Report

Question 1

Approach

- Created a function preprocess that takes the input text and a parameter to determine whether to print the intermediate results or not.
- Within the preprocess function, use the BeautifulSoup library to remove HTML tags from the text.
- Lowercase the text, tokenize it, remove stopwords, remove punctuation, and remove blank space tokens sequentially.
- If the show parameter is set to 'yes', print the actual content along with each preprocessing step's output.
- Save the processed token to an output file.

Methodology

- Lowercasing: Convert all text to lowercase to ensure consistency.
- Tokenization: Tokenize the text using the word_tokenize function from the NLTK library.
- Stopword Removal: Remove stopwords using the NLTK stopwords corpus.
- Punctuation Removal: If all characters in the token are punctuation characters, the token is excluded from the result list. If any character in the token is not punctuation, the token is retained..
- Blank Space Removal:Removed tokens consisting only of blank spaces.

Assumption

- The input text may contain HTML tags, which should be removed.
- The text may contain upper and lowercase letters both.
- Stopwords and punctuation need to be removed to get the actual words.
- Blank space should not be considered as a token.

Results

- It will be processed without stopwords , punctuations , blank space which will be written to an output file.

Shazra Irshad
MT23089

```
Actual Content:
I got this for my son for Christmas and he really loves it.  He also as a condenser mic that needs phantom power in order to work.  With this focusrite scarlett 2i2 Audio
I am very happy with this purchase and recommend it.


a. Lowercased Content:
i got this for my son for christmas and he really loves it.  he also as a condenser mic that needs phantom power in order to work.  with this focusrite scarlett 2i2 audio
i am very happy with this purchase and recommend it.

b. Tokenized Content:
['i', 'got', 'this', 'for', 'my', 'son', 'for', 'christmas', 'and', 'he', 'really', 'loves', 'it', '.', 'he', 'also', 'as', 'a', 'condenser', 'mic', 'that', 'needs', 'phan

c. Without Stopwords:
['got', 'son', 'christmas', 'really', 'loves', '.', 'also', 'condenser', 'mic', 'needs', 'phantom', 'power', 'order', 'work', '.', 'focusrite', 'scarlett', '2i2', 'audio',

d. Without Punctiation:
['got', 'son', 'christmas', 'really', 'loves', 'also', 'condenser', 'mic', 'needs', 'phantom', 'power', 'order', 'work', 'focusrite', 'scarlett', '2i2', 'audio', 'interfac

e. Without Blank:
['got', 'son', 'christmas', 'really', 'loves', 'also', 'condenser', 'mic', 'needs', 'phantom', 'power', 'order', 'work', 'focusrite', 'scarlett', '2i2', 'audio', 'interfac

Actual Content:
```

Question 2

Approach

- The create_unigram_inverted_index function constructs the index by iterating over each document.
- For each document, it tokenizes the content, sorts the tokens, and updates the index with the document frequency (DF) and the corresponding document IDs.
- The index is stored in memory as a sorted dictionary, where each term maps to a list containing its DF and a sorted set of document IDs.
- Load the unigram inverted index from a pickle file.
- Prompt the user to input the number of queries and the queries themselves.
- Preprocess the query strings and split the operator strings.
- Retrieve the document IDs corresponding to the terms in the query from the inverted index.
- Perform the specified Boolean operations on the retrieved document IDs.
- Print the results, including the number of documents retrieved and their names.

Methodology

- Load the index from a pickle file containing term information and document IDs.
- For each query:
    - Input the query string and operators.
    - Preprocess the query.
    - Retrieve document IDs for each term
    - Check if the term is present or not if not present return empty..
    - Apply Boolean operations (AND, OR, AND NOT, OR NOT) between document IDs.
    - Print the number and names of retrieved documents.

Assumption

Shazra Irshad
MT23089

- The number of operators between queries is one less than tokens.
- If not, the query is considered as invalid.
- If the term is not present in the unigram inverted index empty sortedset will be returned.
- The query is processed from left to right.

Results

- The loadQ2 function processes each query, applies the specified Boolean operations, and returns the set of document IDs satisfying the query conditions

```
[64] loadQ2()

    1
    Car bag in a canister
    OR, AND NOT
    Query 1:
    car or bag and not canister
    Number of documents retrieved for query 1: 31
    Names of the documents retrieved for query 1: ['file797.txt', 'file956.txt', 'file542.txt', 'file73.txt', 'file942.txt', 'file174.txt', 'file264.txt', 'fil
```

Question 3

Approach

- Iterate through each processed document.
- Tokenize the content and store the positions of each token in the positional index.
- Save the positional index to a pickle file.
- For each query, retrieve initial documents containing the first query term from the positional index.
- Check if subsequent query terms occur near the first term within a specified proximity.
- Build a list of valid documents satisfying the query conditions.
- Print the number and names of documents retrieved for each query.

Methodology

- The create_positional_index function constructs a positional index from the processed documents.
- It iterates over each document, tokenizes the content, and stores the positions of each token in the positional index.
- The index is represented as a nested dictionary, where each word maps to a dictionary of document IDs, and each document ID maps to a list of positions where the word occurs in that document.
- The positional index is then stored in memory and saved to a pickle file for future use.
- The loadQ3 function handles the processing of queries using the positional index.
- For each query:
    - It preprocesses the query string to lowercase and tokenize it.
    - It retrieves the initial set of documents containing the first query term from the positional index.
    - For each document, it checks if the subsequent query terms occur in proximity to the first term within a specified distance (position difference).

- If all query terms are found within the specified proximity in a document, the document is considered a match and added to the list of valid documents.
- Finally, it prints the number and names of documents retrieved for the query using the positional index.

Assumption

- Perform preprocessing on input sequence. So stopword will not be considered between the words.

Result

- The loadQ3 function accurately retrieves documents matching the query conditions using the positional index, providing relevant results based on term proximity.

```
[72] loadQ3()

    1
    fit right
    Number of documents retrieved for query 1 using positional index: 1
    Number of documents retrieved for query 1 using positional index: ['file3.txt']
```