IR REPORT-1

Name:- Mahisha Ramesh
MT23121

Introduction

In the era of big data, efficient management and retrieval of information are crucial. Natural Language Processing (NLP) techniques play a significant role in processing and understanding textual data. This report presents a detailed analysis and implementation of data preprocessing techniques and the construction of a unigram inverted index with support for boolean queries. The system aims to preprocess textual data effectively and provide efficient querying capabilities to retrieve relevant information from the dataset.

TASK-1

1) Data Preprocessing

Data preprocessing is a critical step in NLP tasks, as it helps clean and prepare the text data for further analysis. The following subsections detail the preprocessing steps applied to each text file in the dataset:

1.1 Lowercasing the Text:

- Lowercasing the text involves converting all uppercase characters to lowercase. This step ensures uniformity in text representation and prevents duplication of terms due to case differences.

1.2 Tokenization:

- Tokenization is the process of splitting text into individual words or tokens. We employed the NLTK library to tokenize the text data, breaking down sentences into tokens based on whitespace and punctuation.

1.3 Removing Stopwords:

- Stopwords are common words in a language that do not carry significant meaning, such as "the," "is," and "at." Removing stopwords helps reduce noise in the data and focuses on the essential content. We utilized NLTK's stopwords corpus to remove stopwords from the tokenized text.

1.4 Removing Punctuations:

- Punctuation marks such as periods, commas, and exclamation marks were removed from the text. This step simplifies the text and improves the accuracy of subsequent analysis.

1.5 Removing Blank Space Tokens:

- After removing punctuations, there might be instances where consecutive punctuations result in blank space tokens. These blank space tokens were removed to retain only meaningful tokens in the text data.

Code:-

```
def preprocess_text(text):
    # Lowercase the text
    text = text.lower()

    # Remove HTML tags using BeautifulSoup
    text = BeautifulSoup(text, "html.parser").get_text()

    # 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 and ellipsis
    tokens = [token for token in tokens if token not in string.punctuation and token.strip() != "...."]

    # Join tokens back into text
    processed_text = ' '.join(tokens)

    return processed_text
```

Results:-

1)File: file337.txt

Original Text:

This is a great option for creating a lively, party-like atmosphere. It lights up with many different colors and can be used for many different environments. Its very portable and I like that there is very little set-up involved.

"This product was received at a free or discounted price in exchange for my review. I am a technology journalist and pride myself on providing fair and honest reviews. Please feel free to ask any questions you might have."

Preprocessed Text:

great option creating lively party-like atmosphere lights many different colors used many different environments portable like little set-up involved product received free discounted price exchange review technology journalist pride providing fair honest reviews please feel free ask questions might

2) File: file26.txt

Original Text:

Perfect Fit!! Color match is very close. Great product!! Added pickguard to a Les Paul Studio Deluxe..

Preprocessed Text:

perfect fit color match close great product added pickguard les paul studio deluxe

3)File: file503.txt

Original Text:

I own 3 of these Minis. Quality on this one is the worst. Glue joints at the neck/body were messy. Had a couple of small dings (see photo of one under strings near bridge) but the important stuff is OK (neck is straight and the bridge is sound). Fret edges are very smooth. Sounds great but not as good as the Ovangkol.

Preprocessed Text:

3 minis quality one worst glue joints neck/body messy couple small dings see photo one strings near bridge important stuff ok neck straight bridge sound fret edges smooth sounds great good ovangkol

4)File: file977.txt

Original Text:

Great material and very easy to work with. I cut this on my band saw with no issue and the surface cleans up very nice after cutting.

Preprocessed Text:

great material easy work cut band saw issue surface cleans nice cutting

5) File: file635.txt

Original Text:

I love programming with this device and software. Being able to make lighting fixtures and controls that work is amazing.

I use a touch screen to make lighting scene changes quickly and easy without a mouse

Preprocessed Text:

love programming device software able make lighting fixtures controls work amazing use touch screen make lighting scene changes quickly easy without mouse

TASK-2

Unigram Inverted Index and Boolean Queries

The construction of a unigram inverted index facilitates efficient information retrieval by mapping terms to the documents in which they occur. Additionally, the system supports boolean operations for querying the dataset. Key components of this section include:

2.1 Creating Unigram Inverted Index:

- The unigram inverted index was constructed from scratch, where each unique term is mapped to the list of documents in which it appears, along with their frequencies. The index facilitates quick access to documents containing specific terms.

2.2 Saving and Loading Unigram Inverted Index:

- Python's pickle module was utilized to save the constructed unigram inverted index to disk. This allows for reusing the index without the need for recomputation, enhancing efficiency in subsequent querying tasks.

2.3 Supporting Boolean Operations:

- The system provides support for boolean operations such as AND, OR, and NOT, enabling users to construct complex queries. These operations allow users to specify logical conditions for document retrieval based on the presence or absence of terms.

2.4 Generalized Queries:

- Users can specify generalized queries involving multiple terms and boolean operators. This flexibility allows users to formulate diverse and complex queries tailored to their information needs.

2.5 Input and Output Format:

- Queries follow a specific input format, consisting of an input sequence and operations separated by commas. The system outputs relevant document retrieval results, including the query ID, the number of documents retrieved, and the names of the retrieved documents.

2.6 Preprocessing Input Sequences:

- Input sequences undergo preprocessing similar to the dataset to ensure consistency and accuracy in query processing. This preprocessing step aligns the input data with the preprocessed text data used to construct the inverted index.

Code:-

```
def tokenize(text):
    # Tokenization using a simple regular expression
    words = re.findall(r'\b\w+\b', text.lower())
    return words

def build_inverted_index(directory):
    inverted_index = defaultdict(list)
    file_count = 0

    for filename in tqdm(os.listdir(directory)):
        file_count += 1
        with open(os.path.join(directory, filename), 'r', encoding='utf-8') as file:
            content = file.read()
            tokens = tokenize(content)
            for token in tokens:
                inverted_index[token].append(filename)

    return inverted_index, file_count

def save_model(model, output_file):
    with open(output_file, 'wb') as file:
        pickle.dump(model, file)
```

Results

```
justice: ['preprocessed_file61.txt']
skeptical: ['preprocessed_file61.txt', 'preprocessed_file766.txt', 'preprocessed_file777.txt'
ordered: ['preprocessed_file61.txt', 'preprocessed_file489.txt', 'preprocessed_file143.txt', 'preprocessed_file3.txt', '
tremolo: ['preprocessed_file61.txt', 'preprocessed_file49.txt', 'preprocessed_file806.txt', 'preprocessed_file992.txt',
concerned: ['preprocessed_file61.txt', 'preprocessed_file152.txt', 'preprocessed_file48.txt', 'preprocessed_file328.txt'
planed: ['preprocessed_file61.txt']
broken: ['preprocessed_file61.txt', 'preprocessed_file796.txt', 'preprocessed_file45.txt', 'preprocessed_file386.txt', '
imported: ['preprocessed_file61.txt']
kramer: ['preprocessed_file61.txt', 'preprocessed_file380.txt', 'preprocessed_file380.txt', 'preprocessed_file380.txt']
st300: ['preprocessed_file61.txt', 'preprocessed_file380.txt', 'preprocessed_file380.txt']
refinished: ['preprocessed_file61.txt']
screws: ['preprocessed_file61.txt', 'preprocessed_file61.txt', 'preprocessed_file873.txt', 'preprocessed_file355.txt', '
strength: ['preprocessed_file61.txt', 'preprocessed_file849.txt', 'preprocessed_file434.txt', 'preprocessed_file831.txt'
seated: ['preprocessed_file61.txt', 'preprocessed_file465.txt', 'preprocessed_file196.txt']
careful: ['preprocessed_file61.txt', 'preprocessed_file141.txt', 'preprocessed_file853.txt', 'preprocessed_file801.txt',
philip: ['preprocessed_file61.txt']
driver: ['preprocessed_file61.txt', 'preprocessed_file61.txt', 'preprocessed_file145.txt', 'preprocessed_file915.txt', '
quit: ['preprocessed_file61.txt', 'preprocessed_file400.txt']
number: ['preprocessed_file61.txt', 'preprocessed_file89.txt', 'preprocessed_file8.txt', 'preprocessed_file590.txt', 'pr
shared: ['preprocessed_file61.txt']
shop: ['preprocessed_file61.txt', 'preprocessed_file221.txt', 'preprocessed_file514.txt', 'preprocessed_file907.txt', 'p
larger: ['preprocessed_file61.txt', 'preprocessed_file442.txt', 'preprocessed_file839.txt', 'preprocessed_file525.txt',
wound: ['preprocessed_file463.txt', 'preprocessed_file340.txt', 'preprocessed_file340.txt', 'preprocessed_file781.txt',
wrapped: ['preprocessed_file463.txt']
separate: ['preprocessed_file463.txt', 'preprocessed_file684.txt', 'preprocessed_file578.txt', 'preprocessed_file308.txt
admittedly: ['preprocessed_file463.txt', 'preprocessed_file879.txt']
tried: ['preprocessed_file463.txt', 'preprocessed_file961.txt', 'preprocessed_file79.txt', 'preprocessed_file584.txt', '
pedal: ['preprocessed_file684.txt', 'preprocessed_file684.txt', 'preprocessed_file610.txt', 'preprocessed_file79.txt', '
breaks: ['preprocessed_file684.txt', 'preprocessed_file684.txt', 'preprocessed_file906.txt', 'preprocessed_file591.txt',
break: ['preprocessed_file684.txt', 'preprocessed_file684.txt', 'preprocessed_file557.txt', 'preprocessed_file782.txt',
```

Process queries with operations

```
    return tokens

def AND(term1_docs, term2_docs):
    return list(set(term1_docs).intersection(set(term2_docs)))

def OR(term1_docs, term2_docs):
    return list(set(term1_docs).union(set(term2_docs)))

def AND_NOT(term1_docs, term2_docs):
    return list(set(term1_docs).difference(set(term2_docs)))

def OR_NOT(term1_docs, term2_docs, all_docs):
    return list(set(all_docs).difference(AND_NOT(all_docs, term1_docs)))

def process_query(query_tokens, operations, inverted_index):
    num_docs = None
    doc_names = None

    result = inverted_index.get(query_tokens[0], [])

    i = 1
    while i < len(query_tokens):
        operation = operations[i - 1]
```

Results:-

```
Enter the number of queries: 1
Enter query 1: Replaced the stock neck plate
Enter operation between 'replaced' and 'stock': AND
Enter operation between 'stock' and 'neck': AND
Enter operation between 'neck' and 'plate': AND
Number of documents for query 1: 1
Names of documents for query 1: ["'preprocessed_file548.txt'"]
```

TASK-3

Positional Inverted Index

Definition: A positional inverted index extends the functionality of a unigram inverted index by additionally storing the positions of each term within each document. This enables efficient retrieval of documents based on **phrase queries** and **proximity searches**, where the order or closeness of specific terms is important.

Key Components:

- **2.1 Creating Positional Inverted Index:**
    - Similar to the unigram index, but each term also maps to a list of documents and their corresponding positions.
    - For example, instead of just having [doc1, doc2], the entry for "quick" might be [(doc1, [2, 7]), (doc2, [3, 5]) indicating its positions in each document.
- **2.2 Saving and Loading Positional Inverted Index:**
    - Same principles as the unigram index, using libraries like Pickle for efficient storage and retrieval.
- **2.3 Supporting Phrase Queries:**
    - Enables searching for specific phrases like "the quick brown fox" by checking if terms appear consecutively in documents according to their stored positions.

- **2.4 Supporting Proximity Searches:**
    - Allows finding documents where terms are within a certain distance of each other (e.g., words within 5 positions).
- **2.5 Generalized Queries:**
    - Similar to the unigram index, users can specify queries involving multiple terms, combining phrase and proximity searches with boolean operators.
- **2.6 Input and Output Format:**
    - Query format depends on the specific implementation, but could involve specifying terms, operators, and desired proximity constraints.
    - Output includes relevant documents, their IDs, and potentially the identified term positions within them.
- **2.7 Preprocessing Input Sequences:**
    - Similar to the unigram index, preprocessing ensures consistency between user input and the format used in the index for accurate query processing.

Code:-

```
def build_positional_index(dataset_path):
    positional_index = {}

    # Iterate over each file in the dataset
    for filename in os.listdir(dataset_path):
        if filename.endswith('.txt'):
            file_path = os.path.join(dataset_path, filename)
            with open(file_path, 'r', encoding='utf-8') as file:
                text = file.read()
                tokens = tokenize(text)
                # Update positional index for each token
                for position, token in enumerate(tokens):
                    if token not in positional_index:
                        positional_index[token] = {}
                    if filename not in positional_index[token]:
                        positional_index[token][filename] = []
                    # Store the position of the token in the document
                    positional_index[token][filename].append(position)

    return positional_index

# Define the path to the dataset obtained from Q1
dataset_path = "/content/drive/MyDrive/preprocessed_files"

# Build the positional index
```

Results:-

```
Streaming output truncated to the last 5000 lines.
    Positions: [58]
Term: ew20ase
  Document: preprocessed_file156.txt
    Positions: [2]
Term: recessed
  Document: preprocessed_file156.txt
    Positions: [8, 74]
  Document: preprocessed_file538.txt
    Positions: [21]
Term: ratched
  Document: preprocessed_file156.txt
    Positions: [12]
Term: clasp
  Document: preprocessed_file156.txt
    Positions: [13]
Term: mad
  Document: preprocessed_file156.txt
    Positions: [14]
Term: sneeze
  Document: preprocessed_file156.txt
    Positions: [27]
Term: as93
  Document: preprocessed_file156.txt
    Positions: [34]
Term: seatbelt
```

✓ 10s completed at 9:12 PM

```
Enter the number of queries: 1
Enter phrase query: i assembled the product and the put a some pressure on the support frame
Number of documents retrieved for query i assembled the product and the put a some pressure on the support frame using pos
1
Names of documents retrieved for query i assembled the product and the put a some pressure on the support frame using posi
preprocessed_file10.txt
```