Aruba Sood
MT23022

CSE508_Winter2024_A1_Report

QUESTION NO.1

1. Introduction:

The preprocessing steps are applied to text files using Python code. The objective is to prepare the text data for further analysis by performing tasks such as lowercasing, tokenization, stopwords removal, punctuation removal, and blank tokens removal

2. Approach:

- The approach involves reading each text file from a dataset directory, applying a series of preprocessing steps, and saving the preprocessed files to a designated directory.
- The preprocessing steps include lowercasing the text, tokenizing it, removing stopwords, punctuation marks, and blank tokens.

3. Methodologies:

- Lowercasing: Convert text to lowercase using the `lowercase_text` function.
- Tokenization: Split text into tokens using NLTK's `word_tokenize` function.
- Stopwords Removal: Remove common stopwords from the tokens using NLTK's stopwords corpus.
- Punctuation Removal: Eliminate punctuation marks from the tokens using Python's `string.punctuation`.
- Blank Tokens Removal: Remove any remaining blank tokens after other preprocessing steps using the `remove_blank_tokens` function.

4. Assumptions:

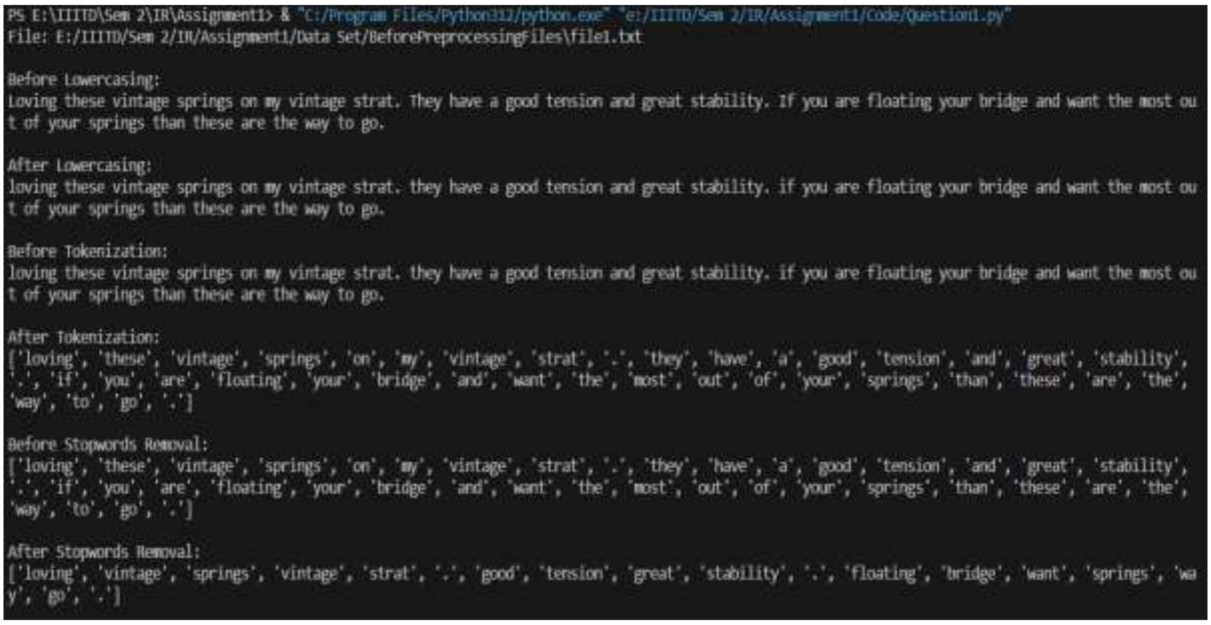
- The text files are assumed to be in English.
- The NLTK stopwords corpus is used for English stopwords removal.

5. Results:

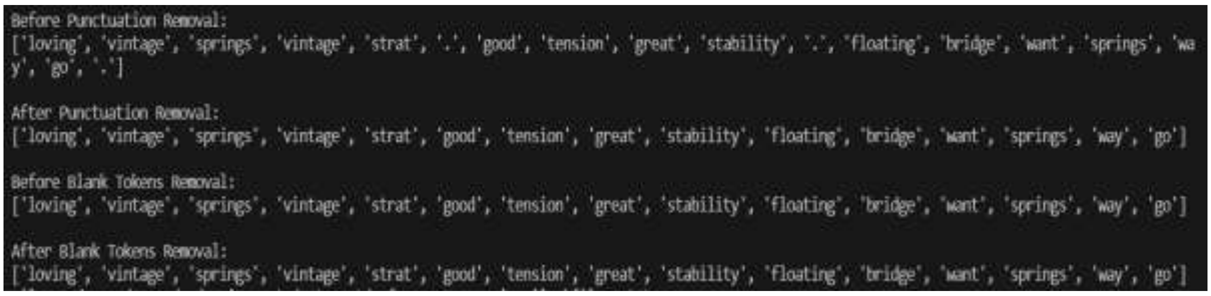
- The preprocessing steps were applied to each text file in the dataset.
- Samples of original text and preprocessed text were compared to evaluate the effectiveness of each preprocessing step.
- Observations:
    - Lowercasing: Reduced the complexity by standardizing the text to lowercase.
    - Tokenization: Split text into meaningful tokens, facilitating further analysis.
    - Stopwords Removal: Eliminated common stopwords, reducing noise in the data.
    - Punctuation Removal: Streamlined the text by removing unnecessary punctuation marks.
    - Blank Tokens Removal: Ensured that only meaningful tokens were retained for analysis.

Aruba Sood
MT23022

Screenshots:



```
PS E:\IIITD\Sem 2\IR\Assignment1> & "C:/Program Files/Python312/python.exe" "e:/IIITD/Sem 2/IR/Assignment1/Code/Question1.py"
File: E:/IIITD/Sem 2/IR/Assignment1/Data Set/BeforePreprocessingFiles\file1.txt

Before Lowercasing:
Loving these vintage springs on my vintage strat. They have a good tension and great stability. If you are floating your bridge and want the most ou
t of your springs than these are the way to go.

After Lowercasing:
loving these vintage springs on my vintage strat. they have a good tension and great stability. if you are floating your bridge and want the most ou
t of your springs than these are the way to go.

Before Tokenization:
loving these vintage springs on my vintage strat. they have a good tension and great stability. if you are floating your bridge and want the most ou
t of your springs than these are the way to go.

After Tokenization:
['loving', 'these', 'vintage', 'springs', 'on', 'my', 'vintage', 'strat', '.', 'they', 'have', 'a', 'good', 'tension', 'and', 'great', 'stability',
'.', 'if', 'you', 'are', 'floating', 'your', 'bridge', 'and', 'want', 'the', 'most', 'out', 'of', 'your', 'springs', 'than', 'these', 'are', 'the',
'way', 'to', 'go', '.']

Before Stopwords Removal:
['loving', 'these', 'vintage', 'springs', 'on', 'my', 'vintage', 'strat', '.', 'they', 'have', 'a', 'good', 'tension', 'and', 'great', 'stability',
'.', 'if', 'you', 'are', 'floating', 'your', 'bridge', 'and', 'want', 'the', 'most', 'out', 'of', 'your', 'springs', 'than', 'these', 'are', 'the',
'way', 'to', 'go', '.']

After Stopwords Removal:
['loving', 'vintage', 'springs', 'vintage', 'strat', '.', 'good', 'tension', 'great', 'stability', '.', 'floating', 'bridge', 'want', 'springs', 'wa
y', 'go', '.']
```



```
Before Punctuation Removal:
['loving', 'vintage', 'springs', 'vintage', 'strat', '.', 'good', 'tension', 'great', 'stability', '.', 'floating', 'bridge', 'want', 'springs', 'wa
y', 'go', '.']

After Punctuation Removal:
['loving', 'vintage', 'springs', 'vintage', 'strat', 'good', 'tension', 'great', 'stability', 'floating', 'bridge', 'want', 'springs', 'way', 'go']

Before Blank Tokens Removal:
['loving', 'vintage', 'springs', 'vintage', 'strat', 'good', 'tension', 'great', 'stability', 'floating', 'bridge', 'want', 'springs', 'way', 'go']

After Blank Tokens Removal:
['loving', 'vintage', 'springs', 'vintage', 'strat', 'good', 'tension', 'great', 'stability', 'floating', 'bridge', 'want', 'springs', 'way', 'go']
```

QUESTION NO. 2

1. Introduction:

This report presents the implementation of a unigram inverted index and support for boolean queries using Python code. The dataset obtained from the preprocessing steps performed in Question 1 is used for building the inverted index and executing boolean queries.

2. Approach:

- The approach involves building a unigram inverted index from scratch without using any external libraries.
- Python's `pickle` module is utilized to save and load the inverted index.
- Support for various boolean operations such as AND, OR, AND NOT, and OR NOT is provided for querying the inverted index.
- Preprocessing steps similar to those performed in Question 1 are applied to both the dataset and input queries.

Aruba Sood
MT23022

3. Methodologies:

- Inverted Index Construction:
- The inverted index is constructed by iterating through each preprocessed document in the dataset.
- For each token in the document, the corresponding posting list is updated in the inverted index.
- Boolean Query Evaluation:
- Boolean queries are processed by evaluating each term in the query against the inverted index.
- Operations are performed based on the specified boolean operators, and the result set of documents is computed accordingly.

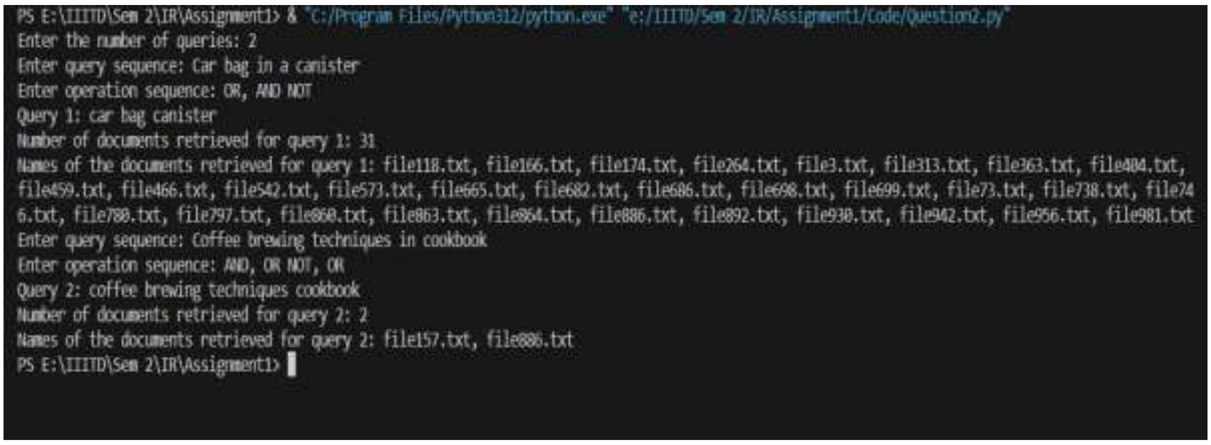
4. Assumptions:

- The input queries are assumed to be in a predefined format, with terms and operations separated by comma.
- Only boolean operations specified in the assignment (AND, OR, AND NOT, OR NOT) are supported.

5. Results:

- The inverted index is successfully built from the preprocessed dataset and saved using the `pickle` module.
- Queries are executed against the inverted index, and the results are displayed as per the specified format.
- Sample results are provided to illustrate the functionality of the boolean query system.

Screenshots:



```
PS E:\IIITD\Sem 2\IR\Assignment1> & "C:/Program Files/Python312/python.exe" "e:/IIITD/Sem 2/IR/Assignment1/Code/Question2.py"
Enter the number of queries: 2
Enter query sequence: Car bag in a canister
Enter operation sequence: OR, AND NOT
Query 1: car bag canister
Number of documents retrieved for query 1: 31
Names of the documents retrieved for query 1: file118.txt, file166.txt, file174.txt, file264.txt, file3.txt, file313.txt, file363.txt, file404.txt,
file459.txt, file466.txt, file542.txt, file573.txt, file665.txt, file682.txt, file686.txt, file698.txt, file699.txt, file73.txt, file738.txt, file74
6.txt, file780.txt, file797.txt, file860.txt, file863.txt, file864.txt, file886.txt, file892.txt, file930.txt, file942.txt, file956.txt, file981.txt
Enter query sequence: Coffee brewing techniques in cookbook
Enter operation sequence: AND, OR NOT, OR
Query 2: coffee brewing techniques cookbook
Number of documents retrieved for query 2: 2
Names of the documents retrieved for query 2: file157.txt, file886.txt
PS E:\IIITD\Sem 2\IR\Assignment1> 
```

Aruba Sood
MT23022

QUESTION NO. 3

1. Introduction:

This report outlines the implementation of a positional index and support for phrase queries using Python code. The dataset obtained from the preprocessing steps performed in Question 1 is used to build the positional index, which facilitates efficient retrieval of documents containing specific phrases.

2. Approach:

- The approach involves constructing a positional index from scratch without relying on external libraries.
- Python's `pickle` module is utilized to save and load the positional index for future use.
- Phrase queries are processed by evaluating each term in the query against the positional index to identify documents containing the specified phrases.

3. Methodologies:

- Positional Index Construction:
  - ·The positional index is constructed by iterating through each preprocessed document in the dataset.
  - ·For each token in the document, the corresponding posting list is updated in the positional index, along with the positions of each occurrence.
- Phrase Query Evaluation:
  - ·Phrase queries are processed by identifying documents where all terms in the query occur consecutively and in the correct order.
  - ·The presence of the entire phrase is verified by checking the positions of each term in the document.

4. Assumptions:

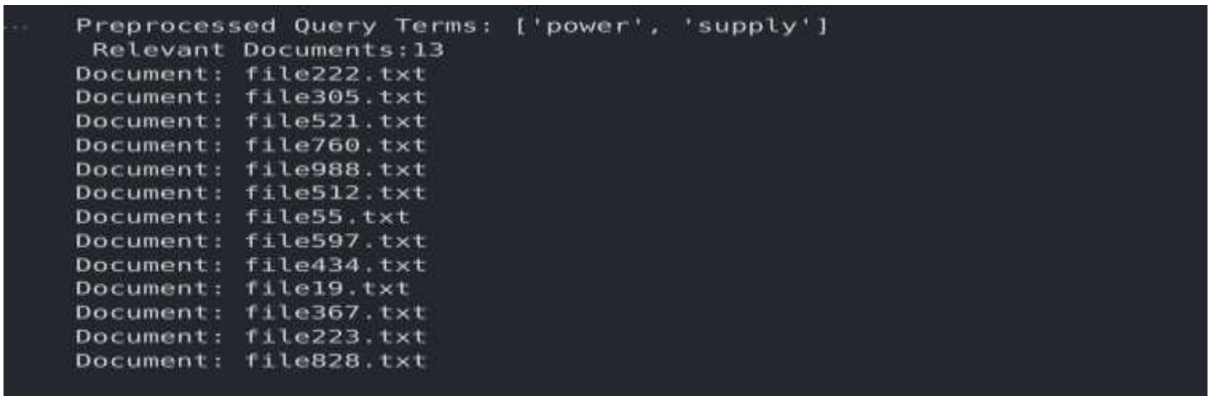
- The input phrase queries are assumed to be in a predefined format.
- Each query may contain multiple terms forming the desired phrase.

5. Results:

- The positional index is successfully built from the preprocessed dataset and saved using the `pickle` module.
- Phrase queries are executed against the positional index, and the results are displayed as per the specified format.
- Sample results are provided to illustrate the functionality of the phrase query system.

Aruba Sood
MT23022

Screenshot:



```
...    Preprocessed Query Terms: ['power', 'supply']
        Relevant Documents:13
       Document: file222.txt
       Document: file305.txt
       Document: file521.txt
       Document: file760.txt
       Document: file988.txt
       Document: file512.txt
       Document: file55.txt
       Document: file597.txt
       Document: file434.txt
       Document: file19.txt
       Document: file367.txt
       Document: file223.txt
       Document: file828.txt
```