

Information Retrieval Assignment 1

Dhyan Vimalkumar Patel

Roll No. 2021041

February 9, 2024

Introduction

Information Retrieval (IR) systems play a pivotal role in extracting relevant information from vast datasets. This assignment ventures into constructing core IR system components, including data preprocessing, positional and unigram inverted index creation, as well as the management of complex Boolean and Phrase queries. It encapsulates the practical enactment of IR theories, emphasizing the extraction and organization of meaningful insights from unstructured datasets.

1 Data Preprocessing

Libraries Used

The preprocessing stage utilized one primary library:

- **nltk**: for natural language processing tasks such as tokenization and stopwords removal.

Preprocessing Steps

The preprocessing pipeline consisted of several key steps:

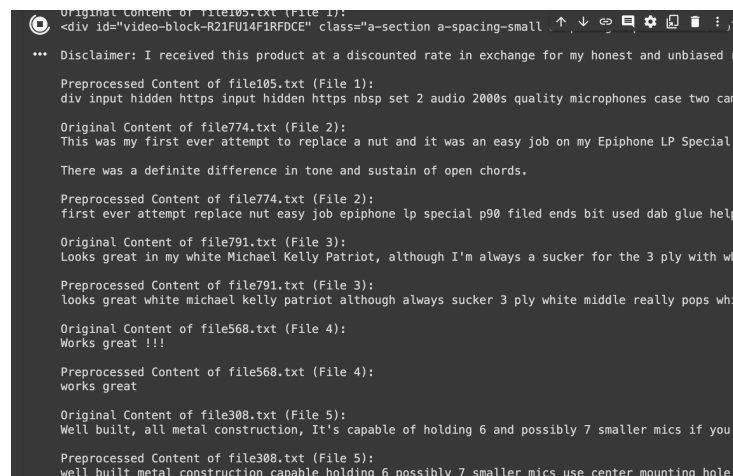
1. **Conversion to Lowercase**: Normalizing the case of all text to lowercase to ensure consistency in processing.

2. **Tokenization:** Using NLTK's *word_tokenize* function to split text into individual words.
3. **Stopwords Removal:** Filtering out common words that offer little value in querying using NLTK's predefined list of stopwords.
4. **Punctuation Removal:** Discarding non-alphanumeric characters to focus on meaningful tokens.
5. **Elimination of Blank Spaces:** Removing tokens that consist solely of whitespace, which are artifacts of processing.

Functionality

The core of the preprocessing stage was encapsulated in two functions:

- **preprocess_text:** Applies the aforementioned preprocessing steps to any given input text, transforming raw data into a cleaned, standardized format ready for indexing.
- **preprocessed_folder:** Automates the preprocessing of a collection of text files within a specified directory, saving the cleaned text for subsequent indexing. This function also includes a mechanism to log the transformation of the first five files processed, providing insight into the effectiveness of the preprocessing steps.



```

Original Content of file105.txt (File 1):
*** Disclaimer: I received this product at a discounted rate in exchange for my honest and unbiased r

Preprocessed Content of file105.txt (File 1):
div input hidden https input hidden https nbsp set 2 audio 2000s quality microphones case two can

Original Content of file774.txt (File 2):
This was my first ever attempt to replace a nut and it was an easy job on my Epiphone LP Special

There was a definite difference in tone and sustain of open chords.

Preprocessed Content of file774.txt (File 2):
first ever attempt replace nut easy job epiphone lp special p90 filed ends bit used dab glue help

Original Content of file791.txt (File 3):
Looks great in my white Michael Kelly Patriot, although I'm always a sucker for the 3 ply with wh

Preprocessed Content of file791.txt (File 3):
looks great white michael kelly patriot although always sucker 3 ply white middle really pops wh

Original Content of file568.txt (File 4):
Works great !!!

Preprocessed Content of file568.txt (File 4):
works great

Original Content of file308.txt (File 5):
Well built, all metal construction, It's capable of holding 6 and possibly 7 smaller mics if you

Preprocessed Content of file308.txt (File 5):
well built metal construction capable holding 6 possibly 7 smaller mics use center mounting hole

```

Figure 1: Data Preprocessing Output

2 Unigram Inverted Index and Boolean Queries

Objectives

- Implement a unigram inverted index based on preprocessed data, enabling efficient document retrieval by term.
- Use Python's `pickle` module for index serialization and deserialization, ensuring long-term storage and quick access.
- Support complex Boolean queries involving AND, OR, and NOT operations to allow for versatile information retrieval.
- Process queries and output results in a specified format, applying pre-processing to query inputs for consistent text analysis.

Functionality and Implementation

1. **Inverted Index Creation:** By iterating through preprocessed text files, this process tokenizes content and meticulously maps each term to documents containing it. This meticulous mapping is crucial for the rapid retrieval of documents based on specific terms.
2. **Index Serialization and Deserialization:** Utilizes `pickle` to efficiently store the index on disk and retrieve it, significantly reducing the system's startup time and resource consumption.
3. **Boolean Query Execution:** This step involves preprocessing query terms to ensure uniformity with the indexed data, parsing Boolean operations to understand the query logic, and executing these operations to dynamically filter and combine document sets. This flexibility allows users to perform detailed searches.
4. **Query Processing:** Comprehensive preprocessing of inputs, strategic execution of the query against the index, and meticulous formatting of results underscore the system's capability to handle complex queries and present results in an intuitive manner.

```
53 # Now, loaded_unigram_inverted_index contains the previously saved index and then printing it
54 print(f"Unigram inverted index loaded from {input_index_path}")
55 print(loaded_unigram_inverted_index)

Unigram inverted index saved to /content/drive/MyDrive/IR_Assignments/unigram_inverted_index/uni
Unigram inverted index loaded from /content/drive/MyDrive/IR_Assignments/unigram_inverted_index/
{'asin': ['processed_file184.txt', 'processed_file742.txt'], 'exact': ['processed_file184.txt',
```

Figure 2: Unigram Inverted Index Output

```
Enter the number of queries: 2
Enter query 1: Car in a Canister
Enter operations: OR
Enter query 2: Fits can CAR in a Canister
Enter operations: OR, OR
Query 1: car OR canister
Number of documents retrieved: 6
Documents: processed_file166.txt, processed_file174.txt, processed_file264.txt, processed_file542
Query 2: fits OR car OR canister
Number of documents retrieved: 66
Documents: processed_file107.txt, processed_file110.txt, processed_file118.txt, processed_file160
```

Figure 3: Boolean Query Output

3 Positional Index and Phrase Queries

Objectives

- Construct a positional index from the dataset obtained after preprocessing, designed to support the retrieval of documents based on the exact position of terms.
- Utilize Python's `pickle` module for index serialization and deserialization, facilitating efficient management and usage of the positional index.
- Facilitate phrase query execution with specific input and output formats, adhering to preprocessing steps to ensure that queries are analyzed in a manner consistent with the indexed data.

Functionality and Implementation

1. **Positional Index Creation:** This process not only maps terms to the documents but also records their positions within each document. This detailed mapping is pivotal for executing phrase queries, where the specific order of terms is essential for retrieving relevant documents.

2. **Index Persistence:** The use of `pickle` for index persistence underscores the technical strategy for ensuring that the positional index is readily available for query processing, without the need for regeneration.
3. **Phrase Query Execution:** The system's ability to process phrase queries by examining consecutive term positions within documents highlights its advanced capability to support complex search requirements.
4. **Query Processing:** Tailoring the processing of user inputs to conform to predefined formats, and the thoughtful presentation of query results, emphasizes the system's user-centric approach. This ensures that users can easily interpret the information retrieved by their queries.

Figure 4: Positional Index Output

Figure 5: Phrase Query Output