Thomas Enns - 20823674
MSCI 541
2022-11-21

HW4 – Report

1. The simplest way to explain how two documents can have the same BM25 score and one be relevant while the other is not, is with an example. Suppose there are two short documents of the same length. The first about a painting, the second about a fish migration.
   The first document: "Every fall, the Pacific salmon migrate up the deep blue rivers of the Rocky Mountains to mate."
   The second document: "Painting of a sunset over the Rocky Mountains with beautiful salmon colour crossing the deep blue sky."
   Say I wanted to learn about Salmon in the rivers of the rocky mountains, I could type the following **query:** "Salmon in Rocky Mountains".
   Now recall that BM25 is a TF-IDF algorithm which takes into account term frequency in both a query and a document. In the above example, the term frequencies in the documents for the terms in the query would be equal. The terms are used in a different context, which is why only the first document is relevant. However, their score would be the same as the documents each have the same query terms at the same frequencies and are the same length.
   Therefore, it is possible to have equal BM25 scores even when some docs are relevant and others not because the same words can have different meanings and be used in different contexts.
2.
   a. The *log(f_i)* notation gives decreasing importance to each additional instance of a term in a document or query. Meaning, for example that the weight for a term appearing twice is significantly higher than the weight of a term appearing once, but the weight of a term appearing 20 times would only be marginally better than a term appearing 19 times. This essentially causes a non-linear increase in weighting proportional to term frequency (weighting increases at a decreasing rate).
   b. BM25 achieves a similar behavior of term weight increasing at a decreasing rate for each occurrence through the fractional formulation of the TF terms.

$$BM25 = \sum_{i \in Q} \underline{\frac{(k_1+1)f_i}{K+f_i}} \times \underline{\frac{(k_2+1)qf_i}{k_2+qf_i}} \times \underline{\log\left(\frac{N-n_i+0.5}{n_i+0.5}\right)}$$

   As you can see in the formulation above, the three terms are underlined, the first is the term frequency for the document, the second is the term frequency of the query, and the third is the inverse document frequency. In the two term frequency terms of the formulation, the frequency of a term is multiplied by some constant and then divided by itself plus either a function of the document length or a constant. This results in a similar behavior of decreasing importance to each additional instance of a term. The TF terms of the formulation would increase by initially large and steadily decreasing amounts for every instance of a word in a document or query.

3. Stemming is the practice of reducing a 'family' of words such as {"running", "runner", "runs"} to one 'root' word, such as "run". When this is done for the terms in queries as well as the documents in a repository, it increases recall as there will be more matches between terms in documents and terms in documents. Since the number of terms in documents matching those in the query is increased, BM25 must be computed for more documents. Therefore, stemming slows down retrieval speed.
4. When queries contain words that are not found in the document collection, they are essentially ignored; they do not affect the score of the documents in any way. This is because the BM25 score is a sum for every term in the query, of the product of the term frequency in the doc, the term frequency in the query, and the inverse document frequency. When the document TF is 0 for every document, the value of the partial BM25 score for that term will be zero. Thus, this will not add to or take away from the overall BM25 score. Essentially, adding a word that is not found in the document collection to any query will not have any effect on what is returned by that query.
5. The number of documents is 131,896 (retrieved from properties of collection folder). The number of words in the vocabulary is 247,034 (retrieved by printing the length of the lexicon). Assuming that these figures are correct, the matrix would use:
$$131{,}896 * 247{,}034 * 4\ bytes = 1.3E^{11}\ bytes = 130.33\ gigabytes$$
The amount of memory used by the inverted index is calculated by looping over the index and summing the length of each postings list. Thus, the number of integers stored is 31,922,653 in the inverted index. Assuming this is correct, the inverted index uses:
$$31{,}922{,}653 * 4\ bytes = 1.28 * 10^8\ bytes = 127.69\ megabytes$$
Thus, you would save 130.20 gigabytes by not storing zeroes (The index uses less than 1% of the memory used by the matrix).
6. In this situation, Gary stems the tokens in the query using the Porter Stemmer. However, he does not stem the tokens in the documents. Stemming only the query means that the stemmed 'root' versions of words will have few or no matches in the documents which still have the long versions of words. It is in fact possible that stemming only the query results in worse performance than no stemming at all.
7. Instructions to run this code is contained in Appendix A.

The two BM25 retrieval was performed both with stemming and without. After using the HW3 programs to calculate precision and gain metrics, we can conclude that stemming significantly improves search retrieval overall but not in the top ten.

Run Name	Mean Average Precision	Mean P@10	Mean NDCG@ 10	Mean NDCG@ 1000	Mean TBG
baseline	0.209	0.253	0.335	0.425	1.767
stem	0.251	0.284	0.374	0.487	2.046
p-value	0.005	0.151	0.101	0.002	0.001

Table 1: Mean Run Metrics BM25

In Table 1, we can see that there is a statistically significant improvement in Mean Average Precision ($p = 0.005$), Mean NDCG at rank 1000 ($p = 0.002$), and Mean Time Biased Gain ($p = 0.001$) when using stemming compared to BM25 without stemming. This is significant at a more than 99% certainty level. However, for Average Precision at rank 10 ($p = 0.151$), and Mean NDCG at rank 10 ($p = 1.01$), there was only a small and not statistically significant improvement. This is most likely because overall, the top ten documents most likely changed less than the rest of the list of returned documents. Stemming is a tool that primarily enhances recall. The documents with the highest BM25 scores before stemming are unlikely to be displaced by others even when stemmed.

Below, we can see the percent change in metrics for each topic from the Baseline BM25 algorithm to the Stemmed BM25 algorithm.

Topic	AP	AP_10	NDCG_10	NDCG_1000	TBG
401	121%	200%	160%	30%	51%
402	242%	50%	31%	125%	92%
403	0%	0%	0%	0%	0%
404	139%	0%	0%	18%	159407%
405	**-24%**	**-50%**	**-52%**	**-9%**	**-7%**
406	14%	50%	20%	5%	8%
407	**-18%**	**-33%**	**-22%**	**-10%**	**-9%**
408	45%	**-25%**	**-34%**	27%	87%
409	0%	0%	0%	0%	0%
410	0%	0%	0%	0%	0%
411	83%	50%	45%	27%	54%
412	24%	14%	10%	14%	7%
413	725%	0%	0%	80%	2022%
414	15%	0%	21%	6%	**-11%**
415	0%	0%	0%	0%	0%
417	5%	0%	**-1%**	2%	10%
418	91%	0%	1%	81%	23%
419	15%	50%	18%	3%	37%
420	**0%**	0%	1%	**0%**	**-2%**
421	5%	0%	0%	17%	4%
422	4%	75%	71%	3%	6%
424	650%	**-50%**	**-55%**	414%	46%
425	69%	17%	21%	46%	21%
426	25%	**-67%**	**-65%**	6%	14%
427	22%	0%	0%	26%	27%
428	**-57%**	0%	**-50%**	**-30%**	30%
429	182%	300%	136%	60%	131%
430	29%	0%	5%	10%	17%
431	225%	Infinite	Infinite	103%	38%

432	**-53%**	0%	0%	**-34%**	**-71%**
433	**-3%**	0%	0%	**-1%**	**-24%**
434	**-2%**	0%	0%	**-1%**	**-5%**
435	**-36%**	**-100%**	**-100%**	**-23%**	**-29%**
436	146%	133%	184%	57%	65%
438	21%	**-50%**	**-37%**	6%	15%
439	285%	0%	0%	33%	3381%
440	**-2%**	**-17%**	**-10%**	**-1%**	**-2%**
441	7%	0%	3%	3%	**0%**
442	**-8%**	0%	**-5%**	**-7%**	**-13%**
443	**-3%**	0%	**-19%**	**-3%**	18%
445	0%	0%	0%	0%	0%
446	**-16%**	0%	0%	13%	**-48%**
448	**-48%**	0%	0%	**-8%**	**-94%**
449	**-22%**	0%	0%	**-6%**	**-21%**
450	7%	0%	45%	11%	0%

Table 2: Per-Topic Percent Improvement of Stemming Over Baseline

In Table 2 above, the instances in which there was a negative percent improvement in a metric for stemming when compared to baseline are bolded and highlighted. The instances where there was no change in performance for a metric are highlighted but not bolded. Any non-formatted instances indicate improvement from the baseline BM25 algorithm. For many topics, stemming did not offer significant improvement over the baseline algorithm. Notably, the two metrics that only consider the top ten results showed the least improvement. However, there were significant changes in performance for a number of query topics.

Some of the topics with the most significant improvement due to stemming are topics 401 (foreign minorities, Germany), 402 (behavioral genetics), 429 (Legionnaires' disease), 431 (robotic technology), and 436 (railway accidents). Notably, for query 401, the number of relevant documents in the top ten increased from 0. All the words in each of these queries is reduced to a different form by the porter stemmer (except “railway”). This would drastically improve recall for all of these queries. In the case of 401, 402, and 436, all of these queries contain plural words that you would be unlikely to find in a document but you would be far more likely to find their singular form, especially in reference to the desired topics. In that same vein, these are general topics for which many documents could be relevant, meaning that increased recall is a significant improvement All of these factors contributed to the excellent performance of the algorithm with stemming for the queries in question. In essence, stemming is likely to improve queries with general topics, with plural terms, as well as those with a high frequency of ‘stemmable’ words.

The topics for which the stemming had the most significant negative effect on performance are 405 (cosmic events), 407 (poaching, wildlife preserves), 435 (curbing population growth), 440 (child labor). In the case of 405 (cosmic events), there were two relevant documents returned in the top ten with the baseline algorithm and then only one with stemmed words. In this case, precision was decreased as recall was seemingly increased, as stemming also removes some of the meaning of words. This same sort of situation takes place with the other queries, where stemming essentially dilutes the results of the query, putting more irrelevant documents at a higher rank. All of these queries are quite specific in their information needs, and use words which, are dependent on more than their root to discern their meaning. For example, “curbing” and “curb” (noun), have very different meetings. The queries are all context dependent terms that are likely to be used in any document discussing the desired information. Thus, it is likely that stemming them would add more incorrect results than correct ones (the correct ones having already been found).

Appendix A – README File (Instructions to run code)

```
# MSCI-541-HW4

Thomas Enns - 20823674

This repository consists of 9 programs which work together to create then query
an index of files, then to evaluate search results and performance.

The first, IndexEngine.py, accepts two arguments: the filepath to latimes.gz, and
the filepath to a folder that will be created and will house all the
files/documents. The program will split the gzip into the approx. 131 000 files,
create an inverted index of each token (word) present linked to all the docs in
which it appears, store the index, store their metadata, and store the files in
the specified folder, divided by date.  An example of the appropriate way to run
this program from the console is as follows:
python IndexEngine.py C:\Users\thoma\Documents\3Bterm\MSCI541\latimes.gz
C:\Users\thoma\Documents\3Bterm\MSCI541\testdir

The second, GetDoc.py, accepts three arguments: the filepath to the document
store, the type of identifier used to request a document (either DOCNO or doc
id), and the identifier itself. The program will then return the corresponding
document along with it's metadata to the console. An example of the appropriate
way to run this program from the console is as follows:
python GetDoc.py C:\Users\thoma\Documents\3Bterm\MSCI541\testdirectory docno
LA123190-0134

The third, BooleanAND.py, accepts three arguments: the location of the index
file, the name of a query input file, and the name of a query output file. The
program will parse and tokenize the queries and print the list of documents
containing all the query terms in the output file. An example of the appropriate
way to run this program from the console is as follows:
python BooleanAND.py C:\Users\thoma\Documents\3Bterm\MSCI541\testdir queries.txt
hw2-results-tenns.txt

The fourth program, Helper.py, is not meant to be run. It only contains methods
that can be used by the other programs such as a tokenizer and a method to
convert tokens to ids. The tokenizer is used by both the InvertedIndex program
and all querying programs. It has an option to use the stemmer or not.

The fifth program, Evaluate.py, accepts two arguments: the name of the .txt file
of relevance judgements for the queries and the name of the run file with the
results of the queries. It computes effectiveness measures for each query and
outputs them to both a .txt and a .csv in the measures folder. An example of the
appropriate wat to run this program from the console is as follows:
```

```
python Evaluate.py LA-only.trec8-401.450.minus416-423-437-444-447.txt hw4-bm25-stem-tenns.txt

The sixth program, Summarize.py, accepts no arguments. It iterates over all the results files created by the Evaluate program and outputs a table of the average performance metrics for each run file. An example of the appropriate wat to run this program from the console is as follows:
python Summarize.py

The seventh program, compare.py, accepts three arguments, the names of the two .csv files created by the Evaluate program which the user wishes to compare, and a number corresponding to the desired metric 1-MAP. 2-AP@10, 3-Mean NDCG@10, 4-Mean NDCG@1000, 5-Mean TBG. The program outputs a simple .csv file with the desired metric for each run, the percent improvement of the better performing run over the worse performing run, and the p-value. An example of the appropriate wat to run this program from the console is as follows:
python compare.py hw4-bm25-baseline-tenns-measures.csv hw4-bm25-stem-tenns-measures.csv 1

The 8th program, Calc.py, accepts no arguments, and simply computes the number of words in collection, number of non-zero instances of terms in documents, and the average document length in collection. It outputs these to the console. (This was used for Q5 and for avg doc length for BM25) An example of the appropriate wat to run this program from the console is as follows:
python Calc.py

The ninth program BM25.py, accepts three arguments: the location of the index file, the name of a query input file, and the name of a query output file. The program will parse and tokenize the queries and print the list of the top 1000 ranking documents according to the BM25 equation in the output file. An example of the appropriate wat to run this program from the console is as follows:
python BM25.py C:\Users\thoma\Documents\3Bterm\MSCI541\testdirS queries.txt hw4-bm25-stem-tenns.txt
```