lower case. We proceed to tokenize the document into 2 different ways, the first being to tokenize into sentences and the other to tokenize into paragraphs. This was an experiment to improve the query results by returning a paragraph instead of a sentence as it is intuitive to get a bit more information in a paragraph over a sentence.

Tokenization into paragraphs was done using the blankline\_tokenize function while nltk.sent\_tokenize was used to tokenize it into sentences. The paragraphs/sentences were then pre-processed, by removing the stopwords, punctuation, and any other irrelevant tokens, using the stopwords and punctuation lists from the NLTK and string packages, respectively. The words were also stemmed, using the SnowballStemmer class from the NLTK package, which is a more advanced version of the PorterStemmer class. The pre-processed paragraphs/sentences were stored in a list of documents, which contained 678 sentences in total. The data was ready for analysis, without any further cleaning or transformation.

The TF-IDF matrix was created using the TfidfVectorizer class from the Scikit-learn package. The TF-IDF matrix was a sparse matrix of shape (678, 3310), where 678 was the number of sentences, and 3310 was the number of unique words in the vocabulary. The TF-IDF matrix was converted to a dense array, using the toarray method, and the IDF vector was extracted, using the idf\_ attribute as seen below:



The BM25 matrix was created using the formula described in the introduction, using the TF matrix, the IDF vector, and the parameters k1, b, and epsilon. The values of the parameters were set to 1.2, 0.75, and 0.25, respectively, based on the literature review and the trial-and-error method. The BM25 matrix was also a dense array of shape (678, 3310). Three queries were provided by the author, to test the similarity and relevance of the documents to the queries, using the TF-IDF and BM25 ranking functions. The queries were:

Query 1: “three witches,magic,death,thunder”

Query 2: “Macbeth knife wound,red blood witch”

Query 3: “Ruthless ambition king prophecies beast unmake fair”

The queries were also pre-processed, by tokenizing, filtering, and stemming the words, using the same methods as the documents. The similarity scores were computed, by transforming the queries into vectors, using the transform method of the TfidfVectorizer object, and calculating the cosine similarity between the query vectors and the document matrices, using the cosine similarity function from the Scikit-learn package. We build the BM25 formula using the tf matrix. The similarity scores were stored in dictionaries, where the keys were the queries, and the values were the arrays of scores. The rankings were computed, by sorting the indices of the scores in descending order, using the sorted function from the Python built-in package. The rankings were also stored in dictionaries, where the keys were the queries, and the values were the lists of indices. The top 5 documents for each query were printed, along with their scores, using the print function from the Python built-in package.

**RESULTS:**

The results of the data analysis showed that the TF-IDF and BM25 methods had some similarities and some differences in ranking the documents. For example, for the query 1, both methods ranked the documents that contained the words “thunder” and “three witches” higher than the others. However, for query 2, the TF-IDF method ranked the document that contained the words “knife”, “wound”, and “blood” higher than the BM25 method, while the BM25 method ranked the document that contained the word “macbeth” higher than the TF-IDF method. This suggests that the BM25 method gives more weight to the term frequency, while the TF-IDF method gives more weight to the inverse document frequency. The results also showed that the performance and effectiveness of the two methods depended on various factors, such as the type and length of the queries, the choice of the parameters, and the evaluation metrics. The research demonstrated the applicability and usefulness of the TF-IDF and BM25 ranking functions for text analysis of literary texts, such as Macbeth.

**CONCLUSION:**

After running many different queries both short and long, it was seen that BM25 generally performed very similar to TD IDF for smaller queries while there was a bit of difference for bigger queries where BM25 gave better results than TD IDF. While for smaller queries both had similar results, however BM25 almost always had better cosine similarity scores. There was a lack of in-depth analysis of performance of both algorithms as we solely relied on cosine similarity scores. While using other metrics like precision, recall, f1 scores would have helped but it involves manual labeling for true results for it to be used which would be out of scope for this project, thus wasn’t used. It was also seen that for both algorithms they didn’t always give relevant results to some queries that returned very short sentences which did not make any sense. A possible solution to it would be to filter small sentence results and could be implemented in the future. The hyperparameters for BM25 were also not experimented with, which could be another topic for optimization. An attempt to retrieve paragraphs over sentences was also made but was not successful in getting better results when compared to sentences.

**REFERENCES:**

1.R Robertson, S. E., & Zaragoza, H. (2009). The probabilistic relevance framework: BM25 and beyond. Foundations and Trends® in Information Retrieval, 3(4), 333-389. <https://doi.org/10.1561/1500000019>

2.Ramos, J. (2003, July). Using TF-IDF to determine word relevance in document queries. In Proceedings of the first instructional conference on machine learning (Vol. 242, pp. 133-142).

3.Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to information retrieval. Cambridge University Press.

4.Bird, S., Klein, E., & Loper, E. (2009). Natural language processing with Python: analyzing text with the natural language toolkit. " O’Reilly Media, Inc.".

5.Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. the Journal of machine Learning research, 12, 2825-2830.

6.Jones, E., Oliphant, T., Peterson, P., et al. (2001). SciPy: Open-source scientific tools for Python. http://www.scipy.org/

7.Shakespeare, W. (2003). Macbeth. The Norton Shakespeare, 2, 2579-2658.

8.Porter, M. F. (1980). An algorithm for suffix stripping. Program, 14(3), 130-137.

9.NLKT official documentation https://www.nltk.org/book/ch02.html