Text Retrieval and Search Engines

Assignment 2

Submitters:

Ariel Suller: 324369412

Avi Ferdman: 316420132

Vector Space Model

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | d1 | d2 | d3 | d4 |
| a | 0 | 1 | 1 | 1 |
| b | 1 | 2 | 0 | 1 |
| c | 2 | 0 | 0 | 0 |
| d | 0 | 0 | 0 | 0 |
| e | 1 | 0 | 1 | 1 |
| f | 7 | 5 | 7 | 2 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  | tf | wf | df | Idf | qi = wf \* idf |
| a | 3 | 2.09861229 | 3 | -1.0986123 | -2.3055612 |
| b | 4 | 2.38629436 | 3 | -0.8697417 | -2.0754597 |
| c | 2 | 1.69314718 | 1 | -0.526589 | -0.8915927 |
| d | 0 | 0 | 0 | 0 | 0 |
| e | 3 | 2.09861229 | 3 | -0.7412763 | -1.5556516 |
| f | 21 | 4.04452244 | 4 | -1.3973635 | -5.651668 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | d1 |  |  |
|  | tf | wf | di |
| a | 0 | 0 | 0 |
| b | 1 | 1 | 0.27171698 |
| c | 2 | 1.69314718 | 0.46005684 |
| d | 0 | 0 | 0 |
| e | 1 | 1 | 0.27171698 |
| f | 7 | 2.94591015 | 0.80045381 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | d2 |  |  |
|  | tf | wf | di |
| a | 1 | 1 | 0.30595074 |
| b | 2 | 1.69314718 | 0.51801964 |
| c | 0 | 0 | 0 |
| d | 0 | 0 | 0 |
| e | 0 | 0 | 0 |
| f | 5 | 2.60943791 | 0.38322429 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | d3 |  |  |
|  | tf | wf | di |
| a | 1 | 1 | 0.30593202 |
| b | 0 | 0 | 0 |
| c | 0 | 0 | 0 |
| d | 0 | 0 | 0 |
| e | 1 | 1 | 0.30593202 |
| f | 7 | 2.94591015 | 0.90124825 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | d4 |  |  |
|  | tf | wf | di |
| a | 1 | 1 | 0.41274558 |
| b | 1 | 1 | 0.41274558 |
| c | 0 | 0 | 0 |
| d | 0 | 0 | 0 |
| e | 1 | 1 | 0.41274558 |
| f | 2 | 1.69314718 | 0.69883902 |

0.713309932

0.447508076

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | tf | wf | df | Idf | qi = wf \* idf |
| a | 3 | 2.09861229 | 3 | -1.0986123 | -2.3055612 |
| b | 4 | 2.38629436 | 3 | -0.8697417 | -2.0754597 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| f | 21 | 4.04452244 | 4 | -1.3973635 | -5.651668 |

Term Weighting

Short-documents bias happens because cosine similarity focuses on the direction of the document vector and doesn't consider the document's length. As a result, short documents usually have fewer terms, and the few terms they have can have a big impact on their similarity score. If two short documents share even a few words, their similarity score can be much higher than it should be, compared to longer documents with more diverse content.

This bias can be an issue because:

* **Limited context in short documents:** Short documents often don't have enough content to make meaningful comparisons, but they can still score highly if they share a few terms.
* **Sparse term representation:** When short documents share terms, these terms might be given too much importance, which doesn't reflect how relevant the documents are overall.

**Given the weighted tf function:**

Where:

* is a constant between 0 and 1
* is the maximum raw term frequency in document d,

**Why is this weighted tf function useful?**

This function adjusts raw term frequency by:

1. **Normalizing term frequencies**: Dividing​ by ensures that term importance is scaled relative to the most frequent term in the document. This avoids overemphasizing documents with very high raw term frequencies for a few terms.
2. **Addressing term-frequency imbalance**: The constant ensures that even low-frequency terms receive a baseline weight, preventing their importance from being completely diminished in the normalization process.

**What issue might arise?**

1. **Sensitivity to parameter α\alphaα**: The choice of can significantly affect the weighting. For example:
   * If is too high, all terms will have similar weights, reducing the impact of term frequency and hurting relevance.
   * If is too low, rare terms might receive too little weight, potentially skewing the model.
2. **Favoring high-term-frequency documents**: Documents with high might still dominate the similarity score because terms with high raw frequencies retain relatively high weights after normalization.

Relevance feedback and evaluation

|  |  |
| --- | --- |
| DocId | Relevance |
| 5 | 4 |
| 2 | 1 |
| 1 | 1 |
| 3 | 3 |
| 4 | 0 |

**Rocchio’s Algorithm:**

**Rocchio’s Algorithm with Graded Relevance:**

**Rocchio’s Algorithm with Rank of Relevant Documents:**

TODO: Missing Explanations here (**Rocchio’s Algorithm with Graded Relevance**

and **Rocchio’s Algorithm with Rank of Relevant Documents**)

Evaluation

1. **Explain the problem of using NDCG for evaluation when negative labels (negative relevance judgments) for documents are used.**
2. **Breaking the Assumption of** :

Negative scores can make NDCG go below 0 or even be undefined, breaking its key assumption.

1. **Division by 0**:

If all relevance scores are negative, the "ideal DCG" (used for normalization) can be zero. This leads to division by 0, making NDCG invalid.

1. **Non-monotonicity**:

Adding a more relevant document (with a less negative score) can make the total DCG decrease instead of increase. This violates the expectation that better rankings improve the score.

1. **Score Misinterpretation**:

Negative scores confuse the meaning of NDCG because it’s designed to measure "gain." A negative score implies a loss, making results harder to interpret.

1. **Loss of Normalization Meaning**:

NDCG is normalized to allow comparison across queries, but negative labels make normalization meaningless or unreliable, as it assumes positive scores.

1. **When MAP and MRR Yield the Same Results**

We’re looking for cases where:

1. **Perfect Ranking**: If all relevant documents are ranked consecutively at the top, the AP value equals the reciprocal rank of the first relevant document.
2. **Single Relevant Document Per Query**: Both metrics prioritize the rank of the first relevant document. If there’s only one relevant document, the AP and RR values are identical.
3. **First Relevant Document Determines AP**: If subsequent relevant documents are irrelevant for the AP calculation (e.g., they appear after many irrelevant documents), the AP and RR are identical.
4. **No Relevant Documents Per Query**: If no relevant document is found for a query, the reciprocal rank () is considered 0. If there are no relevant documents, as well.
5. **Name two different examples where:**
6. The removal of stopwords reduces the recall:
   1. When stopwords form an essential part of key phrases, removing them can result in missing documents that contain the exact phrase.
   2. When stopwords are part of titles or headers, their removal may prevent relevant documents from being retrieved.
7. The removal of stopwords reduces precision:
   1. When stopwords are removed and the query becomes too broad, irrelevant documents may be retrieved, reducing precision.
   2. When stopwords help clarify the meaning of a query, removing them can lead to ambiguous results and reduce precision.
8. **Given a query for which there are 3 relevant documents in the collection. The of the search engine for this query is . Answer the following bullets based on this information.**
   1. What is the F1 score at rank 3 for this query?

(2 relevant documents out of 3 retrieved)

(2 out of 3 relevant documents retrieved)

* 1. What are the maximum and minimum possible AP (average precision) values at rank 3 for this query? (You should provide possible values and not general bounds.)

**Maximum AP**: Achieved when the 2 relevant documents are ranked **first and second:**

**Minimum AP**: Achieved when the 2 relevant documents are ranked **second and third:**

True/False questions

1. p@k is a monotonic non-decreasing function with respect to k.

**False**

For example, if we retrieved these docs:

|  |  |
| --- | --- |
| DocId | Relevance |
| 1 | 1 |
| 2 | 0 |
| 3 | 1 |

1. Vector space-based retrieval is always more effective than Boolean retrieval.

**False**

Vector space retrieval uses scoring based on term frequencies and cosine similarity, which can lead to less accurate results when documents contain ambiguous terms or irrelevant information. Boolean retrieval, by contrast, simply requires the presence or absence of specific terms, providing clearer results when exact matches are needed. Therefore, vector space retrieval may introduce noise and rank irrelevant documents higher, making it less effective in some situations.

1. In the vector space model, the higher the value of the normalization factor for a document is, the lower are the chances of retrieval for that document.
2. The stemming process increases the number of unique terms in the index.

**False**

Stemming reduces the number of unique terms in the index. The stemming process involves reducing words to their root form, so different forms of the same word are treated as a single term. This results in fewer unique terms being indexed because words with similar meanings are grouped together under a common root.

1. Values of in F-measure emphasize precision.

**False**

In the F-measure, the parameter beta determines the relative importance of precision and recall. When , the F-measure places more emphasis on recall, not precision. This is because a larger beta value gives greater weight to recall in the harmonic mean calculation. Conversely, when , more weight is given to precision.

1. In Rocchio's model, might be closer to the centroid of the relevant documents than .

**True**

In Rocchio's model, represents the original query, while is the modified query after incorporating feedback. It is possible that , the initial query, might be closer to the centroid of the relevant documents than , especially if the feedback modifies the query in such a way that it moves away from the true centroid of relevant documents. This could happen if the weights of non-relevant documents are mistakenly added too strongly or if the model overemphasizes less relevant feedback, causing the updated query to diverge from the optimal centroid of relevant documents.

Wet Part – Intro to Pyserini

PART A:

1. An index was created with the commands supplied, and retrieval was performed using the parameter file.
   1. The retrieval of the word “corporation” returned one document – D2:

****

* 1. From data examination, we would expect to retrieve two documents – D2, as retrieved, and D3 – which was not retrieved. The reason D3 was not retrieved is that the word “corporation” appears as “corporation**s**” in the document, and when performing BM25 retrieval, the exact term is being searched – so we will get no match, even though it should be retrieved.

Creating a new Index using Porter stemmer:

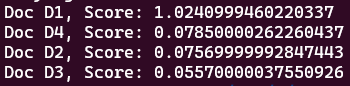
python3 -m pyserini.index.lucene --collection TrecCollection --input ./WET\_PART\_A --index ./WET\_PART\_A\_INDEX --keepStopwords --stemmer porter --storePositions --storeDocvectors -store Raw --optimize

and retrieving again (with required changes in Pyserini – from ‘none’ stemmer to ‘porter’), we get the desired (and expected) results:



1. For this part we used the index created with Porter stemmer.

Retrieving the Query “the nobel” returns the following ranking:



And for the query “nobel”:

  
In both retrievals, D1 gets the highest score. D1 is the only document containing the word “nobel” and that’s the reason for which D1 is getting the highest score for both queries. In the first query the word “The” was added to retrieve more than only one document as this word appears in all documents, but D1 will still be ranked first, for the reason mentioned above.

1. Running the query “Michael Jackson” returns as expected D4:



* 1. For our opinion, document D4 is not relevant for the information need expressed by the query, since a person searching for “Michael Jackson” will expect to get information about the singer, and not about another person admiring him. In our data, this is the only document containing the query words, so it will be retrieved with high score, even though it’s not quite relevant.
  2. A query for which D4 can be relevant (in the terms we defined in previous part (a)), can be “Lady Gaga”, for which we get the result:



Which is higher that the score we got for “Michael Jackson” – which is reasonable as the document is about the biography of Lady Gaga, so her name will appear more frequently the Michael Jackson’s name, hence retrieving higher score.

Part B:

1. Indexing done:
   1. With stopwords removal and Krovetz stemmer:

python3 -m pyserini.index.lucene --collection TrecCollection --input ./WET\_PART\_B/AP\_Coll/AP\_Coll/ --index ./WET\_PART\_B\_INDEX\_Kro --stemmer Krovetz --storePositions --storeDocvectors -storeRaw –optimize --stopwords ./WET\_PART\_B/StopWords.txt

* 1. With stopwords removal and no stemmer:

python3 -m pyserini.index.lucene --collection TrecCollection --input ./WET\_PART\_B/AP\_Coll/AP\_Coll/ --index ./WET\_PART\_B\_INDEX\_no\_stem --stemmer none --storePositions --storeDocvectors -storeRaw –optimize --stopwords ./WET\_PART\_B/StopWords.txt

1. The table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Stopword Removal | Krovetz Stemmer | MAP | P@5 | P@10 |
| With | With | 0.2144 | 0.4121 | 0.3913 |
| With | Without | 0.1896 | 0.4094 | 0.3758 |

We can notice that using Krovetz stemmer improves the MAP in comparison to not using it. The explanation is that when using the stemmer the words in index and query are reduced to their root – that implies that more matches will be made because there won’t be mismatched due to variations in word form (like “corporation” and “corporations”.