CSE508 Information Retrieval

Winter 2024

Assignment-1

Report :

Vijay Krishna.P
2019183
ECE-2024.

Abstract :

The report details the development and implementation of a text analysis pipeline for a dataset consisting of various text files. This process involves three key stages: Data Preprocessing, Unigram Inverted Index and Boolean Queries, and Positional Index and Phrase Queries. Initially, text data undergoes preprocessing to normalize and refine the content, including tokenization, stopwords removal, punctuation elimination, and whitespace adjustment. Subsequently, a Unigram Inverted Index is constructed to facilitate efficient boolean query processing, supporting operations like AND, OR, and NOT. Finally, a Positional Index is created, enabling precise phrase query searches. This document outlines each step's methodology, implementation details, and sample query outputs to demonstrate the system's effectiveness in text retrieval and analysis.

1.Data Preprocessing .

Lowercase Conversion: All text was converted to lowercase to ensure uniformity and prevent case sensitivity issues during indexing.

- Tokenization: The text was broken down into individual words or tokens to facilitate further processing like stop word removal.
- Stop Word Removal: Common words that add little value to the text analysis were removed to focus on more meaningful words.
- Punctuation Removal: Punctuations were removed to clean the text for processing, leaving only alphanumeric characters and intra-word apostrophes.
- Blank Space Removal: Extra spaces were eliminated to ensure tokens are accurately represented.

  Examples are :

  ```
  Original content of file1.txt:
  ```
- `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.`
-
- `Preprocessed content of file1.txt:`
- `loving vintage springs vintage strat good tension great stability floating bridge want springs way go`
-
- `Original content of file10.txt:`
- `Awesome stand!`
-
- `Tip: The bottom part that supports the guitar had a weird angle when arrived, making the guitar slide back, becoming almost 100% on a vertical.`
- `To solve this, I assembled the product and the put a some pressure on the support frame, making it bend a little. Now my guitar sits perfectly. Check photos!`
-
- `Preprocessed content of file10.txt:`
- `awesome stand tip bottom part supports guitar weird angle arrived making guitar slide back becoming almost 100 vertical solve assembled product put pressure support frame making bend little guitar sits perfectly check photos`
-

2. Unigram Inverted Index and Boolean Queries

This component involved creating a unigram inverted index from the preprocessed data, which maps each unique term to the documents it appears in. Boolean query operations (AND, OR, AND NOT, OR NOT) were implemented, allowing for the combination of multiple search terms to refine query results. The use of Python's pickle module facilitated the serialization and deserialization of the inverted index, ensuring efficiency in data storage and retrieval.

Code Snippets:

```
import pickle
```

```


# Define the path to your consolidated preprocessed text file

output_file_path =
'/content/drive/MyDrive/IR_DataSet_Files/Preprocessing_data_textfiles/consolidated_preprocessed.txt'



# Initialize an empty dictionary for the inverted index

inverted_index = {}



# Open and read the preprocessed file

with open(output_file_path, 'r', encoding='utf-8') as file:

    # Assume each line corresponds to a document

    for doc_id, line in enumerate(file):

        # Tokenize the line into unigrams (since it's already preprocessed)

        unigrams = line.strip().split()

        # Update the inverted index

        for unigram in unigrams:

            if unigram in inverted_index:

                inverted_index[unigram].add(doc_id)

            else:

                inverted_index[unigram] = {doc_id}



#Serializing the Inverted Index with Pickle



# Define a path to save the serialized inverted index

index_file_path =
'/content/drive/MyDrive/IR_DataSet_Files/Preprocessing_data_textfiles/inverted_index.pkl'



# Serialize and save the inverted index using pickle

with open(index_file_path, 'wb') as index_file:

    pickle.dump(inverted_index, index_file)

```

```
print(f"Inverted index created and saved to {index_file_path}")
```

And next we checked the input , and output according to the given format by following the Process_Query functions using

```
def query_and(set1, set2):
    return set1 & set2


def query_or(set1, set2):
    return set1 | set2


def query_and_not(set1, set2):
    return set1 - set2



def query_or_not(set1, set2, universe_set):
    return set1 | (universe_set - set2)
```

Above are the Boolean Queries and process Query follows with i/o which we did in google colab.

```
def user_input_and_search(loaded_inverted_index, universe_set):
    num_queries = int(input("Enter number of queries: "))
    for i in range(num_queries):
        query_str = input(f"Enter query {i+1}: ")
        ops_str = input(f"Enter operations for query {i+1} (comma-separated): ")
        query_representation, num_docs, doc_names = process_query(query_str, ops_str,
loaded_inverted_index, universe_set)


        print(f"Query {i+1}: {query_representation}")
        print(f"Number of documents retrieved for query {i+1}: {num_docs}")
        if doc_names:
            print(f"Names of the documents retrieved for query {i+1}: {', '.join(doc_names)}")
        else:
            print("No documents found.")
        print()
```

```
Enter number of queries: 1

Enter query 1: Car bag in a canister

Enter operations for query 1 (comma-separated): 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: 125.txt, 14.txt, 154.txt, 172.txt, 231.txt, 254.txt,
266.txt, 28.txt, 304.txt, 319.txt, 38.txt, 402.txt, 423.txt, 430.txt, 474.txt, 513.txt, 562.txt, 62.txt,
673.txt, 69.txt, 691.txt, 74.txt, 759.txt, 772.txt, 822.txt, 885.txt, 896.txt, 899.txt, 907.txt, 93.txt,
934.txt
```

3.) Positional Index and Phrase Queries

The development of a positional index allowed for the support of phrase queries, enabling the identification of documents that contain exact sequences of words. This involved mapping each term not only to the documents in which it appears but also to the specific positions within those documents. This index was crucial for supporting complex search queries that go beyond single-term searches, providing users with the ability to perform precise information retrieval.

Same like we did in above , we are doing same here

Following are the code snippets and input and output .

```
source_dir = '/content/drive/MyDrive/IR_DataSet_Files/Preprocessed_Files'
```

```
positional_index = {}
```

```
for filename in os.listdir(source_dir):
    file_path = os.path.join(source_dir, filename)
    with open(file_path, 'r', encoding='utf-8') as file:
        content = file.read()
        words = content.split()
        for position, word in enumerate(words):
            if word not in positional_index:
                positional_index[word] = {}
```

```
            if filename not in positional_index[word]:
                positional_index[word][filename] = []
            positional_index[word][filename].append(position)


import string
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import pickle
import nltk

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


def preprocess_text(text):
    text = text.lower()
    tokens = word_tokenize(text)
    table = str.maketrans('', '', string.punctuation)
    stripped = [w.translate(table) for w in tokens]
    words = [word for word in stripped if word.isalpha()]
    stop_words = set(stopwords.words('english'))
    words = [w for w in words if not w in stop_words]
    return words


def find_documents_for_query(query, positional_index):
    query_terms = preprocess_text(query)
    documents = None
    for i, term in enumerate(query_terms):
        if term in positional_index:
            if documents is None:
                documents = {doc: positions for doc, positions in positional_index[term].items()}
            else:
                temp_documents = {}
                for doc, positions in documents.items():
```

```
                    if doc in positional_index[term]:
                        new_positions = [pos for pos in positional_index[term][doc] if pos-1 in
positions]
                        if new_positions:
                            temp_documents[doc] = new_positions
                documents = temp_documents
        else:
            return set()
    return set(documents.keys())


def user_input_and_search(loaded_positional_index):
    n = int(input("Enter number of queries: "))
    for i in range(n):
        query = input(f"Enter query {i+1}: ")
        matching_docs = find_documents_for_query(query, loaded_positional_index)
        print(f"Number of documents retrieved for query {i+1} using positional index:
{len(matching_docs)}")
        if matching_docs:
            print(f"Names of documents retrieved for query {i+1} using positional index: {',
'.join(sorted(matching_docs))}")
        else:
            print("No documents found.")



user_input_and_search(loaded_positional_index)
```

```
.Enter number of queries: 1

Enter query 1: product

Number of documents retrieved for query 1 using positional index: 106

Names of documents retrieved for query 1 using positional index: file10.txt, file105.txt, file115.txt,
file121.txt, file134.txt, file142.txt, file152.txt, file167.txt, file168.txt, file171.txt, file178.txt,
file180.txt, file186.txt, file19.txt, file196.txt, file208.txt, file217.txt, file225.txt, file228.txt,
file229.txt, file241.txt, file251.txt, file26.txt, file269.txt, file282.txt, file298.txt, file30.txt,
file326.txt, file337.txt, file35.txt, file358.txt, file363.txt, file372.txt, file374.txt, file375.txt,
file378.txt, file388.txt, file389.txt, file404.txt, file42.txt, file422.txt, file423.txt, file425.txt,
file435.txt, file437.txt, file443.txt, file447.txt, file46.txt, file460.txt, file477.txt, file487.txt,
file501.txt, file525.txt, file544.txt, file558.txt, file561.txt, file565.txt, file569.txt, file573.txt,
file576.txt, file586.txt, file592.txt, file616.txt, file622.txt, file634.txt, file64.txt, file647.txt,
```

file659.txt, file677.txt, file68.txt, file685.txt, file698.txt, file70.txt, file718.txt, file719.txt, file720.txt, file75.txt, file758.txt, file769.txt, file783.txt, file793.txt, file815.txt, file817.txt, file819.txt, file831.txt, file835.txt, file843.txt, file849.txt, file861.txt, file872.txt, file888.txt, file898.txt, file917.txt, file923.txt, file931.txt, file935.txt, file938.txt, file943.txt, file944.txt, file945.txt, file947.txt, file951.txt, file956.txt, file96.txt, file966.txt, file999.txt

Conclusions:

The implementation of Data Preprocessing, Unigram Inverted Index and Boolean Queries, and Positional Index and Phrase Queries demonstrates a robust approach to information retrieval within a corpus of text documents. By leveraging Python for data manipulation and indexing, the system offers an effective solution for searching and retrieving information based on various query types. The methodologies outlined in this report can be applied to a wide range of information retrieval applications, showcasing the versatility and efficiency of the developed system.

Google Colab Link : Assignment1 IR 2019183 VijayKrishna

GitHub Repo : Repo Link