**Pinaki Mohanty (0029544263)  
CS 473(Web Information Search, Retrieval, and Management) - Fall 2020  
Prof. Chris Clifton (MWF - 1:30 pm – 2:20 pm – KRAN G016)   
Project 1 – Part 2  
Due on October 11th, 2020**

Q1 TFIDF vs Okapi

**Comparison:**

TF-IDF works on the simple idea of keyword matching and visualizing the document and query in an n-dimensional space. Cosine Similarity is used as a scoring metric to know how close the document and the query are to one another. My model performs stopword removal and stemming on both queries and documents. I also disregard punctuations.

Okapi is essentially the BIM, but with improvements. It is a probabilistic model i.e. ranks documents based on probabilities of query terms appearing in relevant and non-relevant documents. RSV is used as a scoring metric. Also, here queries are weighted differently (unlike TF-IDF where both query and document are treated alike; however in our implementation, query is unweighted i.e. only the presence or absence matters). Term frequency also matters(making it similar to TF-IDF). Another plus point of Okapi is that it considers the length of the document, something not seen in TF-IDF. Galago’s Okapi model is a lot quicker than the TF-IDF I implemented when it comes to retrieval. It has the feature to report the top k documents.

**Formal Evaluation:**

Say we take q1(“what articles exist which deal with tss time sharing system an operating system for ibm computers”)

My Model’s Output with full corpus for q1:

<https://docs.google.com/document/d/1OaNmpQMYBAREdrqsrbDwQbAl6f-WtEym8TCNjymfZnE/edit?usp=sharing>

Okapi’s output for full corpus for q1:

Query: galago batch-search --index=project1-index --defaultTextPart=postings.krovetz --query="#combine(what articles exist which deal with tss time sharing system an operating system for ibm computers)" --scorer=bm25

<https://docs.google.com/document/d/1jOVhp-f3LIJHgqIo613b-Gz7rPZqMQYZ1L6sa1gFuSg/edit?usp=sharing>

Relevant documents for q1:

Graphical user interface, text

Description automatically generated

Both the models contain all the relevant documents-

Hence,

PrecisionTF-IDF=5/1321=0.003

RecallTF-IDF= 5/5=1

F1-scoreTF-IDF=0.005

PrecisionOkapi=5/1000=0.005

RecallOkapi= 5/5=1

F1-scoreOkapi=0.009

Certainly, Okapi beats TF-IDF in terms of precision and F1-score, but the difference is not much. Now, let’s look at the rankings

|  |  |  |  |
| --- | --- | --- | --- |
| Document | Ranking in TF-IDF | Ranking in Okapi |  |
| 1410 | 40 | 4 | 36 |
| 1572 | 57 | 36 | 21 |
| 1605 | 18 | 6 | 12 |
| 2020 | 715 | 415 | 300 |
| 2358 | 225 | 115 | 110 |

Clearly, for every document, Okapi does a better ranking.

*Remark: When working with full corpus, I was getting broken pipe error because the screen was static. Once I had a few print statements in my source code, I saw the output for full corpus being printed out.*

*Basically, if testing with full corpus, have a few print statements to ensure some monitor activity.*

**Which is better?**

Galago’s Okapi is better than my model.

Reasons-

1. Scales well to big data; multi-threading nature. Faster execution
2. Considers relevant documents for ranking
3. Can accommodate large vocabulary and documents.
4. Considers length of document while finding RSV
5. Addition of more terms to query does not hurt RSV. In TF-IDF however, if the new word does not match, it might bring down the similarity, as the query vector is now further from document vector.

*Observation:*

While working with my model, in my experience, TF-IDF can get extremely computationally expensive if vocab and/or docs increase in number. For example, working with the smaller corpus ( 2k words and 100 docs) was a lot easier and manageable than working with full corpus( 17.8k words and 3k docs)

Q2

*Interesting Discoveries*

I worked my way from the queries to the documents

The documents to be considered would be union of all docs having the query words.

Ex,

qw1={doc1,doc2,doc3,doc4}

qw2={doc3}

qw3={doc2,doc5}

List={doc1,doc2,doc3,doc4,doc5}

A good way to know if you calculated the TF-IDF correctly is that, all your retrieved items should have positive(>0) cosine similarities( because at least 1 term is matching).

The Krovetz Stemmer was something new to know about. This is a hybrid stemmer i.e. the output is dictionary based; it produces words not stems.

Potential Problem I notice is context transformation, for example, policy 🡪police.

I also noticed how stemming can take down the vocab but increase the matches by a lot. Thereby, taking computation time up. For example, my vocab for full corpus went from 17.8k to 14.3k after stemming and stopword removal.

I noticed how the .json file has terms like ‘by’ , ‘in’, ‘of’, ‘or’. Hence, I decided to remove these stopwords and also stem the queries; operating got stemmed to operate, computers got stemmed to computer etc.

Since runtime might be an issue, these are the things I did: used numpy for faster computation, avoided redundant calculations by crosschecking and removing duplicates, cut down on galago commands, disregarded punctuations.