

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 stopwords 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.

```

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)

```

```

Pr
Original content of file: file686.txt
let'sve had this tuner for about 4 years and it's been great. It tunes fine. For a long time the tuner just sat in my room, but recently I started taking the tuner around to practice and transporting it in my bag. Unfortunately one of the 3 plastic pieces where the gooseneck attaches to the tuner broke off. It still works but the head sometimes falls off of the tuner. Now I'm looking for a clip on tuner that is a little more durable. If anyone has any suggestions, please let me know.
d.
hah I've added a photo that shows where one of the plastic pieces is missing (so you can see the joint)

Content of file after preprocessing: file686.txt
've tuner 4 years 's great tunes fine long time tuner sat room recently started taking tuner around practice transporting bag unfortunately one 3 plastic pieces gooseneck attaches tuner broke still works head sometimes falls tuner 'm looking clip tuner little durable anyone suggestions please let know 've added photo shows one plastic pieces missing see joint

Original content of file: file692.txt
For practical purposes, these patches are thin (which I like) and clear (if you get the clear ones). They are not hard "plastic" (yes, I made up a new word...remember you heard it here first) but have a snug, comfortable feel under the teeth. I use them on my OBrien crystal and Richard Hawkins Model G mouthpieces and they look great. Now, the best thing about them for me personally is the R....my name is Robert. Hah. Looks like I have custom mouthpiece patches. :-))

Content of file after preprocessing: file692.txt
practical purposes patches thin like clear get clear ones hard '' plastic '' yes made new word ... remember heard first snug comfortable feel teeth use obrien crystal richard hawkins model g mouthpieces look great best thing personally r .... name robert hah looks like custom mouthpiece patches

Original content of file: file662.txt
This black guard fit perfectly on my squier affinity telecaster that I was doing my rendition of macauber(Keith Richards telecaster). Very impressed with how well it was made. I just had to file out the pickup hole just a little bit for it to fit with ease. But all mounting screw holes lined up and I didn't have to do any fit work that you have to do most of the time, even on more expensive pick guards sometimes. I am very happy with how it looks and fits, I do really like the matte finish of it too.

Content of file after preprocessing: file662.txt
black guard fit perfectly squier affinity telecaster rendition macauber keith richards telecaster impressed well made file pickup hole little bit fit ease mounting screw holes lined n't fit work time even expensive pick guards sometimes happy I looks fits really like matte finish

Original content of file: file676.txt
I really really loved these till this happened at a gig last night ( see picture)
Don't let me discourage you. The 2.0 looks way smarter and better designed. I wish they would let me trade in my old ones.

Content of file after preprocessing: file676.txt
really really loved till happened gig last night see picture n't let discourage 2.0 looks way smarter better designed wish would let trade old ones

Original content of file: file731.txt
So far so good. Never heard of Lazzaro and who cares? The saxophone can perform. The brand doesn't matter. It is just a label. I'm on month 5 and it is holding up. Play it everyday. Upgraded the mouthpiece to a Clark Forbes Debut mouthpiece. Th e reads that the sax comes with isn't that great. Mine came with Lazzaro reads 2 1/2 strength and now I use D'Addario Select Jazz reads 3S and it does the trick. If you know how to take care of the saxophone it'll last. Well worth the investment. This is the perfect saxophone for those looking for a great instrument without paying hundreds to thousands of dollars.

Content of file after preprocessing: file731.txt
far good never heard lazzaro cares saxophone perform brand n't matter label 'm month 5 holding play everyday upgraded mouthpiece clark forbes debut mouthpiece reads sax comes n't great mine came lazzaro reads 2 1/2 strength use d'addario select jazz reads 3s trick know take care saxophone 'll last well worth investment perfect saxophone looking great instrument without paying hundreds thousands dollars

Original content of file: file725.txt
I must have have gotten lucky? All mine are gorgeous, and with a tueak or two, bridge tueak, neck tuea k, string height tueak, I am still in shock that a vintage , large headstock, 50's style Strat...the look, finish, and playability are rare in discounted axes! Hey, each and every piece screams FENDER. My studio wall is very, pinkies up! ...classy, so much, for so little!

Content of file after preprocessing: file725.txt
must gotten lucky mine gorgeous tueak two bridge tueak neck tuea k string height tueak still shock vintage large headstock 50 's style strat ... look finish playability rare discounted axes hey every piece screams fender studio wall pinkies ... classy much little

```

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()
```

```
Preprocess.py  
preprocessing Result.png ?  
mal_index and Phrase Queries  
let Feb.zip  
Inverted Index and Boolean  
d_index.pkl  
mal_index.pkl  
if __name__ == '__main__':  
    main()  
  
20 def create_inverted_index(dataset_directory):  
21     inverted_index = defaultdict(set)  
22     for file_name in os.listdir(dataset_directory):  
23         file_path = os.path.join(dataset_directory, file_name)  
24         with open(file_path, 'r', encoding='utf-8') as file:  
25             tokens = preprocess.text(file.read())  
26             for token in tokens:  
27                 inverted_index[token].add(file_name)  
28     return inverted_index  
  
29 # Function to perform AND operation  
20 def perform_AND(op1, op2):  
21     return op1.intersection(op2)  
22  
23 # Function to perform OR operation  
19 def perform_OR(op1, op2):  
17     return op1.union(op2)  
18  
24 # Function to perform AND NOT operation  
14 def perform_AND_NOT(op1, op2):  
13     return op1.difference(op2)  
15  
25 # Function to perform OR NOT operation  
18 def perform_OR_NOT(op1, op2, all_files):  
9     return all_files.difference(op2).union(op1)  
10  
26 # Load inverted index  
5 def load_inverted_index(file_path):  
4     with open(file_path, 'rb') as file:  
3         inverted_index = pickle.load(file)  
2         return inverted_index  
1  
65 # Save inverted index  
1     with open(file_path, 'wb') as file:  
2         pickle.dump(inverted_index, file)  
3  
4 # Execute queries  
5 def execute_queries(inverted_index, queries, operations):  
6     results = []  
7     for query in queries:  
8         operations = query.split(' ')  
9         result = inverted_index[operations[0]]  
10        for i in range(1, len(operations), 2):  
11            operator = operations[i]  
12            operand = operations[i+1]  
13            if operator == 'AND':  
14                result = perform_AND(result, inverted_index[operand])  
15            elif operator == 'OR':  
16                result = perform_OR(result, inverted_index[operand])  
17            elif operator == 'AND NOT':  
18                result = perform_AND_NOT(result, inverted_index[operand])  
19            elif operator == 'OR NOT':  
20                result = perform_OR_NOT(result, inverted_index[operand], set(inverted_index.keys()))  
21        results.append(result)  
22    return results  
23
```

Result

```
le[master][~/Desktop/Attri/IR]$ source ./env/bin/activate  
pr((rev)) *([master][~/Desktop/Attri/IR]$ python Unigram\ Inverted\ Index\ and\ Boolean\ Queries.py  
epEnter the number of queries: 2  
naEnter the query: good is skill  
isEnter the operations: AND  
isEnter the query: bought flight  
dEnter the operations: AND, OR NOT  
naQuery 1: good skill  
Number of documents retrieved for query 1: 0  
Names of the documents retrieved for query 1:  
  
Query 2: bought flight  
Number of documents retrieved for query 2: 0  
Names of the documents retrieved for query 2:  
  
([rev]) *([master][~/Desktop/Attri/IR]$ python Unigram\ Inverted\ Index\ and\ Boolean\ Queries.py  
Enter the number of queries: 1  
Enter the query: good is skill  
Enter the operations: AND  
Query 1: good skill  
Number of documents retrieved for query 1: 0  
Names of the documents retrieved for query 1:  
  
([rev]) *([master][~/Desktop/Attri/IR]$
```

Reference

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

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

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.

```
le@remv: *laster][~/Desktop/Attri/IR]$ python Positional\ Index\ and\ Phrase\ Queries.py
PrEnter the number of queries: 2
epEnter the query: good is the price
naEnter the query: split the time
lnNumber of documents retrieved for query 1 using positional index: 7
lnNames of documents retrieved for query 1 using positional index: file234.txt file586.txt file393.txt file857.txt file775.txt file715.txt file378.txt
d.
naNumber of documents retrieved for query 2 using positional index: 1
lnNames of documents retrieved for query 2 using positional index: file5.txt
```

```
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()

```

```

les
Preprocessing.py
Preprocessing Result.png
na1_index_and Phrase Queries
les Feb.zip
Inverted Index and Boolean
d_index.pkl
na1_index.pkl
Index

34 # Create unigram inverted index
35 def create_inverted_index(dataset_directory):
36     inverted_index = defaultdict(set)
37     for file_name in os.listdir(dataset_directory):
38         file_path = os.path.join(dataset_directory, file_name)
39         with open(file_path, 'r', encoding='utf-8') as file:
40             tokens = preprocess.textfile.read()
41             for token in tokens:
42                 inverted_index[token].add(file_name)
43     return inverted_index
44
45 # Function to perform AND operation
46 def perform_AND(op1, op2):
47     return op1.intersection(op2)
48
49 # Function to perform OR operation
50 def perform_OR(op1, op2):
51     return op1.union(op2)
52
53 # Function to perform AND NOT operation
54 def perform_AND_NOT(op1, op2):
55     return op1.difference(op2)
56
57 # Function to perform OR NOT operation
58 def perform_OR_NOT(op1, op2, all_files):
59     return all_files.difference(op2).union(op1)
60
61 # Load inverted index
62 def load_inverted_index(file_path):
63     with open(file_path, 'rb') as file:
64         inverted_index = pickle.load(file)
65     return inverted_index
66
67 # Save inverted index
68 def save_inverted_index(inverted_index, file_path):
69     with open(file_path, 'wb') as file:
70         pickle.dump(inverted_index, file)
71
72 # Execute queries
73 def execute_queries(inverted_index, queries, operations):
74     results = []
75     for query in queries:
76         operations = query.split(':', 1)
77         result = inverted_index[operations[0]]
78         for i in range(1, len(operations), 2):
79             operator = operations[i]
80             operand = operations[i+1]
81             if operator == 'AND':
82                 result = perform_AND(result, inverted_index[operand])
83             elif operator == 'OR':
84                 result = perform_OR(result, inverted_index[operand])
85             elif operator == 'AND NOT':
86                 result = perform_AND_NOT(result, inverted_index[operand])
87             elif operator == 'OR NOT':
88                 result = perform_OR_NOT(result, inverted_index[operand], set(inverted_index.keys()))
89             results.append(result)
90     return results

```

Reference

<https://nlp.stanford.edu/IR-book/html/htmledition/positional-indexes-1.html>

[https://www.brainkart.com/article/Types-of-Queries-in-IR-](https://www.brainkart.com/article/Types-of-Queries-in-IR-Systems_11609/#:~:text=A%20phrase%20query%20consists%20of,searching%20that%20we%20mention%20below.)

[Systems_11609/#:~:text=A%20phrase%20query%20consists%20of,searching%20that%20we%20mention%20below.](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>