Kyle Tapang

Cheng Zhai

CS410

Understanding BM25

Many state-of-the-art ranking functions tend to rely on these four things. Bag-of-words, Term Frequency (TF), Document Frequency (DF), and document length. BM25 is a ranking function that incorporates all of these. We are going to be breaking down how BM25 works and see how it applies these features.

Let’s start off with trying to understand TF. Assume that if a query has terms that show up in a document (ex: a webpage, pdf, etc.) that the number of matching terms between the query and the document can act as a metric for ranking. We’ll refer to this metric as TF. The greater the number of matching terms, the higher the rank of the document. Let’s look at an example.

query = “NYC places to eat”

document_one = “...**places** 1st in **NYC** marathon”

document_two = “**NYC**…cool **places to eat**”

In this scenario, we see that document_one has 2 matching terms while document_two has 4 matching terms. In other words, document_two is more relevant to our query.

However, there is an issue with this type of ranking. Suppose we take the previous scenario, and we introduce a new document

document_3 = “…travel guide …**places** …**to** …**to**…**to**…**to**…**to**…**to**…**to**…**to**”.

This document would have 9 matching terms, making it the most relevant to our query. That seems odd as you can argue that the focus of this query is “NYC” or “eat”. We can say that document 2 is more relevant since it mentions these terms while document 3 does not mention them at all. The issue here is that each term is equivalent in weight. Should “to” be weighed the same as “NYC”? How can we give words like “NYC” or “eat” greater value and decrease the weight of common words like “to”?

We can use something that is referred to as Inverse Document Frequency (IDF). IDF is a formula that measures how often a term occurs in a collection of documents and reduces the significance of terms that are common. In this case, IDF will penalize “to” and give higher weight to “NYC” or “eat”. IDF is defined by taking the log of the number of documents in a collection divided by the number of documents containing that word.

IDF(w) = log [number_of_documents in the collection + 1) / document_frequency]

The combination of these two methods is TF * IDF, known as TF-IDF. This leads into BM25 which is essentially an improvement of TF-IDF.

The first thing that BM25 improves upon is term saturation. Going back to the previous scenario, suppose we have a document_four which contains the word “NYC” 500 times. It’s clear that this document has relevancy, but can we say that it is 500 times more relevant than document_two that only has 1 occurrence of this word? Once a document hits a certain point, we can say that the number of occurrences for a given word shouldn’t have a significant impact and that it is now “saturated”. The formula for this is:

TF(w, d) = (k + 1) * x / (x + k)

x = # of occurrences of the word for a given document

k = upper bound tuning parameter

In this formula *k* provides an upper bound for term frequency. The upper bound being *k + 1*. Observing this function, you will see that term frequency will steadily increase and then slowly taper off as it approaches the upper bound.

Another issue that BM25 covers is document length. Suppose we compare two documents that have the same TF but different document lengths, should these two documents be weighed differently? We can argue that a longer document is less relevant because the occurrence of terms is relative to a document’s length.

document_five has 1 matching term “NYC” but has a length of 100 words document_six has 1 matching term “NYC” but has a length of 1000 words

In the above scenario, we can say that document_five is more likely about “NYC” than document_six is. This is where we use a pivoted length normalizer.

normalizer = 1 – b + b * (document_length / average_document_length_in_a_collection)

b = penalizing tuning parameter

Using *b*, we can control how we penalize length. The greater the value of *b* the greater the penalization of long documents. Overall, each of these features provides a rough description of how BM25 and its components work.

References:

https://kmwllc.com/index.php/2020/03/20/understanding-tf-idf-and-bm-25/

https://www.elastic.co/blog/practical-bm25-part-2-the-bm25-algorithm-and-its-variables

https://opensourceconnections.com/blog/2015/10/16/bm25-the-next-generation-of-lucene-relevation/