WikiSearch
RetrievalRangers

Berk Güler
berk.guler@metu.edu.tr

Mustafa Barış Emektar
baris.emektar@metu.edu.tr

Abstract

In this project, it is aimed to construct an ad-hoc information retrieval system using the $\text{ir}_{datasets}$: $DPRWiki100$ dataset by utilizing PyTerrier framework. Initially, BM25 ranking scheme will be employed the for retrieval. However, its performance will be enhanced by incorporating optimizations. Additionally, the system will feature a user interface, providing an intuitive experience for users interacting with the retrieved information.

1 Introduction

The information retrieval system will be designed to take a user query as input and return a set of relevant documents from a given collection. It aims to efficiently retrieve relevant information based on the user's search intent.

- Indexing: Creating an index of your document collection (in this case, the DPR Wiki100 dataset).
- Query Processing: Implementing query parsing, tokenization, stemming, and removing stop words.
- Term Frequency-Inverse Document Frequency (TF-IDF): Calculating TF-IDF scores for terms in documents and queries.
- Vector Space Models: Using cosine similarity or other similarity measures to rank documents based on query relevance.
- Document Preprocessing: Applying standard preprocessing steps (e.g., stop-word removal, stemming) to improve retrieval quality.

The dataset consists of 6.5K queries, 21M docs, and 980K query relevance judgments.

2 Overview of the Proposed System

- **Data parsing and indexing:** The text will be tokenized (splited it into individual words) using a suitable tokenizer. Apply lowercasing to ensure case-insensitive matching. Remove common stop words (e.g., "the," "and," "in") that don't significantly contribute to meaning. Optionally, perform stemming or lemmatization to reduce words to their root form. Our document collection consists of passages from the DPR Wiki100 dataset. Each passage will be treated as a separate document. PyTerrier framework will be used to create the inverted index. PyTerrier will automatically handle the mapping of terms to documents. The index will store additional information such as term frequency (TF) and document IDs. Moreover, positional index and permuterm index will be implemented.
- **Query processing:** The system will support free-text queries where users can input natural language queries. Users can express their information needs without strict syntax requirements. User queries will be parsed to extract relevant terms. Tokenization and preprocessing (similar to document preprocessing) will be applied to queries. Case-insensitivity and remove stop words will be handeled. Our system will support the following query types: Boolean Queries: Users can combine terms using logical operators (AND, OR, NOT). Wildcards: Users can use wildcards (e.g., "appl*") to match variations of a term. Phrase Queries: Users can search for exact phrases (e.g., "machine learning").
- **Ranking and retrieval**: The ranking implementation will start with a simple VSM using BM25 as our baseline. Then, our baseline system will be enhanced . The following components may be used to improve the ranking system:
  - **Pseudo-Relevance Feedback (PRF)**: After retrieving an initial set of documents using BM25, the top-ranked documents will be analyzed. Extract relevant terms from these documents and expand the original query. Re-rank the results based on the expanded query.
  - The vectors for queries and documents can be produced using:
    - **Word2Vec**: Learn dense vector representations for words.
    - **Sent2Vec**: Extend Word2Vec to learn vector representations for sentences.
    - **Doc2Vec**: Extend Word2Vec to learn document embeddings.

    If there is not enough time in the project schedule, the pre-trained word embeddings method can be used instead.
- **User Interface**: A user interface that is useful and appealing to the eye will be designed. In this user interface, settings to accommodate different search methods will be presented to the user. The user interface will be designed using python since the integration cost will be low.