CSE508 Information Retrieval

Winter 2023
Assignment-1
Group - 70

Ashish Kamathi - 2020364
Ashwin Sheoran - 2020288
Vedant Patil - 2020348

Github Repo link : https://github.com/Vedant0925/CSE508_Winter2023_A1_Group-72

Question 1: Data Preprocessing

(i) Relevant Text Extraction

1st Printed the initial 5 files before and preprocessing.
We will also unpack the compressed file to get the folder of our dataset to use it.
Then loaded all the files one by one using BeautifulSoup.Then we separated the title and the text part of the files and concatenated only the title and text part of the files. Then we again wrote this concatenated string back to the original files and then printed the first 5 files.

(ii) Preprocessing

We loaded the dataset obtained after Title and Text extraction, Using the nltk library, we 1st lowercase the text.
Then we did tokenization, in which we also considered blank spaces as individual tokens by replacing them with '@' and after tokenization replacing them back to blank space, so they don't get deleted during tokenization.
We also considered a-b-c as three different tokens 'a' , 'b' , 'c'.
Then we removed stop words from the tokens using nltk library.
Then we removed punctuations using isalpha() function
Then we removed blank spaces by removing ' ' from tokens.
We then saved our token in the respective original files as string of list of tokens.

Question 2:

a. Building inverted index-

os: used os.walk() to traverse through the folder and read file content for all files present in 'Work' folder. Each word of each doc is split and appended with the doc id it is present in.
Inverted index is being constructed in the build_inverted_index function where we take folder path as input and return a sorted inverted index.
save_inverted_index, load_inverted_index respectively use the python pickle module to save and load the inverted index after taking a file path as input(inverted_index.pickle here).

Building the index-Used defaultdict to handle missing keys. We traverse through each file and set the doc_id variable as the file name. Each file is split into words and doc_id is appended. The inverted index is then returned as a dict.

b. Query and operations-

Used nltk for the processing operations to be done on string provided by user(mainly removing stopwords from inputted query)

We then read through each file in 'Work' whose path has been provided and create an "index" dictionary. For each word in each file, the program checks if it is not a stopword or punctuation mark, and adds it to the index. If a word repeats, we add its file to the set of documents it repeats in. If the word is not yet in the index, a new key is created for the word, with the associated value being a set containing the file name.

Then take input from user and store the queries and operations to be performed for each query in a list. We iterate through each query in the list ignoring stopwords and punctuation(if any).

Question 3:

(i)

Firstly we are accessing the processed data set from the path and passing to create_index function. Using the OS library we are iterating through all the files and reading contect of each and breaking each file's content all possible bigram. We are maintaining a dictionary with all bigrams stored in it and if there is an existing bigram in it we just append the file name, if the bigram doesnt exist in dictionary we add the bigram as well as file to it. Then we store the returned index using pickle.

We go on to take user input in form of query which is then processed in pre_process function and all stopwords are removed using NLTK library. Further the processed query is sent to find_query function in which the query is converted to bigram and each bigram is then searched in the previously created bigram inverted index. If the bigram is present we append the corresponding files to the doc in a set and if the bigram doesnt exist we append an empty set. Finally after searching for all possible bigrams we intersect them all to find the files thats are common to all and return them back.

(ii) Positional Index

Using os.walk(), We will access the Processed Data and iterate through each document, to form a list of tokens from each documents. Then we created a Positional index using a dictionary, in which we stored the token, the total number of occurrences across all the documents and the index and document in which it was appearing. We would save the index in a file and then load it again using pickle.

Then we will take queries as input and preprocess the queries like we did for our given data. Then we took each word in query and compared it with its adjacent word if they are present in the same filein the same given order as the input phrase, so that we can add such file to our list. We then loop over inverted index and print each word and the docs it corresponds to.Then we give the list of required documents.

iii) Inverted bigram index returns false positive results, even if the words aren't present there is high possibility that the bigrams would be present hence leading to false results. Where as positional index is comparatively accurate because its stores tokenised words with their position in a sentence hence making it easier to search with accurate results.