CSE508_Winter_A1_2021354

Name : Shubham Attri
Roll No : 2021354
Github : shubham-attri
TA : Karan

PART 1 (DATA PREPROCESSING)

Introduction

Text data preprocessing is an essential step in natural language processing tasks. It involves cleaning and transforming raw text into a format suitable for analysis. In this assignment, the goal of data preprocessing was to prepare text data for building unigram-based and positional inverted indexes.

Data Preprocessing Steps

The code performs the following data preprocessing steps:

1. **Lowercasing:** All text is converted to lowercase. This ensures consistency in token matching.
2. **Tokenization:** The text is broken down into individual tokens (words) using NLTK's word_tokenize function.
3. **Stopword Removal:** Common stopwords (e.g., 'and', 'the', 'is') that do not contribute much to the meaning of the text are removed.
4. **Punctuation Removal:** Punctuation marks are eliminated to focus on the essential content of the text.
5. **Blank Space Token Removal:** Any tokens consisting solely of whitespace characters are removed.

Methodologies

The code utilizes Python's NLTK (Natural Language Toolkit) library for tokenization and stopwords removal. Custom functions are implemented for lowercase conversion, punctuation removal, and blank space token removal.

Assumptions

- The dataset contains English text.
- Standard English stopwords provided by NLTK are appropriate for removal.
- Punctuation marks can be safely removed without losing critical information.
- Tokens consisting only of whitespace characters are not relevant for analysis.

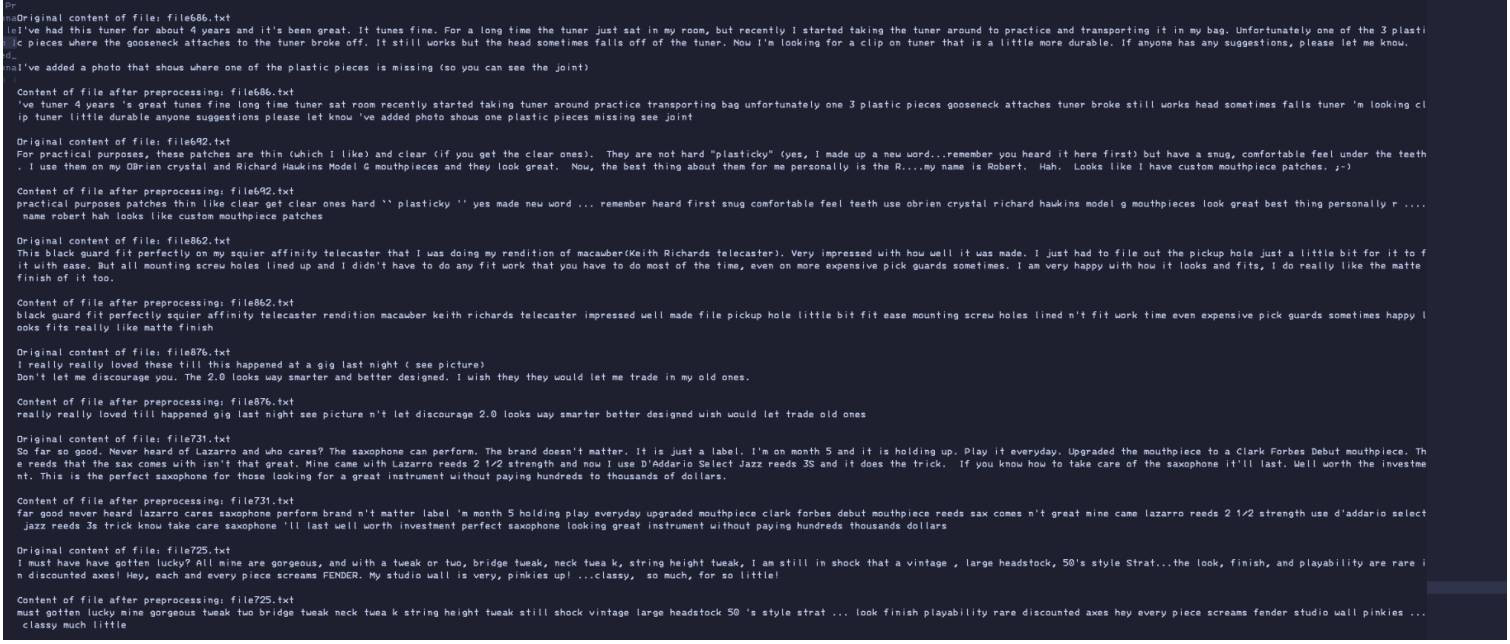
Results

The data preprocessing steps result in clean, standardized text data suitable for building inverted indexes. By removing stopwords and punctuation, the focus is shifted to content words, potentially improving the quality of the indexes.

Code Walkthrough

1. The necessary NLTK resources (punkt for tokenization and stopwords for stopword removal) are downloaded.
2. The `preprocess_text` function is defined. It takes a file path as input, reads the file's content, applies the preprocessing steps, and saves the preprocessed text to a new file in the `preprocessed_text_files` directory.
3. The script lists all text files in the `text_files` directory.
4. For each file, the script prints the original content, calls the `preprocess_text` function to preprocess the content, and then prints the preprocessed content.

```python
import os
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import string

# Download NLTK resources
nltk.download('punkt')
nltk.download('stopwords')

def preprocess_text(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        text = file.read()

    # Lowercasing
    text = text.lower()

    # Tokenization
    tokens = word_tokenize(text)

    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    tokens = [token for token in tokens if token not in stop_words]

    # Remove punctuations
    tokens = [token for token in tokens if token not in string.punctuation]

    # Remove blank space tokens
    tokens = [token for token in tokens if token.strip()]

    # Save preprocessed text
    preprocessed_text = ' '.join(tokens)
    preprocessed_file_path = file_path.replace(dataset_directory, preprocessed_directory)
    os.makedirs(os.path.dirname(preprocessed_file_path), exist_ok=True)

    with open(preprocessed_file_path, 'w', encoding='utf-8') as preprocessed_file:
        preprocessed_file.write(preprocessed_text)

    return preprocessed_text

# List all text files in the dataset directory
dataset_directory = 'text_files'
preprocessed_directory = 'preprocessed_text_files'
text_files = [file for file in os.listdir(dataset_directory) if file.endswith('.txt')]

# Print contents of all files before and after preprocessing
for file_name in text_files:
    file_path = os.path.join(dataset_directory, file_name)
    print(f"\nOriginal content of file: {file_name}")
    with open(file_path, 'r', encoding='utf-8') as file:
        original_text = file.read()
        print(original_text)

    preprocessed_text = preprocess_text(file_path)

    print(f"\nContent of file after preprocessing: {file_name}")
    print(preprocessed_text)
```



References

https://www.analyticsvidhya.com/blog/2021/06/text-preprocessing-in-nlp-with-python-codes/
https://www.geeksforgeeks.org/text-preprocessing-in-python-set-1/

PART 2 (UNIGRAM INVERTED INDEX AND BOOLEAN QUERY)

Introduction

This code is a Python script for creating and querying an inverted index, which is a data structure used in information retrieval systems to quickly find documents containing specific words or phrases. The script performs text preprocessing, creates an inverted index from a set of text files, and then allows the user to enter queries to retrieve relevant documents.

Data Preprocessing

The `preprocess_text` function performs the following steps on the input text:

1. **Lowercasing:** Convert all text to lowercase for consistency in token matching.
2. **Tokenization:** Break down the text into individual tokens (words) using NLTK's `word_tokenize` function.
3. **Stopword Removal:** Remove common stopwords (e.g., 'and', 'the', 'is') that do not contribute much to the meaning of the text.
4. **Punctuation Removal:** Eliminate punctuation marks to focus on the essential content of the text.
5. **Blank Space Token Removal:** Remove any tokens consisting solely of whitespace characters.

Inverted Index Creation

The `create_inverted_index` function creates an inverted index from a set of text files in a specified directory. It uses a defaultdict to map each token to the set of filenames containing that token.

Query Processing

The script provides functions to perform logical operations (AND, OR, AND NOT, OR NOT) on sets of filenames retrieved from the inverted index. The `execute_queries` function takes a list of queries and operations, processes them using these functions, and returns the results.

Input/Output Handling

The script provides an `input_format` function to prompt the user for the number of queries, the queries themselves, and the operations to be performed. The `preprocess_query` function preprocesses the input query text in the same way as the text files.

The `output_format` function prints the results of the queries, including the original query, the number of documents retrieved, and the names of the retrieved documents.

Main Function

The `main` function is the entry point of the script. It performs the following tasks:

1. Checks for an existing inverted index file. If it exists, it loads the index. Otherwise, it creates a new index and saves it to a file.
2. Prompts the user for input queries and operations.
3. Executes the queries using the inverted index.
4. Prints the results using the output format.

Results

The script provides a framework for creating and querying an inverted index, with the ability to perform various logical operations on the retrieved document sets. It can be used as a starting point for more complex information retrieval systems.

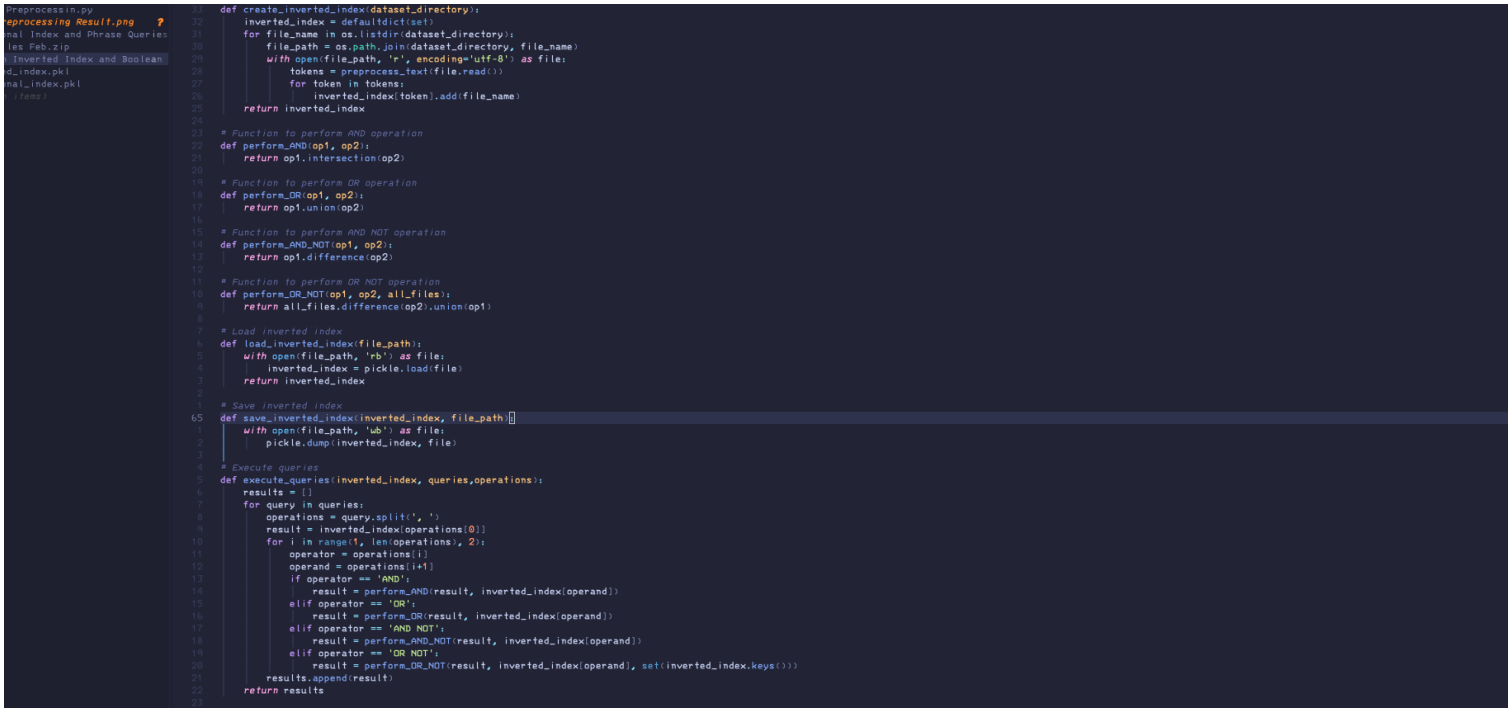
```
import os
import pickle
from collections import defaultdict
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import string

# Preprocessing function
def preprocess_text(text):

    # Lowercasing
    text = text.lower()

    # Tokenization
    tokens = word_tokenize(text)
```

```
    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    tokens = [token for token in tokens if token not in stop_words]

    # Remove punctuations
    tokens = [token for token in tokens if token not in string.punctuation]

    # Remove blank space tokens
    tokens = [token for token in tokens if token.strip()]

    return tokens

# Create unigram inverted index
def create_inverted_index(dataset_directory):
    inverted_index = defaultdict(set)
    for file_name in os.listdir(dataset_directory):
        file_path = os.path.join(dataset_directory, file_name)
        with open(file_path, 'r', encoding='utf-8') as file:
            tokens = preprocess_text(file.read())
            for token in tokens:
                inverted_index[token].add(file_name)
    return inverted_index

# Function to perform AND operation
def perform_AND(op1, op2):
    return op1.intersection(op2)

# Function to perform OR operation
def perform_OR(op1, op2):
    return op1.union(op2)

# Function to perform AND NOT operation
def perform_AND_NOT(op1, op2):
    return op1.difference(op2)

# Function to perform OR NOT operation
def perform_OR_NOT(op1, op2, all_files):
    return all_files.difference(op2).union(op1)

# Load inverted index
def load_inverted_index(file_path):
    with open(file_path, 'rb') as file:
        inverted_index = pickle.load(file)
```

```
    return inverted_index

# Save inverted index
def save_inverted_index(inverted_index, file_path):
    with open(file_path, 'wb') as file:
        pickle.dump(inverted_index, file)

# Execute queries
def execute_queries(inverted_index, queries,operations):
    results = []
    for query in queries:
        operations = query.split(', ')
        result = inverted_index[operations[0]]
        for i in range(1, len(operations), 2):
            operator = operations[i]
            operand = operations[i+1]
            if operator == 'AND':
                result = perform_AND(result, inverted_index[operand])
            elif operator == 'OR':
                result = perform_OR(result, inverted_index[operand])
            elif operator == 'AND NOT':
                result = perform_AND_NOT(result, inverted_index[operand])
            elif operator == 'OR NOT':
                result = perform_OR_NOT(result, inverted_index[operand],
set(inverted_index.keys()))
        results.append(result)
    return results

def preprocess_query(query):
    # Lowercase the text
    query = query.lower()

    # Tokenization
    query_tokens = nltk.word_tokenize(query)

    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    query_tokens = [token for token in query_tokens if token not in stop_words]

    # Remove punctuations
    query_tokens = [re.sub(r'[^\w\s]', '', token) for token in query_tokens]

    # Remove blank space tokens
    query_tokens = [token for token in query_tokens if token.strip()]
```

```
    return ' '.join(query_tokens)

# Input format
def input_format():
    N = int(input("Enter the number of queries: "))
    queries = []
    operations =[]
    for _ in range(N):
        query = input("Enter the query: ")
        operation = input("Enter the operations: ")
        operations.append(operation.split(', '))
        cleaned_query = preprocess_query(query)
        queries.append(cleaned_query)
    return N, queries, operations

# Output format
def output_format(N, queries, results):
    for i in range(N):
        print(f"Query {i+1}: {queries[i]}")
        print(f"Number of documents retrieved for query {i+1}: {len(results[i])}")
        print(f"Names of the documents retrieved for query {i+1}: {'
'.join(results[i])}\n")

# Main function
def main():
    dataset_directory = 'preprocessed_text_files'  # Specify the directory containing
preprocessed files
    inverted_index_file = 'inverted_index.pkl'  # File to save the inverted index

    # Create inverted index if it doesn't exist, otherwise load it
    if not os.path.exists(inverted_index_file):
        inverted_index = create_inverted_index(dataset_directory)
        save_inverted_index(inverted_index, inverted_index_file)
    else:
        inverted_index = load_inverted_index(inverted_index_file)

    # Input
    N, queries,operations = input_format()

    # Execute queries
    results = execute_queries(inverted_index, queries,operations)

    # Output
```

```
    output_format(N, queries, results)

if __name__ == "__main__":
    main()
```



Result

Reference

https://huggingface.co/learn/nlp-course/chapter6/7

https://williamscott701.medium.com/information-retrieval-unigram-postings-and-positional-postings-

a28b907c4e8
https://web.stanford.edu/class/cs276/handouts/lecture3-tolerant-retrieval-handout-1-per.pdf
https://www.geeksforgeeks.org/types-of-queries-in-ir-systems/

PART 3 (POSITIONAL INDEX AND PHRASE QUERY)

Introduction

This Python script is designed to create and query a positional index, a data structure used in information retrieval systems to find documents containing specific phrases or word sequences. The script performs text preprocessing, creates a positional index from a set of text files, and then allows the user to enter phrase queries to retrieve relevant documents.

Data Preprocessing

The `preprocess_text` function performs the following steps on the input text:

1. **Lowercasing:** Convert all text to lowercase for consistency in token matching.
2. **Tokenization:** Break down the text into individual tokens (words) using NLTK's `word_tokenize` function.
3. **Stopword Removal:** Remove common stopwords (e.g., 'and', 'the', 'is') that do not contribute much to the meaning of the text.
4. **Punctuation Removal:** Eliminate punctuation marks to focus on the essential content of the text.
5. **Blank Space Token Removal:** Remove any tokens consisting solely of whitespace characters.

Positional Index Creation

The `create_positional_index` function creates a positional index from a set of text files in a specified directory. The index is a dictionary where each key is a token, and the associated value is another dictionary that maps filenames to lists of positions where the token appears in that file.

Query Processing

The `execute_phrase_queries` function takes a positional index and a list of queries, processes them using the following steps:

1. Preprocess the query text.
2. For each query, initialize the result set with the documents containing the first term.
3. Intersect the result set with the documents containing the subsequent terms.
4. For each document in the final result set, check if the terms appear in the correct order and positions.
5. Return the list of document IDs where the phrase appears.

Input/Output Handling

The script provides an `input_format` function to prompt the user for the number of queries and the queries themselves. The `preprocess_query` function preprocesses the input query text in the same way as the text files.

The `output_format` function prints the results of the queries, including the number of documents retrieved and the names of the retrieved documents.

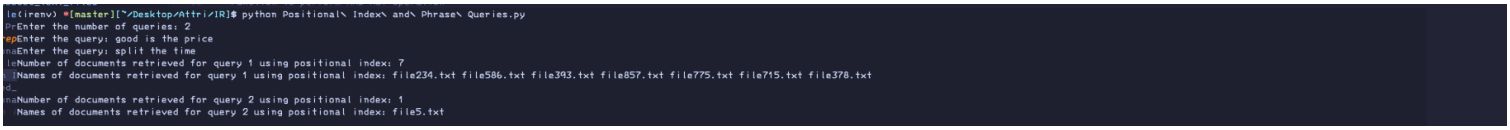
Main Function

The `main` function is the entry point of the script. It performs the following tasks:

1. Checks for an existing positional index file. If it exists, it loads the index. Otherwise, it creates a new index and saves it to a file.
2. Prompts the user for input queries.
3. Executes the phrase queries using the positional index.
4. Prints the results using the output format.

Results

The script provides a framework for creating and querying a positional index, with the ability to process phrase queries and retrieve relevant documents. It can be used as a starting point for more advanced information retrieval systems that require phrase-level matching.

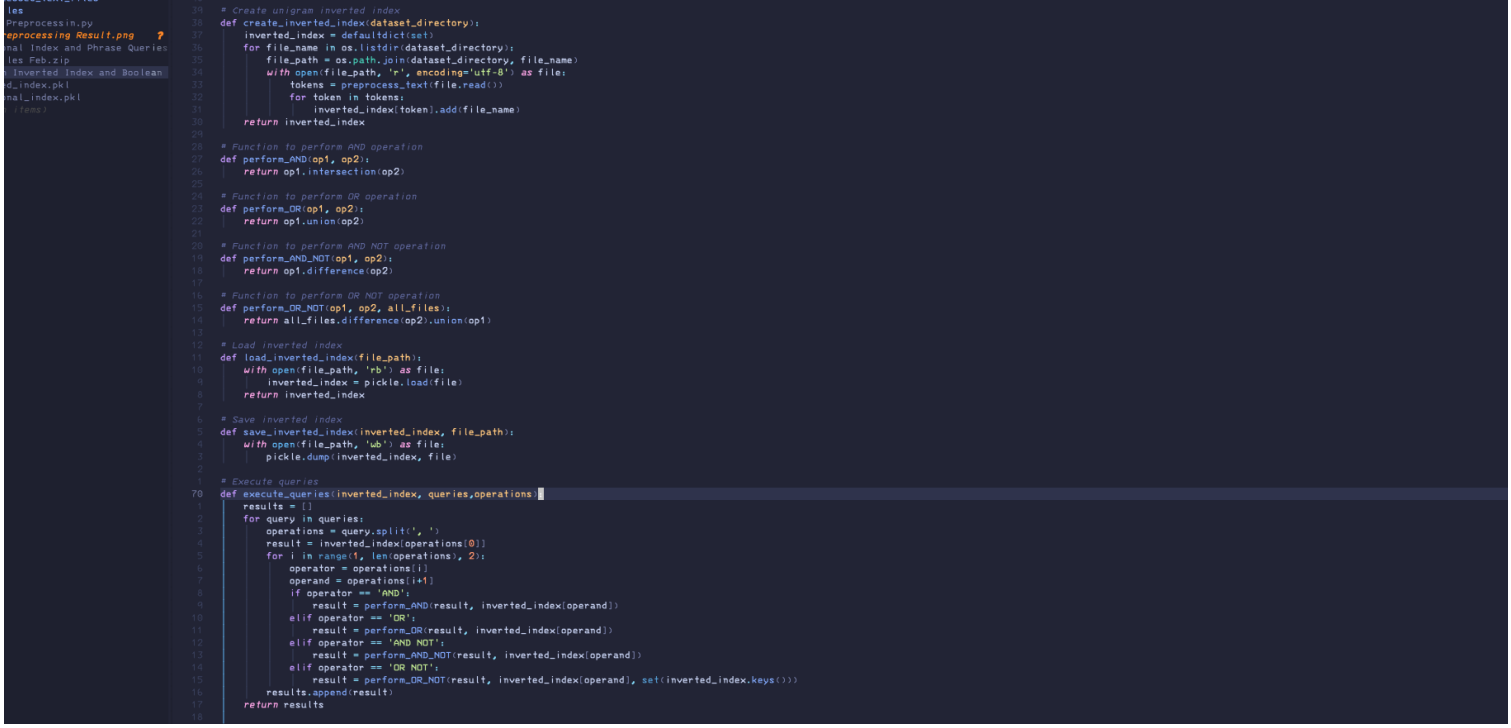



```
import os
import pickle
import re
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import string

# Preprocessing function
def preprocess_text(text):

    # Lowercasing
    text = text.lower()

    # Tokenization
    tokens = word_tokenize(text)

    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    tokens = [token for token in tokens if token not in stop_words]

    # Remove punctuations
```

```
    tokens = [token for token in tokens if token not in string.punctuation]

    # Remove blank space tokens
    tokens = [token for token in tokens if token.strip()]

    return tokens

# Create positional index
def create_positional_index(dataset_directory):
    positional_index = {}
    for file_name in os.listdir(dataset_directory):
        file_path = os.path.join(dataset_directory, file_name)
        with open(file_path, 'r', encoding='utf-8') as file:
            tokens = preprocess_text(file.read())
            for position, token in enumerate(tokens):
                if token not in positional_index:
                    positional_index[token] = {}
                if file_name not in positional_index[token]:
                    positional_index[token][file_name] = []
                positional_index[token][file_name].append(position)
    return positional_index

# Load positional index
def load_positional_index(file_path):
    with open(file_path, 'rb') as file:
        positional_index = pickle.load(file)
    return positional_index

# Save positional index
def save_positional_index(positional_index, file_path):
    with open(file_path, 'wb') as file:
        pickle.dump(positional_index, file)

# Execute phrase queries
def execute_phrase_queries(positional_index, queries):
    results = []
    for query in queries:
        query_terms = preprocess_text(query)
        query_result = set(positional_index[query_terms[0]].keys())
        for term in query_terms[1:]:
            if term in positional_index:
                query_result =
query_result.intersection(positional_index[term].keys())
            else:
```

```
                query_result = set()
                break
        if query_result:
            final_result = []
            for doc_id in query_result:
                positions = positional_index[query_terms[0]][doc_id]
                for pos in positions:
                    if all(pos + i + 1 in positional_index[term][doc_id] for i, term
in enumerate(query_terms[1:])):
                        final_result.append(doc_id)
                        break
            results.append(final_result)
        else:
            results.append([])
    return results

#Preprocess query
def preprocess_query(query):
    # Lowercase the text
    query = query.lower()

    # Tokenization
    query_tokens = nltk.word_tokenize(query)

    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    query_tokens = [token for token in query_tokens if token not in stop_words]

    # Remove punctuations
    query_tokens = [re.sub(r'[^\w\s]', '', token) for token in query_tokens]

    # Remove blank space tokens
    query_tokens = [token for token in query_tokens if token.strip()]

    return ' '.join(query_tokens)

# Input format
def input_format():
    N = int(input("Enter the number of queries: "))
    queries = []
    for _ in range(N):
        query = input("Enter the query: ")
        cleaned_query = preprocess_query(query)
        queries.append(cleaned_query)
```

```
    return N, queries

# Output format
def output_format(N, queries, results):
    for i in range(N):
        print(f"Number of documents retrieved for query {i+1} using positional index:
{len(results[i])}")
        print(f"Names of documents retrieved for query {i+1} using positional index:
{' '.join(results[i])}\n")

# Main function
def main():
    dataset_directory = 'preprocessed_text_files'  # Specify the directory containing
preprocessed files
    positional_index_file = 'positional_index.pkl'  # File to save the positional
index

    # Create positional index if it doesn't exist, otherwise load it
    if not os.path.exists(positional_index_file):
        positional_index = create_positional_index(dataset_directory)
        save_positional_index(positional_index, positional_index_file)
    else:
        positional_index = load_positional_index(positional_index_file)

    # Input
    N, queries = input_format()

    # Execute phrase queries
    results = execute_phrase_queries(positional_index, queries)

    # Output
    output_format(N, queries, results)

if __name__ == "__main__":
    main()
```



Reference

https://nlp.stanford.edu/IR-book/html/htmledition/positional-indexes-1.html
https://www.brainkart.com/article/Types-of-Queries-in-IR-Systems_11609/#:~:text=A%20phrase%20query%20consists%20of,searching%20that%20we%20mention%20below.
https://nlp.stanford.edu/IR-book/html/htmledition/phrase-queries-1.html