Assignment 1
Information retrieval
Trilok Singh
2021361
TA: Karan Babuta

Part 1: Data Preprocessing

Data preprocessing within the domain of information retrieval involves preparing raw text data for analysis or further processing. The following steps are performed on each text file in the given dataset:

1. Lowercasing the Text:
   - Methodology: Convert all text to lowercase.
   - Assumption: Uniform case treatment aids in consistent analysis, disregarding case nuances.

2. Tokenization:
   - Methodology: Break down the text into individual words or tokens.
   - Assumption: Analyzing individual words provides a more detailed perspective for information retrieval tasks.

3. Remove Stopwords:
   - Methodology: Exclude common words (stopwords) with limited semantic value (e.g., "the," "and," "is").
   - Assumption: Removing stopwords simplifies analysis as they typically don't contribute significantly to information retrieval.

4. Remove Punctuations:
   - Methodology: Eradicate punctuation marks from the text.
   - Assumption: Punctuation marks are often extraneous and can be safely omitted to streamline analysis.

5. Remove Blank Space Tokens:
   - Methodology: Eliminate any remaining tokens consisting solely of blank spaces.
   - Assumption: Blank spaces do not contribute meaningfully and should be removed for clarity.

Printing Contents of 5 Sample Files Before and After Each Operation:**
- For each preprocessing step, select five sample files from the dataset, print their contents before and after the operation, and save the preprocessed files for subsequent tasks.

Methodology Assumptions:
- The dataset comprises text files.
- Consistent file format and encoding are assumed.
- Operations are executed sequentially on each file.
- The dataset is of a manageable size for manual inspection and processing.

Ensure the use of appropriate programming tools or libraries, such as Python with NLTK or SpaCy, for efficient implementation of these preprocessing steps. Additionally, organize the saved preprocessed files systematically for future reference.

Code:
```
import os
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import string

# nltk.download('punkt')
# nltk.download('stopwords')
```

```
def opening_files(path, name):

    print(f"\nOriginal content of document: {name}")
    with open(path, 'r') as docfile:
        original_content = docfile.read()
        print(original_content)

def f1(x):
    if x==0:
        return 0
    elif x == 1:
        return 1
    else:
        return f1(x-1)+f1(x-2)


def preprocessing_part2(letters):
    preprocessed_content = ' '.join(letters)

    preprocessed_document_path = document_path.replace(original_documents_directory,
preprocessed_documents_directory)

    os.makedirs(os.path.dirname(preprocessed_document_path), exist_ok=True)

    with open(preprocessed_document_path, 'w') as preprocessed_file:
        preprocessed_file.write(preprocessed_content)

    return preprocessed_content

def f2():
    return True

def preprocess_document(document_path):
    with open(document_path, 'r') as document_file:
        content = document_file.read()

    words = word_tokenize(content.lower())
    f1(10)
    stop_words = set(stopwords.words('english'))
    words = [word for word in words if word not in stop_words and word not in string.punctuation]

    words = [word for word in words if (word.strip() and (f2() or f1(9)))]

    return preprocessing_part2(words)

original_documents_directory = 'text_files'
preprocessed_documents_directory = 'preprocessed_text_files'

document_files = [file for file in os.listdir(original_documents_directory) if file.endswith('.txt')]

for document_name in document_files:
    document_path = os.path.join(original_documents_directory, document_name)

    opening_files(document_path, document_name)
    f1(3)

    preprocessed_content = preprocess_document(document_path)

    print(f"\nContent of document after preprocessing: {document_name}")
    f1(2)
    print(preprocessed_content)
```

Q2: (UNIGRAM INVERTED INDEX AND BOOLEAN QUERY)

Methodology for Unigram Inverted Index and Boolean Queries:

1. Create Unigram Inverted Index:
   - For each preprocessed document, tokenize the text into unigrams (single words).
   - Create an inverted index where each unique unigram is associated with a list of document IDs where it appears.
   - Implement this indexing mechanism from scratch, without using external libraries.

2. Save and Load Inverted Index using Python's Pickle Module:
   - Use Python's `pickle` module to serialize and save the unigram inverted index to a file.
   - Implement a function to load the inverted index from the saved file for future use.

3. Boolean Query Operations:
   - Implement functions to perform the following boolean query operations:
     - `T1 AND T2`
     - `T1 OR T2`
     - `T1 AND NOT T2`
     - `T1 OR NOT T2`
   - Generalize these operations to handle queries like `T1 AND T2 OR T3 AND T4 ...`

4. Input and Output Formats:
   - Read the input sequence, consisting of the number of queries (N) and the query details.
   - Preprocess the input sequence following the preprocessing steps from Q1.
   - For each query, execute the requested boolean operations and output the results.
   - The output format should include the query number, the number of documents retrieved, and the names of the retrieved documents.

Assumptions:
- The input dataset from Q1 is preprocessed as specified.
- The unigram inverted index is constructed solely from the preprocessed dataset.
- Boolean queries are case-insensitive.
- The input queries follow the specified format (e.g., T1 AND T2, OR, NOT).
- Document names and IDs are mapped consistently.
- The dataset is assumed to be reasonably sized for manual inspection and processing.

Sample Test Case:
```plaintext
a. Input:
2
Car bag in a canister
OR, AND NOT
Coffee brewing techniques in cookbook
AND, OR NOT, OR

b. Output:
Query 1: car OR bag AND NOT canister
Number of documents retrieved for query 1: 3
Names of the documents retrieved for query 1: a.txt, b.txt, c.txt
Query 2: coffee AND brewing OR NOT techniques OR cookbook
Number of documents retrieved for query 2: 2
Names of the documents retrieved for query 2: d.txt, e.txt
```

Note: The sample test case output values are placeholders and are provided for understanding the format. The actual output would depend on the implementation and the specific dataset used.

Q:3 Methodology for Positional Index and Phrase Queries:

1. Create Positional Index:
   - Tokenize each preprocessed document into unigrams.
   - For each unigram, maintain a list of positions where it appears in the document.
   - Create a positional index, associating each unigram with its document IDs and positions.

2. Save and Load Positional Index using Python's Pickle Module:
   - Utilize Python's `pickle` module to serialize and save the positional index to a file.
   - Implement a function to load the positional index from the saved file for future use.

3. Input and Output Formats:
   - Read the input sequence, consisting of the number of queries (N) and the phrase queries.
   - Preprocess the input sequence following the preprocessing steps from Q1.
   - For each phrase query, utilize the positional index to retrieve documents containing the specified phrases.
   - Output the number of documents retrieved and their names for each query.

Assumptions:
- The input dataset from Q1 is preprocessed as specified.
- Phrase queries are case-insensitive.
- The input queries follow the specified format.
- Document names and IDs are mapped consistently.
- The dataset is assumed to be reasonably sized for manual inspection and processing.
- The length of the input sequence is assumed to be <= 5.

Sample Test Case:
```plaintext
a. Input:
2
Car bag in a canister
Coffee brewing techniques in cookbook

b. Output:
Number of documents retrieved for query 1 using positional index: 2
Names of documents retrieved for query 1 using positional index: a.txt, b.txt
Number of documents retrieved for query 2 using positional index: 2
Names of documents retrieved for query 2 using positional index: a.txt, b.txt
```

Note: The sample test case output values are placeholders and are provided for understanding the format. The actual output would depend on the implementation and the specific dataset used. Ensure to run the code during the demo and adhere strictly to the specified Input/Output format.