Assignment -1

Q1. Unigram inverted index data structure

Assumptions:

File indexing starts from 0.

PreProcessing Steps:

Normalization : All the text is converted into lower case format.

Tokenization : The text is split into smaller units using nltk library.

StopWord Removal : All the stop words from the English language list from the nltk library are removed from the list.

Punctuation removal: All the special symbols are removed including the numbers 0-9.

Whitespace removal : All the white spaces are removed if any are present.

Lemmatization : It stems from the word but makes sure that the word does not lose its meaning.

Preprocessing returns a list of all the clean words.

Methodology:

First the file is read in lexicographical order using the *natsorted* function. Each file is preprocessed and the words are stored in a list for each file. Each word is then converted into a dictionary, which is the Unigram inverted index, with key as word and value as file index.

The UnigramIndex is saved and loaded using joblib.The query function is called to fetch the document list which contains the enquired word.

Saved Inverted Index : UnigramIndex.joblib

Functions Used:

preprocessing()- takes the text as input and returns the list of clean words.

AND_op() - accepts two lists and performs AND operation using merging techinque and returns the resultant list and the number of comparisons made while merging.

OR_op() - accepts two lists and performs OR operation using merging techinque and returns the resultant list and the number of comparisons made while merging.

NOT_op() - accepts a list and performs set(all files)- list and returns the resultant list.

AND_NOT() -accepts two lists and performs NOT operation on list2 using NOT_op() and then performs AND operation on list1 and new list2 using AND_op() and returns the resultant list and the number of comparisons made while merging

OR_NOT() -accepts two lists and performs NOT operation on list2 using NOT_op() and then performs OR operation on list1 and new list2 using OR_op() and returns the resultant list and the number of comparisons made while merging.

Query() - performs query on the Unigram inverted index data structure and returns the output
Input format: The first line contains the number of queries, N. The next 2N lines would represent the queries. Each query would consist of two lines:

(a) line 1: Input sentence

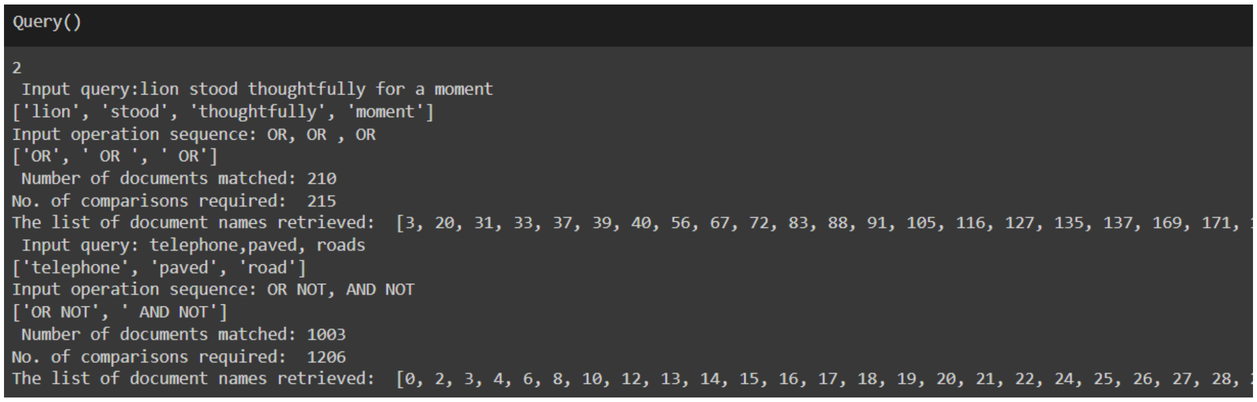
(b) line 2: Input operation sequence separated by “,”. Allowed operations [OR,AND,AND NOT, OR NOT]

Output- Number of documents matched:

No. of comparisons required:

The list of document numbers retrieved:

Output Screenshot:



```
Query()

2
 Input query:lion stood thoughtfully for a moment
['lion', 'stood', 'thoughtfully', 'moment']
Input operation sequence: OR, OR , OR
['OR', ' OR ', ' OR']
 Number of documents matched: 210
No. of comparisons required:  215
The list of document names retrieved:  [3, 20, 31, 33, 37, 39, 40, 56, 67, 72, 83, 88, 91, 105, 116, 127, 135, 137, 169, 171,
 Input query: telephone,paved, roads
['telephone', 'paved', 'road']
Input operation sequence: OR NOT, AND NOT
['OR NOT', ' AND NOT']
 Number of documents matched: 1003
No. of comparisons required:  1206
The list of document names retrieved:  [0, 2, 3, 4, 6, 8, 10, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 24, 25, 26, 27, 28,
```

Q2. Positional index data structure

Methodology:

For this particular question, we needed to make a positional index, combining the dataset given. For reading the dataset, we've used the *natsorted* function to read the files from the dataset folder in lexicographical order. After that while reading the files, I performed the preprocessing of the data. The first step was to preprocess the dataset given, for this, the steps were:

- Read lines from each file in the directory.
- Make every word in the lines to lowercase.
- Remove any numbers from the file
- Remove any special characters from the dataset.
- Convert the data to tokens and then remove stop words from the dataset.
- Stem the words, remove extra letters from similar words e.g (friends, friend are same).

Now for every word in the token list, make a dictionary that contains a list of dictionaries of document number : position in the document.

To make the dictionary, firstly, initialize the position for each word to 1 for a file.

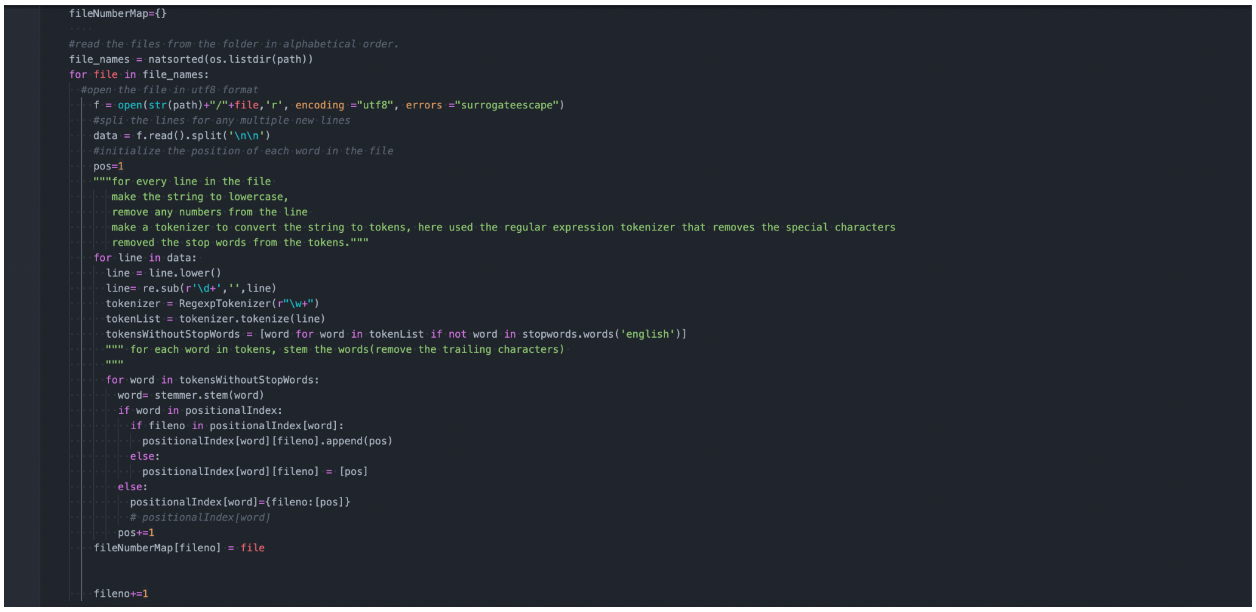
For every word in the preprocessed token list, check if the dictionary contains the word, if yes, then check whether the current file number is present. If yes, then add the position of the word to the already existing file number. If the dictionary does not contain the file number for the word, add a new list containing position against that word. If the dictionary does not contain the word, add a sub-dictionary {file number : [position]} against the word. Increment the position for the file.

For the names against every file number, create a dictionary that contains the file number and the file name.

To handle the phrase queries, we've created a function named phraseQuery() that takes a string/query. Preprocess the query in the same way as above. Convert the query to lowercase, remove numbers and special characters, stem the words and remove stop words.

Search the words in the positional index made. Get the file number and positions for the word in all files. Make a smaller dictionary that contains the file number and positions of only the words in the query for faster search. Now we do not need the files that contain positions less than the number of words in the query, so we make a list of file numbers that contain the number of positions smaller than words in the query and pop them from the dictionary. Now for the resultant dictionary, iterate over the file number and positions and sort the positions given. Make a list that checks for consecutive positions by subtracting the positions in the file. Now in the resultant array, find the maximum number of consecutive ones, and then add the filename and count if we get 1 equal to the number of tokens -1.

Preprocessing:

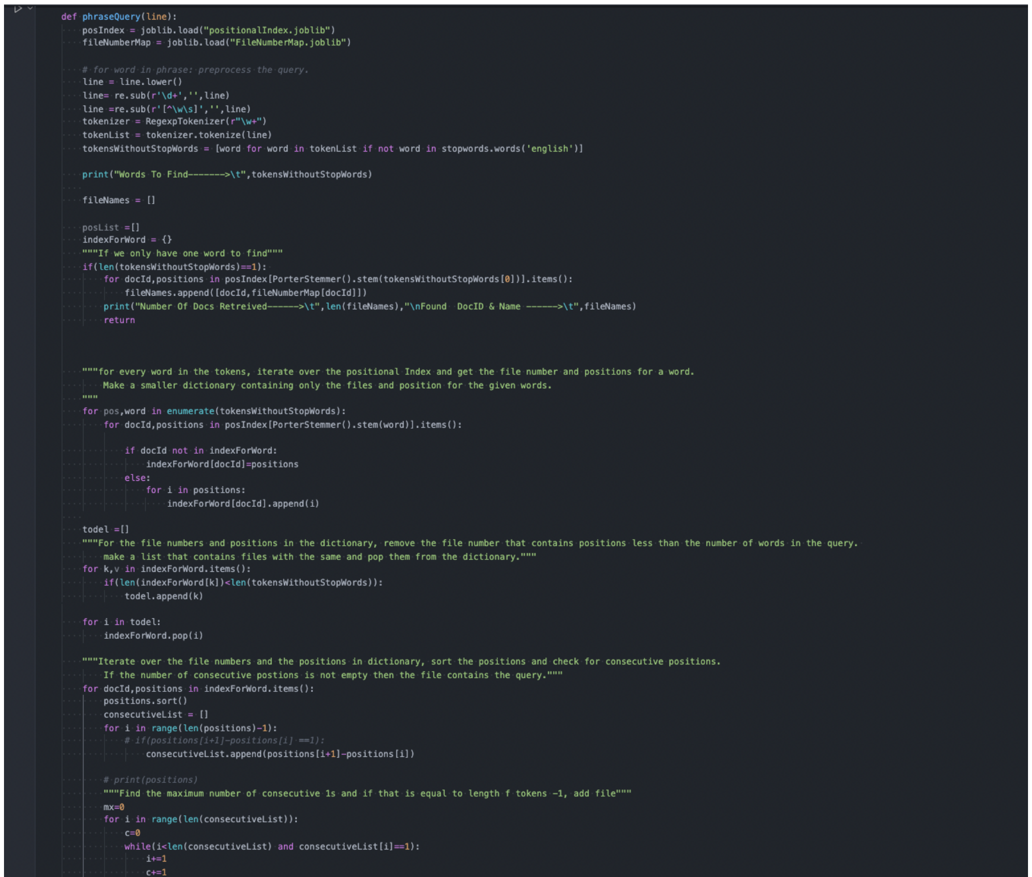


```python
fileNumberMap={}

#read the files from the folder in alphabetical order.
file_names = natsorted(os.listdir(path))
for file in file_names:
  #open the file in utf8 format
    f = open(str(path)+"/"+file,'r', encoding ="utf8", errors ="surrogateescape")
    #spli the lines for any multiple new lines
    data = f.read().split('\n\n')
    #initialize the position of each word in the file
    pos=1
    """for every line in the file
       make the string to lowercase,
       remove any numbers from the line
       make a tokenizer to convert the string to tokens, here used the regular expression tokenizer that removes the special characters
       removed the stop words from the tokens."""
    for line in data:
      line = line.lower()
      line= re.sub(r'\d+','',line)
      tokenizer = RegexpTokenizer(r"\w+")
      tokenList = tokenizer.tokenize(line)
      tokensWithoutStopWords = [word for word in tokenList if not word in stopwords.words('english')]
      """ for each word in tokens, stem the words(remove the trailing characters)
      """
      for word in tokensWithoutStopWords:
        word= stemmer.stem(word)
        if word in positionalIndex:
          if fileno in positionalIndex[word]:
            positionalIndex[word][fileno].append(pos)
          else:
            positionalIndex[word][fileno] = [pos]
        else:
          positionalIndex[word]={fileno:[pos]}
          # positionalIndex[word]
        pos+=1
    fileNumberMap[fileno] = file


    fileno+=1
```

Phrase query:



```python
def phraseQuery(line):
    posIndex = joblib.load("positionalIndex.joblib")
    fileNumberMap = joblib.load("FileNumberMap.joblib")

    # for word in phrase: preprocess the query.
    line = line.lower()
    line= re.sub(r'\d+','',line)
    line =re.sub(r'[^\w\s]','',line)
    tokenizer = RegexpTokenizer(r"\w+")
    tokenList = tokenizer.tokenize(line)
    tokensWithoutStopWords = [word for word in tokenList if not word in stopwords.words('english')]

    print("Words To Find------->\t",tokensWithoutStopWords)

    fileNames = []

    posList =[]
    indexForWord = {}
    """If we only have one word to find"""
    if(len(tokensWithoutStopWords)==1):
        for docId,positions in posIndex[PorterStemmer().stem(tokensWithoutStopWords[0])].items():
            fileNames.append([docId,fileNumberMap[docId]])
        print("Number Of Docs Retreived------>\t",len(fileNames),"\nFound  DocID & Name ------>\t",fileNames)
        return



    """for every word in the tokens, iterate over the positional Index and get the file number and positions for a word.
        Make a smaller dictionary containing only the files and position for the given words.
    """
    for pos,word in enumerate(tokensWithoutStopWords):
        for docId,positions in posIndex[PorterStemmer().stem(word)].items():

            if docId not in indexForWord:
                indexForWord[docId]=positions
            else:
                for i in positions:
                    indexForWord[docId].append(i)

    todel =[]
    """For the file numbers and positions in the dictionary, remove the file number that contains positions less than the number of words in the query.
        make a list that contains files with the same and pop them from the dictionary."""
    for k,v in indexForWord.items():
        if(len(indexForWord[k])<len(tokensWithoutStopWords)):
            todel.append(k)

    for i in todel:
        indexForWord.pop(i)

    """Iterate over the file numbers and the positions in dictionary, sort the positions and check for consecutive positions.
        If the number of consecutive postions is not empty then the file contains the query."""
    for docId,positions in indexForWord.items():
        positions.sort()
        consecutiveList = []
        for i in range(len(positions)-1):
            # if(positions[i+1]-positions[i] ==1):
                consecutiveList.append(positions[i+1]-positions[i])

        # print(positions)
        """Find the maximum number of consecutive 1s and if that is equal to length f tokens -1, add file"""
        mx=0
        for i in range(len(consecutiveList)):
            c=0
            while(i<len(consecutiveList) and consecutiveList[i]==1):
                i+=1
                c+=1
```