Information Retrieval – Final Project

Tali Narovlyansky
ID Number: 318932639
Email: talinaro@post.bgu.ac.il

***Abstract*—In the following Information Retrieval project we asked to build a search engine for English Wikipedia.**

GitHub repo:
https://github.com/talinaro/BGU-IR-Project
Google Storage Bucket:
https://console.cloud.google.com/storage/browser/318932639

1. Key Experiments

In this project, I built a search engine for English Wikipedia as starting at creating inverted indexes for the body and anchor of documents (as was done at HW3). All the experiments were conducted on the whole corpus, running on colab using gcsfs library - a pythonic file-system for Google Cloud Storage.

1.1. First Experiment

The first experiment was to evaluate the performance using the tf-idf algorithm on the body index, but the results were poor in terms of Mean Average Precision at 40 (MAP@40).

The term frequency-inverse document frequency (tf-idf) formula is a common method of weighting the importance of terms in a document. The formula is as follows:

$$tf - idf(t, d) = tf(t, d) \times idf(t) \tag{1}$$

where:

- t is the term
- d is the document
- $tf(t, d)$ is the term frequency, which is the number of occurrences of the term t in document d
- $idf(t)$ is the inverse document frequency, which is calculated as $idf(t) = \log \frac{N}{df_t}$, where N is the total number of documents in the corpus and df_t is the number of documents in the corpus that contain the term t.

1.2. Second Experiment

In the second experiment, I added a search anchor index and found that it improved the search engine's performance in terms of MAP@40, but still not as well as expected.

1.3. Third Experiment

The third experiment was to replace tf-idf with the BM-25 function on the body index.

The BM-25 formula is as follows:

$$BM25(q, d) = \sum_{i=1}^{n} \frac{IDF \times (q_i) f(q_i, d) \times (k_1 + 1)}{f(q_i, d) + k_1 \times (1 - b + b \times \frac{|d|}{avgdl})} \tag{2}$$

where:

- q is the query
- d is the document
- n is the number of query terms
- q_i is the i-th query term
- $f(q_i, d)$ is the frequency of the i-th query term in the document
- $IDF(q_i)$ is the inverse document frequency of the i-th query term
- k_1 is a constant that controls the term frequency scaling. Set to 2
- b is a constant that control the document length normalization factors. Set to 0.75
- $avgdl$ is the average document length

The BM-25 function performed significantly better than the previous two experiments and it had a much higher MAP@40 value.

In addition, I attempted to create title and PageRank inverted indexes to further improve the performance of the search engine. However, I encountered some bugs and was not able to fully implement them within the given time frame.

1.4. Fourth Experiment

In the fourth experiment, I tried to improve the search engine's performance by adjusting the weights that each ranking method received. In the previous experiment, the BM-25 algorithm had an additional weight of 30 added to its ranking score, and the anchor text had a weight of 10. In this experiment, I changed the weight of the BM-25 algorithm to be calculated as

$$\frac{30}{(rank + 1)} \tag{3}$$

This adjustment resulted in an even greater improvement of the search engine's performance, as measured by the Mean

Average Precision at 40 (MAP@40) metric. This experiment demonstrates the importance of fine-tuning the weights of different ranking methods to optimize the search engine's performance.

2. Efficiency Improvements

2.1. Data Structures

In order to speed up the calculations, I used two pre-calculated data structures: The DL (Document Length) and DT (Document Title). Both data structures were created on a cluster in GCP and stored in a bucket, as the indexes.

The DL data structure is a dictionary that stored the length of each document, allowing the search engine to quickly retrieve the length of a document without having to go through the entire document.

The DT data structure is a dictionary that stored the title of each document, which helped to display the search results in a more informative way to the end user.

After creating the indexes, I loaded the code to Google Colab. There created ngrok app and tested the search functions.

2.2. Threads

All the sub-search functions, including the body search, anchor text search, and BM-25, were running as threads in order to make the retrieval efficient. This was done to ensure that multiple calculations could be processed simultaneously, which would increase the overall speed of the search engine.

3. Key Takeaways

Overall, the key takeaways from these experiments were that the anchor index provided additional information that improved the search engine's performance, and that the BM-25 function is a better alternative to tf-idf for ranking documents in a search engine.

4. Qualitative Evaluation

4.1. Well Performed Query: "LinkedIn"

The search engine performed well on the query "LinkedIn" because it is a specific and well-defined query, and the search engine was able to find and retrieve relevant documents that contain the term "LinkedIn" in them. Additionally, the BM-25 algorithm, which was used in the search engine, is well suited for handling such specific queries as it calculates a relevance score by taking into account the frequency of the query term in the document and the length of the document.

4.1.1. Top 10 results.

- Timeline of LinkedIn
- LinkedIn
- LinkedIn Learning
- 2012 LinkedIn hack
- LinkedIn Pulse
- LinkedIn Top Companies
- Ryan Roslansky
- Jeff Weiner
- HiQ Labs v. LinkedIn
- Apache Helix

4.2. Poorly Performed Query: "What is the best place to live in?"

The search engine performed poorly on the query "What is the best place to live in?" because it is an open-ended query. The search engine may have difficulty in understanding the intent behind the query and retrieving relevant documents. Additionally, this type of query might require additional features such as semantic search, natural language processing and understanding, which my search engine does not have.

Another reason could be that the query contains a lot of stop words and the "informative" words are not a part of the relevant document that talks about the best place to live in.

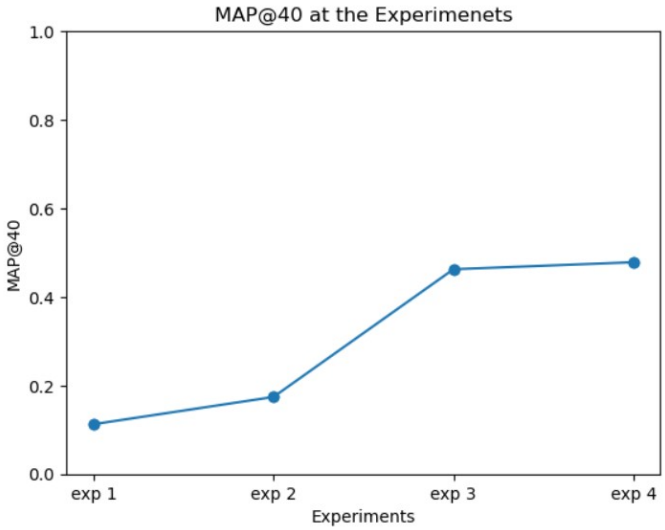
4.2.1. Improvement Suggestions. There are several things that can be done to improve the performance of the search engine for open-ended queries like "What is the best place to live in?":

- **Use more advanced ranking algorithms:** A more advanced ranking algorithm like Latent Semantic Indexing (LSI) or Latent Dirichlet Allocation (LDA) that can take into account the context and meaning of the query rather than just keyword matching.
- **Improve the training data:** By including a larger and more diverse set of data, the search engine can better understand the context of the query and provide more accurate results.

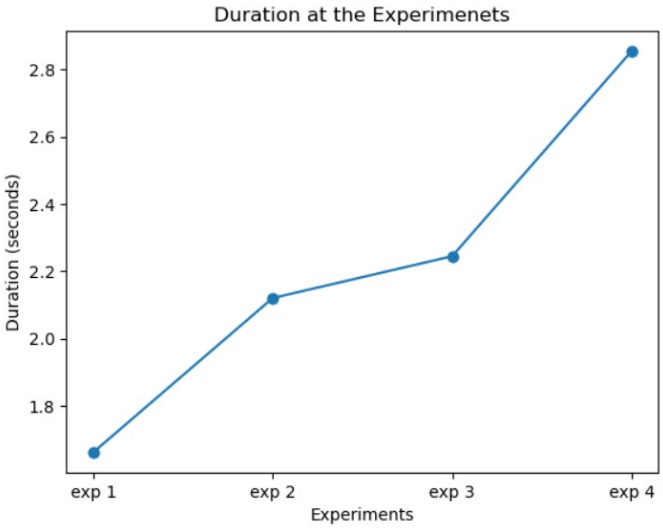
4.2.2. Top 10 results.

- Lee Seung-chul
- No Place to Run (album)
- Saturday Live (British TV programme)
- Campbell Live
- Best (Kenny G album)
- Q Awards
- Chucho Valdés
- Israel Houghton
- The Best of Whats Around Vol. 1
- All the Best – The Live Collection

5. Graphs



A graph showing the engine performance for each major version of my implementation, as described at 1



A graph showing the engine's average retrieval time for each major version of my implementation, as described at 1

6. Appendix

In the github repository, I have a file files.txt with the names of all file located in the bucket.