

Assignment 1

Hrishikesh Khambete(MT23038)

LIBRARIES USED

- **os** : os is used for importing dataset folders from directory.
- **NLTK** : The Natural Language Toolkit (nltk) is used for performing tasks such as: Tokenization, Part-of-speech tagging, Sentiment Analysis Stemming and Lemmatization etc.
- **BeautifulSoup** : It is used for web scraping and parsing HTML and XML documents, providing a convenient and flexible way to extract and manipulate data from web pages.
- **pickle** : The pickle module in Python is used for serializing and deserializing Python objects.
- **re** : Provides operations of detecting patterns with the help of regular expressions.
- **string** : For working with text data present in dataset files.

Q1. Data Preprocessing

PREPROCESSING

1. Read data using `os.listdir(file path)`.
2. A dictionary is created with docId name.
3. Total count of files in the folder=999.

PRE-PROCESSING STEPS ON THE DATASET:

1. Converted all the files text data to lowercase using the `.lower()` function.
2. Used BeautifulSoup library to extract data from HTML/XML text and remove tags.
3. Then performed tokenization using the `.word_tokenize()` method.
4. Removing stopwords from the tokenized text file using stopwords module.
5. Remove the punctuation marks from the text files.
6. Remove the blank spaces from all the files in the folder

File 1

loving vintage springs vintage strat good tension great stability floating bridge want springs way go

File 2

awesome stand tip bottom part supports guitar weird angle arrived making guitar slide back becoming almost 100 vertical solve a ssembled product put pressure support frame making bend little guitar sits perfectly check photos

File 3

amp real deal great crunch gain tones tweaking half bad clean ish tones ve played two 8 orange cabs get cool cute crazy money s ound pleasing revealing practice amp primarily play blackstar stack ve fitted celestion v30s wow is would never know thing tone monster even knobs s easy get lost hours playing thing favorite match chapman ml1 hotrod volume tone control evh fans get lot m ucking around many knobs many options either guitar amp tone go see micro dark came s probably next higher gain buffered effect s loop speaker emu headphone recording direct s thing

File 4

lot mixer great podcasting 4 outputs used monitor record cue audio mute 34 figure every channel fantastic three source switch h eadphonecontrol room must podcasting also aux return inputs used extra stereo inputs volumed aux return knobs thing nt like mix er xlr outputs back require adaptors use rca 14 plugs get adaptors

File 5

mic boss lot better mic ve seen used outofthebox voice sounds great even processing compression eq sounds fantastic rejects ton background noise sounds amazing runs hot ll want clean preamping get clean signal amazing mic price

Fig- printing random preprocessed files

QUESTION 2

FUNCTION DEFINITIONS FOR OR, AND, NOT, AND NOT and OR NOT

AND (operation == 'AND'):

- `set_result = set1.intersection(set2)`: Calculate the intersection of set1 and set2 and store the result in set_result.

AND NOT (operation == 'AND NOT'):

- `set_result = set1.difference(set2)`: Calculate the set difference between set1 and set2 (i.e., elements in set1 that are not in set2) and store the result in set_result.

OR (operation == 'OR'):

- `set_result = set1.union(set2)`: Calculate the union of set1 and set2 (i.e., all unique elements from both sets combined) and store the result in set_result.

OR NOT (operation == 'OR NOT'):

- `set_result = set1.union(set1, universal_set - set2)`: Calculate the union of set1 with the set difference between universal_set and set2 (i.e., elements in universal_set that are not in set2) and store the result in set_result.

Query Processing Function

1. User Input:

- The user provides the number of queries and enters each query's input and operation sequences.

2. Preprocessing:

- Each query undergoes preprocessing, converting to lowercase, tokenizing, removing stopwords, and punctuation.

3. Query Processing:

- The preprocessed query tokens and operators are passed to **queryProcess** for processing.

4. Query Execution:

- **queryProcess** executes the query based on operators, merging lists of document names accordingly.

5. Output Display:

- The number of retrieved documents and their names for each query are printed to the console.

Output:-

```

18 query = word_tokenize(query)
19 print(query)
20 print(operators)
21
22 print("Query ", k+1, ":", end=" ")
23
24 if(len(query) != (len(operators)+1)):
25     print("Invalid query")
26 else:
27     for i in range(len(query)-1):
28         print(query[i]+" "+operators[i]+" ", end="")
29     print(query[len(query)-1])
30     fileNames = queryProcess(query, operators, index)
31     print("Number of documents retrieved for query ", k+1, ":", len(fileNames))
32
33     print("Names of the documents retrieved for query ", k + 1, ":", fileNames)
34
Enter the number of queries: 1
Query 1 :
Enter input sequence: power supply
Enter operation sequence: AND
['power', 'supply']
['AND']
Query 1 : power AND supply
Number of documents retrieved for query 1 : 14
Names of the documents retrieved for query 1 : {'file367.txt', 'file434.txt', 'file512.txt', 'file222.txt', 'file521.txt', 'file597.txt', 'file687.txt', 'file760.txt', 'file988.txt', 'file19.txt', 'file305.txt', 'file828.txt', 'file55.txt', 'file223.txt'}

```

Fig:- Unigram Inverted Index and Boolean Queries

Q3. Positional Index and Phrase Queries

Creating positional indexing

1. Iterate through each text file in the specified directory.
2. Read the content of each file.
3. Tokenize the text into words using NLTK's **word_tokenize** function.
4. For each word token, check if it exists in the **positional_index** dictionary. If not, create a new entry.
5. Update the positional index by adding the document index and position of the word within the document.
6. Saving dumping to pickle file

Creating query processing function

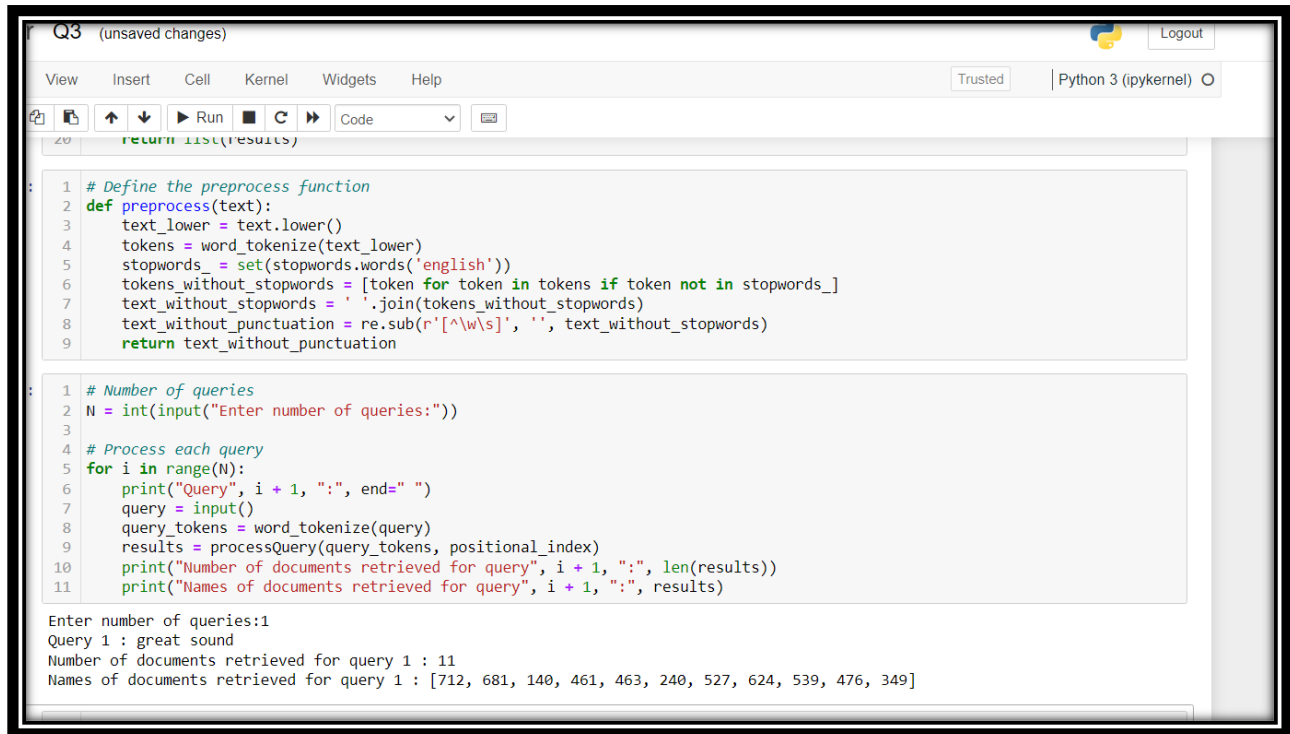
1. Initialize an empty set to store the results.
2. Iterate through the documents containing the first token of the query.
3. For each document, check the positions of the first token.
4. Within each position, iterate through the remaining tokens of the query.
5. Check if subsequent tokens exist in the document at the correct positions relative to the first token.
6. If all tokens are found in the correct order, add the document ID to the results set.
7. Return the list of document IDs containing the full phrase match.

User Query

1. Prompt the user to input the number of queries (N).
2. Read the query input from the user preprocess it and tokenize it using `word_tokenize()`.
3. Call the Queryprocess function with the tokenized query and the `positional_index` dictionary.

4. Retrieve the results returned by the Queryprocess function, which contains the document IDs matching the query.
5. Print the result

Output:-



```
Q3 (unsaved changes)
View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel) Logout

return list(results)

:
1 # Define the preprocess function
2 def preprocess(text):
3     text_lower = text.lower()
4     tokens = word_tokenize(text_lower)
5     stopwords = set(stopwords.words('english'))
6     tokens_without_stopwords = [token for token in tokens if token not in stopwords_]
7     text_without_stopwords = ' '.join(tokens_without_stopwords)
8     text_without_punctuation = re.sub(r'^\w\s', '', text_without_stopwords)
9     return text_without_punctuation

:
1 # Number of queries
2 N = int(input("Enter number of queries:"))
3
4 # Process each query
5 for i in range(N):
6     print("Query", i + 1, ":", end=" ")
7     query = input()
8     query_tokens = word_tokenize(query)
9     results = processQuery(query_tokens, positional_index)
10    print("Number of documents retrieved for query", i + 1, ":", len(results))
11    print("Names of documents retrieved for query", i + 1, ":", results)

Enter number of queries:1
Query 1 : great sound
Number of documents retrieved for query 1 : 11
Names of documents retrieved for query 1 : [712, 681, 140, 461, 463, 240, 527, 624, 539, 476, 349]
```

Fig:- positional index of phase query