Information Retrieval course - Mini Project 2 documentation

Kimia Esmaili - 610398193

To make our system, we will follow these steps:

1. **Preprocessing and Stopword Removal:**
    - The system performs text preprocessing by removing special characters, converting text to lowercase, tokenizing the documents, and removing stopwords.
    - Stopwords are common words (e.g., "is", "the", "and") that do not carry much meaning and can be safely ignored during retrieval.
    - By removing stopwords, the code improves the quality of the inverted index and reduces noise in the search results.

2. **Inverted Indexing:**
    - The code builds an inverted index, mapping each word to the list of document indices where it appears.
    - This indexing technique improves the efficiency of searching and retrieval operations by providing direct access to relevant documents based on query terms.
    - The inverted index allows for faster retrieval of documents that contain specific words or terms, reducing the search space and improving performance.

3. Spelling Correction:

- The code includes a function for spelling correction using the Levenshtein distance.
- It calculates the distance between the input query and each word in the word list and returns the closest word.
- This feature enhances the system by suggesting or automatically correcting misspelled queries, making the search more user-friendly and accurate.

4. Wildcard Queries:

- The code handles wildcard queries by searching for terms that match the query pattern using prefix and suffix matching.
- It supports queries with one or two * symbols.
- By implementing this functionality, the system allows users to perform more flexible and expressive searches, accommodating partial or variable terms in the query.

Index Optimization:

- The methods explained were enhancements that aim to improve retrieval accuracy, search flexibility, and user experience. However, it's important to note that the code provided is a simplified implementation and may not include advanced techniques like Permuterm or K-gram indexing. These techniques can further optimize wildcard query handling.
- Implementing more advanced distance metrics for spelling correction, such as Damerau-Levenshtein distance or Jaro-Winkler distance. These metrics can provide better suggestions for misspelled queries.
- Optimizing the wildcard query handling by using indexing techniques like Permuterm or K-gram indexing. These techniques allow for faster pattern matching and can reduce the search space.
- Implementing ranking algorithms like TF-IDF or BM25 to improve the relevance of search results based on query terms and document statistics.
- Enhancing the system's robustness by implementing error handling and input validation to handle edge cases and prevent unexpected behavior.
- Using caching mechanisms to store frequently accessed data like the inverted index or the results of expensive computations, improves the system's performance.

Code Explanation:

- The code starts by reading and preprocessing the text documents using similar steps as mentioned in the previous question.
- Next, an inverted index is built, mapping each word to the list of document indices where it appears.
- The spelling_correction function takes an input query and finds the closest word in the word_list using Levenshtein distance.
- The wildcard_query function handles wildcard queries by searching for terms that match the query pattern using a combination of prefix and suffix matching.
- Example usage demonstrates the application of spelling correction and wildcard queries on the given documents.
- Note that this code assumes that the word_list provided contains all the words present in the documents. Additionally, further enhancements and optimizations can be made based on specific requirements and use cases, such as using more advanced distance metrics for spelling correction or implementing more efficient wildcard query matching techniques like Permuterm or K-gram indexing.
- There are comments in the code on what every part does, so it is redundant to show how every part and function of the code works in greater detail again in this document.
- To handle Spelling Correction, Wildcard Queries, and make enhancements to the Information Retrieval system, we can follow the following steps:

- Spelling Correction:
  - Implement a function that takes an input query and a word list as parameters.
  - Iterate through the word list and calculate the Levenshtein distance (edit distance) between each word and the input query.
  - Keep track of the word with the minimum distance and return it as the closest word to the query.
  - This function can be used to provide suggestions or correct misspelled queries, improving the user experience.

- Wildcard Queries:
  - Implement a function that handles wildcard queries using techniques like Permuterm or K-gram indexing.
  - The function should be able to handle queries with one or two * symbols.
  - Preprocess the query by removing punctuation and converting it to lowercase.
  - Split the query into query terms.
  - For each query term, check if it contains a wildcard symbol (*).
  - If it does, perform prefix and suffix matching on the inverted index to find terms that match the query pattern.
  - Collect the matching document indices and return them as the result.

➢ To improve accuracy, a post-filtering step can be added to remove false positive outcomes by considering the context or additional constraints.