Eberhard Karls Universität Tübingen
mathematic-natural scientific faculty

TüBing
Modern Search Engines Group Project

Project documentation

Authors:
Lukas Weber, 5406074
Dana Rapp 5445024
Maximilian Jaques, 6200477
Simon Frank, 6095707

Supervisor:
Carsten Eickhoff

April 23, 2024

Contents

In today's digital age, the vast expanse of the internet holds a treasure trove of information waiting to be discovered. However, sifting through this vast sea of data manually is an impossible task. This is where search engines come to the rescue, efficiently indexing and retrieving relevant information. This project implemented a search engine for content related to *Tübingen*. The code base can be found here.

1 Focused crawler

At the core of search engines lies the web crawler, a tool that systematically explores the web content. This section delves into the inner workings of our web crawler, including its design steps and the challenges encountered during its implementation.

1.1 Frontier

The crawling process begins with the frontier, initialized manually with a diverse set of URLs. It comprises links to web pages which are relevant to various queries, along with pages of general importance to individuals living in Tübingen, and information-rich pages, like the Wikipedia-page about Tübingen. The frontier is implemented as a priority queue holding four descending priority levels. Upon creating a new instance of the crawler, all web pages in the initial frontier are assigned the highest priority. The crawler also includes a feature that allows it to resume a previous search. If no frontier is provided when instantiating the crawler, the process continues from where it left off in the previous crawling session.

1.2 Crawling

The crawling process involves a series of iterative steps that continue until either manually interrupted or a predetermined number of crawled documents is reached, if specified. The key steps undertaken during one iteration of the crawling process are as follows:

1. The link with the highest priority is retrieved from the frontier.
2. Web pages have been marked visited in previous iterations are skipped as well as links that lead to non-HTML files.
3. The "robots.txt" file is consulted to determine page access permissions. If access is disallowed, the link is marked as visited, and the corresponding page is skipped.
4. A HTTP request is performed in order to extract page content and all linked URLs.
5. Language detection is performed and topical relevance is verified by inspecting the URL for variations of the word "Tübingen" or by identifying at least one postal code associated with Tübingen. Moreover, the page content is examined for at least three occurrences of "Tübingen" in total. A higher threshold of five occurrences is applied to Wikipedia pages, as even irrelevant ones often contain "Tübingen" multiple times.

A form of the word Tübingen is described with the following regular expression: $(T|t)(\ddot{u}|ue|u)binge(n|r)$

6. Web pages are prioritized. Topic-relevant English documents receive the highest priority, followed by relevant German documents. Non-relevant English content holds priority level 3, while non-relevant German content is assigned priority level 4.
7. The page is marked visited, linked URLs are added to the frontier with same priority.
8. If the page is both English and topically relevant, a similarity duplicate detection process is conducted. If the document is not deemed similar to any other indexed document, it is indexed, signifying its inclusion in the search engine's index.
9. Visited pages, index, and frontier are saved periodically every 25th iteration.

1.3 Further encountered problems and solutions

Initially, topical relevance was determined by checking the single occurrence of "Tübingen". However, after extensive testing, further factors turned out to be necessary to ensure genuine relevance. Therefore, the additional relevance criteria described in point 5 were introduced. Furthermore we encountered the problem that many English web pages lead to German sites within few links. To prevent the index from including a vast amount of German web pages, we introduced language checks with Py3LangID (Barbaresi, 2022) and included page language in page prioritization (see point 6).
In addition, specific websites, such as "Tripadvisor," could detect the crawler. To counter this, we adopted a rotating set of User-Clients in the header of our request to retrieve the associated HTML-document. If a request fails after five seconds, another User-Client is used in the header to avoid detection. By implementing these solutions, we addressed the challenges encountered during the crawling process, ensuring the search engine's efficiency and accuracy in indexing relevant and valuable content.
Python's built-in PriorityQueue framwork does not implement FIFO-queues at different priority levels. We propose a solution with a dictionary of individual FIFO-Queues.

2 Indexing

2.1 Inverted Index

After having crawled the documents, an inverted index is created. This is the core data structure for a modern search engine (Pibiri & Venturini, 2020). The function *inverted_index* in the FocusedWebCrawler.py file implements this functionality. As a data structure we use a dictionary. First, a given document is split into terms. If a term is not in the inverted index yet, it is added as a new item. And as a value, the DocID and the positions in the document itself are added. If the term already exists, then the DocID and the positions in the document are added to the list.

The structure of the implemented inverted index looks like the following:
'term': [[docID1, [pos1, pos5]], [docID2, [pos6, pos9, pos11]], ...].

2.2 Embedding

To use a vector space model for the advanced ranking method, the content of each crawled page must be transformed into a fixed sized vector, which is a representation of the document. While in the lecture a term frequency approach was introduced, we decided to make use of a pre-trained language model for producing the document representation, since they have shown to produce meaningful feature embeddings. Two different language models, BERT and RoBERTa, were tested on a small index. Both models are transformer based bidirectional encoders, which are trained to learn bidirectional representations from a text by jointly conditioning on the left and right contexts in all layers. RoBERTa has the same architecture as BERT, but was pre-trained with a different method (Devlin, Chang, Lee, & Toutanova, 2019)(Liu et al., 2019). Since the better ranking results, were achieved with BERT, we used the base model with 110 million paramters provided by HuggingFace (face, 2023).
BERT is not producing one feature embedding of the input sequence, but one for each token in each layer. In (Ma, Wang, Ng, Nallapati, & Xiang, 2019) the best results on representation tasks were achieved using the average over all tokens in the second to last layer. Therefore, we followed the same approach. The maximum input sequence length of BERT is 512 tokens. If the content of the crawled document exceeds this limit, the input is split into multiple batches and the produced feature embedding for each batch is averaged to obtain one feature embedding for the whole content. Since BERT was trained on full sentences, no pre-processing was applied to the content of the webpage before it was fed into BERT.

3 Ranking

In order to obtain a list of results from a search engine, it is necessary to rank all the documents that have already been indexed. Five different ranking methods have been implemented and the user can choose between them. This feature allows the user to obtain the best results depending on the query.

3.1 Base Methods

As basic ranking methods, TF-IDF and BM25 have been implemented as described in the lecture. The inverted index is used to retrieve all relevant documents. A document is considered relevant if one of the query terms is part of the content of the web page. The scores are then calculated only for the relevant documents. To speed up the calculation in Pyhton, the summations are done by matrix multiplication using numpy (Harris et al., 2020).

Although these are both very simple methods, they still produce good search results, but sometimes lack a bit of variety.

3.2 Embedding

As an innovative ranking approach, the calculated feature embeddings were used in two different ways. First, for a given query, the corresponding feature embedding for the content of the web page was computed as explained in 2.2. Then the cosine similarity between the query feature embedding and all feature embeddings is calculated. This score is then used for the ranking. The results of these methods are quite variable and strongly dependent on the query, which can be explained by the fact that BERT was not trained on a list of terms but on whole sentences. But especially for long queries this method can lead to good results. To make use of the feature embeddings, which have been shown to build meaningful clusters (Ma et al., 2019), a creative and innovative approach is developed. This approach is slightly inspired by the pseudo-relevance feedback method. Therefore, the top five documents ranked by TF-IDF and BM25 are used to compute an average feature embedding vector. This feature embedding is then used as a centre and the documents are ranked by the closest L2 distance of the feature vectors to this point. This method can produce a diverse result list, which may include documents that do not contain the search terms but cover similar topics.

3.3 Mixed Ranking

In order to take advantage of all the different methods described above, the final ranking method combines the results of the TF-IDf, BM25 and Cluster methods to obtain a truly diverse result list. Therefore, a potential gain factor is calculated for each top-ranked element that has not yet been included in the final result list. This gain factor consists of two parts. The first factor is the score calculated by the ranking method. To account for the different scaling of the scores, the TF-IDF score is multiplied by 0.3 and the reciprocal of the cluster score is taken. For BM25, the raw score is used. The second part is independent of the ranking score and depends only on the novelty that the next possible URL adds to the result list. If this site has not been added to the results list, a gain of 10 is added, otherwise a similarity score is calculated between all subpages of the main page. The score for each site is then the profit. As a previously lower ranked site may be an unvisited site, there may be some jumps in the scores and they may not be in descending order. This method can produce a well ranked, diverse list of results.

4 Interface

The website is primarily created for desktop users with the goal of making the search engine pleasant to use for people with poor eyesight. To make the process of finding elements

easier, the background is kept dark and elements are chosen to have a contrasting bright turquoise color with orange visual signals when the mouse hovers over them.

4.1 Landing Page

The landing page was kept pretty simple. Besides a logo there is only a search bar and a drop-down menu to choose the ranker. If no query is entered and the user still wants to search for something, an error message is displayed asking the user to fill in a query. Additionally, when the user starts typing, below the search bar previous queries are displayed starting with the same characters.

4.2 Search Result Page

When a user has entered a query, loading the results takes time. This is due to the creation of audio files which read generated website abstracts for the first 11 results. The audio files read out the abstracts of the search results prepended by the website title and appended by a sentence stating that for further website the user should visit the website. The abstract is generated using Pysummarization (accel brain, 2022), which uses the website content to produce a summary. This is the main functional innovation of our user interface. Its intention is to support visually impaired or dyslexic users in their search process. Most audio files are between 30 and 40 seconds long. They can be sped up using the drop-down menu in each search item or using the one for changing all playback speeds. This guarantees a still rather fast search experience, giving users the tools to gather information for the search results without the need to read any text. When the user actually wants to visit a website, of course he has to rely on other software or methods to cope with his reading disadvantages. Additionally, to support this innovation, the first three results are displayed as a big row across the screen, while the following results are split into two boxes per row each displaying a search result.
In the upper left part of the website again the search engines logo is displayed. On the upper right side, the user has the opportunity to go back to the start page, enter a different query, and use a different ranking model. At the bottom of the page, the next or previous results can be viewed.

References

accel brain. (2022). *pysummarization.* https://pypi.org/project/pysummarization/. PyPI.

Barbaresi, A. (2022). *py3langid.* https://github.com/adbar/py3langid. Github.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). *Bert: Pre-training of deep bidirectional transformers for language understanding.*

face, H. (2023). *Bert base model.* https://huggingface.co/bert-base-uncased. (Accessed on July 5, 2023)

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... Oliphant, T. E. (2020, September). Array programming with NumPy. *Nature*, *585*(7825), 357–362. Retrieved from https://doi.org/10.1038/s41586-020-2649-2 doi: 10.1038/s41586-020-2649-2

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., ... Stoyanov, V. (2019). *Roberta: A robustly optimized bert pretraining approach.*

Ma, X., Wang, Z., Ng, P., Nallapati, R., & Xiang, B. (2019). Universal text representation from bert: An empirical study. *arXiv preprint arXiv:1910.07973*.

Pibiri, G. E., & Venturini, R. (2020). Techniques for inverted index compression. *ACM Comput. Surv.*, *53*(6). doi: 10.1145/3415148