CSE508 Information Retrieval
Assignment 1
Report

Ishwar Babu - 2021532

1 Introduction

This report discusses three Python scripts designed for text preprocessing, creation and usage of an inverted index for boolean queries, and implementation of a positional index for phrase queries within a dataset of text files. These scripts utilize various data structures and libraries to efficiently process and query textual data.

2 Data Preprocessing

2.1 Overview

The data preprocessing script is responsible for preparing text data for further analysis or processing. This involves converting text to lowercase, removing HTML tags, tokenizing, removing stopwords and punctuation, and eliminating blank space tokens.

2.2 Functionalities

- **Text Lowercasing:** Converts all text to lowercase to ensure consistency in processing.
- **HTML Tag Removal:** Uses BeautifulSoup to remove HTML tags, leaving only the text content.
- **Tokenization:** Splits the text into individual words using NLTK's word_tokenize function.
- **Stopwords Removal:** Filters out common words (e.g., "the", "is", "in") that do not add much meaning to the text.
- **Punctuation Removal:** Eliminates tokens that are not alphanumeric to focus on meaningful words.
- **Preprocessing and Saving Files:** Reads files from a given directory, preprocesses the text, and saves the preprocessed text to a new directory.
- **Random Sample Display:** Selects a random subset of preprocessed files to showcase the preprocessing results.

2.3 Structures Used

- The script uses sets for stopwords to optimize the search and removal process.
- Lists are used to store tokens and file names for iterative processing.

3 Unigram Inverted Index and Boolean Queries

3.1 Overview

This script creates an inverted index from the preprocessed text files, allowing efficient querying of the dataset with boolean operations (AND, OR, AND NOT, OR NOT).

3.2 Functionalities

- **Inverted Index Creation:** Builds an inverted index where each word points to a set of documents containing that word.
- **Index Persistence:** Saves and loads the inverted index using Python's pickle module for persistent storage.
- **Boolean Query Processing:** Performs boolean queries on the inverted index to find documents that match the query criteria.
- **Query Formatting:** Formats and displays the queries along with their results.

3.3 Structures Used

- Utilizes defaultdict from the collections module to easily append files to each word entry in the index.
- Sets are used to store file names for each word, supporting efficient boolean operations.

4 Positional Index and Phrase Queries

4.1 Overview

Implements a positional index for processing phrase queries. This index tracks the positions of each word within documents, enabling precise phrase query matching.

4.2 Functionalities

- **Positional Index Creation:** Constructs a positional index where each word is associated with a dictionary. This dictionary maps documents to lists of positions where the word occurs.
- **Index Persistence:** Uses pickle to save and load the positional index for reuse.
- **Phrase Query Processing:** Identifies documents containing the exact sequence of words in a phrase query by comparing positions.

4.3 Structures Used

- The positional index is a nested defaultdict structure, with the outer level mapping words to documents and the inner level mapping documents to lists of word positions.
- Lists are used to store word positions within documents, facilitating the identification of consecutive word sequences.

5 Conclusion

The presented scripts offer a comprehensive approach to preprocessing text data, querying datasets with boolean logic, and executing precise phrase searches using a positional index. Each component plays a crucial role in text analysis and information retrieval systems, demonstrating the effectiveness of Python and its libraries in processing and searching textual data.

```
Processing file: file577.txt

Original text:
-----
Perfect fit for 2001 Mexican Standard Telecaster.  Two protective layers of plastic coating.  Looks great! ...
-----

After removing HTML tags:
-----
perfect fit for 2001 mexican standard telecaster.  two protective layers of plastic coating.  looks great! ...
-----

After tokenization:
-----
['perfect', 'fit', 'for', '2001', 'mexican', 'standard', 'telecaster', '.', 'two', 'protective', 'layers', 'of', 'plastic', 'coating', '.', 'looks', 'great', '!'] ...
-----

After removing stop words:
-----
['perfect', 'fit', '2001', 'mexican', 'standard', 'telecaster', '.', 'two', 'protective', 'layers', 'plastic', 'coating', '.', 'looks', 'great', '!'] ...
-----

After removing non-alphanumeric characters:
-----
['perfect', 'fit', '2001', 'mexican', 'standard', 'telecaster', 'two', 'protective', 'layers', 'plastic', 'coating', 'looks', 'great'] ...
-----

Final preprocessed text:
-----
perfect fit 2001 mexican standard telecaster two protective layers plastic coating looks great ...
-----
```

Figure 1: Example output of the Data Preprocessing script.

```
<ipython-input-4-3025ac412035>:12: MarkupResemblesLocatorWarning: The input looks more like a filename than markup. You may want to open this file and pass the filehandle into Beautiful Soup.
  soup = BeautifulSoup(text, 'html.parser')
Enter the number of queries: 2
Enter query 1: Car bag in a canister
Enter operations for query 1, separated by commas: OR, AND NOT
Enter query 2: Coffee brewing techniques in cookbook
Enter operations for query 2, separated by commas: AND, OR NOT, OR

Query 1: car OR bag AND NOT canister
Number of documents retrieved for query 1: 31
Names of the documents retrieved for query 1: preprocessed_file682.txt, preprocessed_file686.txt, preprocessed_file118.txt, preprocessed_file698.txt, preprocessed_file166.txt, preprocessed_fil

Query 2: coffee AND brewing OR NOT techniques OR cookbook
Number of documents retrieved for query 2: 0
Names of the documents retrieved for query 2:
```

Figure 2: Example output of the Inverted Index script.

```
<ipython-input-4-3025ac412035>:12: MarkupResemblesLocatorWarning: The input looks more like a filename than markup. You may want to open this file and pass the filehandle into Beautiful Soup.
  soup = BeautifulSoup(text, 'html.parser')
Enter the number of queries: 3
Enter phrase query 1: it is a good in front for poutch
Number of documents retrieved for query 1 using positional index: 0
Names of documents retrieved for query 1 using positional index:
Enter phrase query 2: it is good in reliable for fit
Number of documents retrieved for query 2 using positional index: 1
Names of documents retrieved for query 2 using positional index: preprocessed_file9.txt
Enter phrase query 3: it is a fit front poutch
Number of documents retrieved for query 3 using positional index: 1
Names of documents retrieved for query 3 using positional index: preprocessed_file9.txt
```

Figure 3: Example output of the Positional Index script.