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:

1)Subset of Corpus and all queries-

I copied the output of my tfidf model for cacm100 corpus into a text file called tfidfoutput

Query: galago eval --judgments=/homes/cs473/project1/cacm_fullpath.rel --baseline=tfidfoutput.txt

```
num_ret                      all 2098.00000
num_rel                      all  720.00000
num_rel_ret                  all    0.00000
num_unjug_ret@20             all  881.00000
map                          all    0.00000
R-prec                       all    0.00000
bpref                        all    0.00000
recip_rank                   all    0.00000
ndcg                         all    0.00000
ndcg5                        all    0.00000
ndcg10                       all    0.00000
ndcg20                       all    0.00000
ERR                          all    0.00000
ERR10                        all    0.00000
ERR20                        all    0.00000
P5                           all    0.00000
P10                          all    0.00000
P15                          all    0.00000
P20                          all    0.00000
P30                          all    0.00000
P100                         all    0.00000
P200                         all    0.00000
P500                         all    0.00000
P1000                        all    0.00000
```

Query: galago batch-search --defaultTextPart=postings.krovetz --index=project1.4-index /homes/cs473/project1/allqueries.json --scorer=bm25 >outputokapi.txt

Query: galago eval --judgments=/homes/cs473/project1/cacm_fullpath.rel --baseline=outputokapi.txt

```
num_ret                        all 3468.00000
num_rel                        all  720.00000
num_rel_ret                    all    0.00000
num_unjug_ret@20               all  935.00000
map                            all    0.00000
R-prec                         all    0.00000
bpref                          all    0.00000
recip_rank                     all    0.00000
ndcg                           all    0.00000
ndcg5                          all    0.00000
ndcg10                         all    0.00000
ndcg20                         all    0.00000
ERR                            all    0.00000
ERR10                          all    0.00000
ERR20                          all    0.00000
P5                             all    0.00000
P10                            all    0.00000
P15                            all    0.00000
P20                            all    0.00000
P30                            all    0.00000
P100                           all    0.00000
P200                           all    0.00000
P500                           all    0.00000
P1000                          all    0.00000
```

The no of relevant and retrieved for both the models is 0. This is plausible as the corpus size is small and on inspecting the cacm100.rel, I noticed that most relevant files lie outside the corpus's scope. The only thing we can infer is that Okapi returns(3468) more results than my model(2098).

2) Full Corpus and all queries-

I pasted my output for the whole corpus in file tfidfoutputfullcorpus.txt. This took a lot of time.

galago eval --judgments=/homes/cs473/project1/cacm_fullpath.rel --baseline=tfidfoutputfullcorpus.txt

```
mc18 65 $ galago eval --judgments=/homes/cs473/project1/cacm_fullpath.rel --baseline=tfidfoutputfullcorpus.txt
num_ret                        all 64013.00000
num_rel                        all   720.00000
num_rel_ret                    all   667.00000
num_unjug_ret@20               all   777.00000
map                            all     0.18364
R-prec                         all     0.20200
bpref                          all     0.93489
recip_rank                     all     0.51720
ndcg                           all     0.49559
ndcg5                          all     0.28407
ndcg10                         all     0.27774
ndcg20                         all     0.28633
ERR                            all     0.05645
ERR10                          all     0.04821
ERR20                          all     0.05213
P5                             all     0.25106
P10                            all     0.22340
P15                            all     0.19149
P20                            all     0.17340
P30                            all     0.14184
P100                           all     0.07596
P200                           all     0.04738
P500                           all     0.02489
P1000                          all     0.01593
```

Precision=667/64013=0.010

Recall=667/720=0.926

F1-score=0.019

Query: galago batch-search --defaultTextPart=postings.krovetz --index=project1-index /homes/cs473/project1/allqueries.json --scorer=bm25 >outputokapi.txt

Query: galago eval --judgments=/homes/cs473/project1/cacm_fullpath.rel --baseline=outputokapi.txt

```
mc18 82 $ galago eval --judgments=/homes/cs473/project1/cacm_fullpath.rel --baseline=outputokapi.txt
num_ret                          all 45900.00000
num_rel                          all   720.00000
num_rel_ret                      all   633.00000
num_unjug_ret@20                 all   726.00000
map                              all     0.32209
R-prec                           all     0.32116
bpref                            all     0.89472
recip_rank                       all     0.73334
ndcg                             all     0.60969
ndcg5                            all     0.50366
ndcg10                           all     0.47978
ndcg20                           all     0.45360
ERR                              all     0.08378
ERR10                            all     0.07649
ERR20                            all     0.07988
P5                               all     0.42979
P10                              all     0.33617
P15                              all     0.25816
P20                              all     0.22766
P30                              all     0.18085
P100                             all     0.08128
P200                             all     0.04862
P500                             all     0.02430
P1000                            all     0.01387
```

Precision=633/45900=0.013
Recall=633/720=0.879
F1-score=0.025

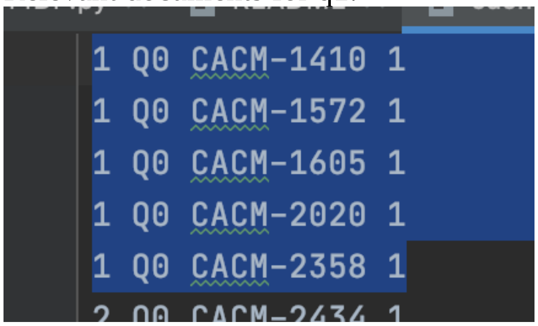
My model beats Okapi when it comes to Recall, but Okapi performs better as far as Precision and F1-score are considered.

3)Full Corpus with 1 random query-
Say we take q1("what articles exist which deal with tss time sharing system an operating system for ibm computers") at random.
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:



Both the models contain all the relevant documents-
Hence,
Precision$_{\text{TF-IDF}}$=5/1321=0.003
Recall$_{\text{TF-IDF}}$= 5/5=1
F1-score$_{\text{TF-IDF}}$=0.005

Precision$_{\text{Okapi}}$=5/1000=0.005
Recall$_{\text{Okapi}}$= 5/5=1
F1-score$_{\text{Okapi}}$=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	δ
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}
Also, I tried removing duplicate query words as the document list for that query would be same and that extra computation would make no use if eventually all the query word lists will be 'union-ed'.

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.

I also found the coverage of doc terms interesting.

Say,

$List_{q1}$={doc1, doc2,doc3}

$List_{q2}$={doc1, doc3,doc4}

For q2, you only need to compute doc4, because doc1,doc2 ϵ $List_{q1}$.doc1,doc2 have already been populated; we know their vector representation. One will notice that my program gets faster with subsequent queries. If we have queries with larger covers in the beginning, the initiation time might be high, but eventually it would go down. Thereby making execution faster.

$\mathrm{Coverage}_{q1} \geq \mathrm{Coverage}_{q2} \geq \mathrm{Coverage}_{q3} \ldots \geq \mathrm{Coverage}_{qn}$ (for fastest execution)

Merits of VS with TF-IDF model-

1) Simple, easy to implement and understand.
2) Great if vocab and no of docs are less.
3) Treats rare words and frequent words differently.

Merits of Okapi Model-

1) Considers relevant documents for ranking.
2) Can accommodate large vocabulary and documents.
3) Considers length of document while finding RSV.
4) 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.