ASSIGNMENT -1

UTKARSH PAL
2020144

Q-1) DATA-PREPROCESSING

pre_process_sentence(sentence): Tokenizes a sentence, converts it to lowercase, removes stopwords, and filters out non-alphanumeric tokens.

pre_process_file(read_file_path, write_file_path, mode): Preprocesses text files by removing stopwords, punctuation, and empty strings. It writes the processed content to new files.

Example Output of file264.txt

```
file264.txt
Original Content:
I just tested this fog fluid with a 1byone 400W fogger. Two 30 second bursts were sufficient to create enough fog layers for a moody atmosphere in a 2 car garage. This being a hot spac
Only downside is that the fog is not very dense. The difference between 2 bursts and 5 bursts was not too pronounced; it's a grey mist that does not become white or truly opaque, with
----------------------------------
Lowercase Content:
i just tested this fog fluid with a 1byone 400w fogger. two 30 second bursts were sufficient to create enough fog layers for a moody atmosphere in a 2 car garage. this being a hot spac
only downside is that the fog is not very dense. the difference between 2 bursts and 5 bursts was not too pronounced; it's a grey mist that does not become white or truly opaque, with
----------------------------------
Tokens:
['i', 'just', 'tested', 'this', 'fog', 'fluid', 'with', 'a', '1byone', '400w', 'fogger', '.', 'two', '30', 'second', 'bursts', 'were', 'sufficient', 'to', 'create', 'enough', 'fog', 'l
----------------------------------
Filtered Tokens without stop words:
['tested', 'fog', 'fluid', '1byone', '400w', 'fogger', '.', 'two', '30', 'second', 'bursts', 'sufficient', 'create', 'enough', 'fog', 'layers', 'moody', 'atmosphere', '2', 'car', 'gara
----------------------------------
Filtered Tokens without punctuations:
['tested', 'fog', 'fluid', '1byone', '400w', 'fogger', 'two', '30', 'second', 'bursts', 'sufficient', 'create', 'enough', 'fog', 'layers', 'moody', 'atmosphere', '2', 'car', 'garage',
----------------------------------
Filtered Tokens after removal of empty spaces:
['tested', 'fog', 'fluid', '1byone', '400w', 'fogger', 'two', '30', 'second', 'bursts', 'sufficient', 'create', 'enough', 'fog', 'layers', 'moody', 'atmosphere', '2', 'car', 'garage',
----------------------------------
```

Code Snippet

```python
def pre_process_sentence(sentence):
    sentence = sentence.lower()
    tokens = word_tokenize(sentence)
    stop_words = set(stopwords.words('english'))
    filtered_tokens = [word for word in tokens if word.lower() not in stop_words]
    filtered_tokens = [word for word in filtered_tokens if word.isalnum()]
    filtered_tokens = list(filter(lambda token: token.strip() != '', filtered_tokens))
    return filtered_tokens
```

Q-2.1)

create_ungigram_inverted_index(*read_directory_path*):

- The function takes a **directory path (read_directory_path)** as input, which is the directory containing the text files to be processed.
- It first calls a function **return_files_in_dir(read_directory_path)** to get a list of file names in the specified directory.
- It initializes an empty dictionary inverted_index to store the inverted index.
- It iterates through each file in the directory. If the file has a .txt extension, it proceeds to read the file.
- For each file, it reads the content and tokenizes it by splitting on whitespace. This creates a list of tokens from the file content.
- It iterates through each token in the list. For each token, it checks if it exists in the inverted index dictionary. If the token is not present, it adds the token as a key in the dictionary with the value being a list containing the numeric id of the file. If the token already exists in the inverted index, it checks if the file id is already present in the list associated with the token. If not, it appends the file identifier to the list.
- Once all files are processed and the inverted index is constructed, it dumps the inverted index dictionary using the pickle module and saves it.
- Finally, it returns the inverted index dictionary.

Example of few inverted index elements :-

```
kit [502, 885, 515, 918, 921, 840, 539, 249, 894, 991, 995, 996, 186, 543, 184, 494, 278, 848, 323, 283]
awesome [502, 846, 529, 105, 307, 851, 869, 672, 36, 302, 895, 314, 937, 824, 415, 562, 44, 239, 588, 765, 413, 83, 54, 610, 80, 808, 78
play [502, 927, 112, 106, 890, 891, 729, 850, 663, 307, 921, 100, 464, 316, 854, 707, 934, 711, 659, 467, 880, 843, 6, 575, 400, 762, 16
garage [502, 264, 571, 140]
personal [502, 69, 823, 821, 569, 989, 168]
enjoyment [502]
performances [502, 761]
anything [502, 270, 25, 31, 273, 659, 428, 563, 82, 361, 148, 781, 839, 805, 73, 966, 631, 182, 632, 801, 443, 324, 928, 679]
take [502, 2, 18, 30, 716, 459, 659, 21, 512, 213, 603, 945, 614, 359, 45, 991, 770, 599, 202, 959, 98, 543, 782, 74, 61, 592, 753, 196,
time [502, 927, 25, 265, 273, 844, 677, 307, 706, 666, 5, 711, 329, 467, 880, 658, 937, 248, 818, 830, 171, 47, 373, 992, 760, 44, 359,
break [502, 477, 846, 273, 51, 45, 759, 407, 823, 99, 233, 557, 782, 425, 684, 17, 491]
settings [502, 712, 920, 103, 467, 997, 344, 442, 130, 444]
able [502, 112, 460, 716, 470, 673, 659, 665, 35, 738, 512, 213, 978, 84, 992, 359, 629, 567, 995, 559, 178, 390, 186, 796, 812, 635, 98
dial [502, 850, 883, 507, 390, 542, 252]
pretty [502, 728, 112, 884, 338, 19, 107, 918, 474, 514, 129, 739, 710, 923, 777, 90, 85, 776, 576, 760, 50, 239, 588, 412, 189, 980, 21
much [502, 264, 270, 2, 927, 674, 18, 30, 462, 891, 926, 501, 33, 312, 845, 879, 514, 504, 869, 896, 302, 317, 23, 507, 936, 665, 301, 3
sound [502, 927, 660, 106, 890, 476, 310, 304, 339, 463, 885, 715, 503, 529, 703, 850, 677, 139, 461, 851, 925, 931, 514, 4, 712, 706, 8
expansion [502]
options [502, 921, 100, 373, 399, 396, 294, 652]
```

Q-2)

These functions perform Boolean operations on sets of documents retrieved from an inverted index.

T1_AND_T2(term1, term2, inverted_index): This function takes two terms (term1 and term2) and an inverted index (inverted_index) as input.
It retrieves the set of documents containing term1 and term2 from the inverted index and returns the **intersection** of these two sets.

T1_OR_T2(term1, term2, inverted_index): This function takes two terms (term1 and term2) and an inverted index (inverted_index) as input. It retrieves the set of documents containing term1 and term2 from the inverted index and returns the **union** of these two sets.

T1_AND_NOT_T2(term1, term2, inverted_index): This function takes two terms (term1 and term2) and an inverted index (inverted_index) as input. It retrieves the set of documents containing term1 and term2 from the inverted index and returns the documents containing term1 but not containing term2.
We do this by using the set **difference** operation.

T1_OR_NOT_T2(term1, term2, inverted_index): This function takes two terms (term1 and term2) and an inverted index (inverted_index) as input. It retrieves the set of documents containing term1 and term2 from the inverted index and returns the documents containing term1 or documents that don't contain term2. We do this by first making a universal set . **Then taking the difference between the universal set and second set and finally taking a union with the first set.**

Sample output

```
Query 1: car AND guitar
Number of documents retrieved for query 1: [166, 174, 542]
Names of documents retrieved for query 1: ['file166.txt', 'file174.txt', 'file542.txt']
```

Q-3)

create_unigram_positional_index(read_directory_path):

- The function takes a directory path (read_directory_path) as input, which is the directory containing the text files to be processed. It returns the list of file names in the specified directory.

- It initializes an empty dictionary positional_index to store the positional index.

- It iterates through each file in the directory. If the file has a .txt extension, it proceeds to read the file.

- For each file, it reads the content and tokenizes it by splitting on whitespace. This creates a list of tokens from the file content.

- It iterates through each token in the list. For each token, it checks if it exists in the positional index dictionary. If the token is not present, it adds the token as a key in the dictionary with the value being another dictionary. This inner dictionary has keys as the file id and values as lists containing the positions of the token within that file. If the token already exists in the positional index, it checks if the file id is already present in the inner dictionary. If not, it adds the file id and its corresponding positions to the inner dictionary.

- After constructing the positional index, it sorts the inner dictionaries within each token's entry based on the file identifiers.

- It saves the positional index dictionary using the pickle module and saves it to a file named positional_index.pickle.

- Finally, it returns the positional index dictionary.

Phrase Queries:-

positional_intersect(term1_list, term2_list, diff):

- This function computes the intersection of positional postings for two words inside a document using a provided positional difference (diff).
-
- It iterates through the two specified positional posting lists (term1_list and term2_list) at the same time, keeping track of two pointers.
-
- For each pair of postings with the same document ID, it compares the locations of the two words inside that document. If the difference between these places is equal to the stated difference, this occurrence is included in the result.
-
- The output is a dictionary in which the keys represent document IDs and the values indicate the points at which the phrases appear with the given difference.
- Finally, the dictionary is transformed into a list of tuples and returned.

positional_search(query_list, positional_index):

- This function performs a search operation on a provided list of query terms (query_list) against a positional index.

- It starts by determining if the query consists of only one phrase. If yes, it fetches and returns the positional posting list for that word from the positional index.

- For queries with more than one phrase, it loops through nearby pairs of terms in the query list.

- It uses the positional index to fetch the positional posting lists for both words in each pair.
- It then uses positional_intersect to discover documents where the phrases appear with the given positional difference, updating the result iteratively for each pair.

- The final result is a list of tuples representing document IDs and positions where all query terms occur with the specified positional differences.

Query processing and retrieval:

After collecting all inquiries, the function starts a new loop to process and get results for each query.
This loop gets queries from the input_query_list.
It preprocesses the question by tokenizing and lowercasing it with the pre_process_sentence() function, yielding query_list.
For each token in the query_list, it generates a list that includes the token and its query index.
It then invokes the positional_search() method with query_list_index and the loaded positional index (load_positional_index) to get suitable documents.
It outputs information about the obtained documents, such as the query number, the query itself, the number of documents retrieved, the document IDs, and the file names.

Output:
The function prints the results for each query, displaying the query number, the query itself, the number of documents retrieved, the document IDs, and the corresponding file names.

```
Query 1: highly recommended
Number of documents retrieved for query 1: 6
Number of documents retrieved for query 1: [(50, [34]), (81, [30]), (171, [10]), (257, [65]), (382, [64]), (626, [64])]
Names of documents retrieved for query 1: ['file50.txt', 'file81.txt', 'file171.txt', 'file257.txt', 'file382.txt', 'file626.txt']
```