IR Assignment 1

Aditya Dixit(MT21004) Shivang Kanwar(MT21084)

Libraries Used: nltk, pandas, numpy, os, pickle, re

Question 1

(a) Carry out the suitable preprocessing steps on the given dataset.

We have done our assignment on google colab. So after downloading the dataset we have extracted it from zip file and uploaded it on google drive. After that, we have mounted our drive on colab.

Steps in Preprocessing are:-

- Converting to lowercase
- Removing underscore
- Converting number to text
- Removing punctuation
- Removing stopword
- Removing blank space
- Removing single character
- Removing non-ASCII words
- Doing Lemmatization

```
############################################## PART A ##############################################
#convert text to lowercase
def convert_lower_case(text):
  return np.char.lower(text)

#remove underscore
def rem_underscore(text):
  t=str(text)
  return t.replace("_","")

#Remove stopwords from text
def remove_stop_words(text):
    stop_words = stopwords.words('english')
    token = word_tokenize(str(text))
    text2 = ""
    for t in token:
      if t not in stop_words:
        text2 = text2+" "+t
    return np.char.strip(text2)

#Remove punctuation marks from text
def remove_punctuation(text):
    symbols = "/:;!\"#+-.&()*^_`<=>?@[\
    for i in range(len(symbols)):
        text = np.char.replace(text, symbols[i], ' ')
        text = np.char.replace(text, "  ", " ")
    text = np.char.replace(text, ',', ' ')
    text = np.char.replace(text, "'", "")
    return text

```

```
#Remove blank space
def remove_blank_space(text):
  t=str(text)
  return " ".join(t.split())

#Remove single characters
def remove_single_char(text):
    words = word_tokenize(str(text))
    text2 = ""
    for w in words:
        if len(w) > 1:
            text2 = text2 + " " + w
    return np.char.strip(text2)

#convert numbers to text
def convert_number_to_text(text):
    text = np.char.replace(text, "0", " zero ")     #replace 0 by zero in text.
    text = np.char.replace(text, "1", " one ")      #replace 1 by one in text.
    text = np.char.replace(text, "2", " two ")      #replace 2 by two in textd.
    text = np.char.replace(text, "3", " three ")    #replace 3 by three in text.
    text = np.char.replace(text, "4", " four ")     #replace 4 by four in text.
    text = np.char.replace(text, "5", " five ")     #replace 5 by five in text.
    text = np.char.replace(text, "6", " six ")      #replace 6 by six in text.
    text = np.char.replace(text, "7", " seven ")    #replace 7 by seven in text.
    text = np.char.replace(text, "8", " eight ")    #replace 8 by eight in text.
    text = np.char.replace(text, "9", " nine ")     #replace 9 by nine in text.
    return text

#performing lemmatization on data
def lem(text):
  lemmatizer = WordNetLemmatizer()
  tokens = word_tokenize(str(text))
  text2 = ""
  for w in tokens:
      text2 = text2 + " " + lemmatizer.lemmatize(w)
  return np.char.strip(text2)

#removing non-ascii characters
def remove_non_ascii(text):
  t=str(text)
  return t.encode("ascii","ignore").decode("utf-8","ignore")

#perform preprocessing
def preprocess_data(text):
  text = convert_lower_case(text)
  text = rem_underscore(text)
  text = convert_number_to_text(text)
  text = remove_punctuation(text)
  text = remove_stop_words(text)
  text = remove_blank_space(text)
  text = remove_single_char(text)
  text = remove_non_ascii(text)
  text = lem(text)
  return text
```

(b) Implement the unigram inverted index data structure.

Steps in creating inverted index are:-

- Taking the data from the file
- Cleaning data with preprocessing
- Tokenization of the data
- Creating index with the dictionary. Words as key and document number as values
- Storing the index in a pickle file

For eg:- Data:- “lion is happy”

Preprocessed:- “lion happy”

Tokens:-[“lion”,”happy”]

Index:- {“lion”:[2,3,45], “happy”:[3,5,78]}

```
################################################ PART B ##########################################
def create_inverted_index(paths):     #function create inverted index.
  store_filename = []                 #list for storing all file names
  docID = 0                           #counter for documents
  tii2=dict()                         # dictionary for storing index with word as key and position as values

  #loop for calculating index for each file
  for path in paths:
    file = open(path, 'r', encoding= 'ISO-8859-1')
    text = " ".join(file.read().split())    # for removing extra spaces
    file.close()
    print(docID)
    preprocessed_data = preprocess_data(text)
    tokens = word_tokenize(str(preprocessed_data))      #tokenize the preprocessed text and convert into tokens

    #loop for storing data in dictionary
    for t in tokens:
      if t not in tii2.keys():
        tii2[t] = list()
        tii2[t].append(docID)
      else:
        if docID not in tii2[t]:
          tii2[t].append(docID)

    filename = os.path.basename(path)        #extract last filename from  the path
    store_filename.append([filename])        #append into list
    docID += 1

  store_filename = pd.DataFrame(store_filename)     #convert filenames list of list into dataframe

  # save the inverted index and filenames in pickle file so that can we build futher
  with open('fn.pickle', 'wb') as handle:
    pickle.dump(tii2, handle, protocol=pickle.HIGHEST_PROTOCOL)

  store_filename.to_pickle("sf")
```

(c) Provide support for the following queries-
(i) x OR y
(ii) x AND y
(iii) x AND NOT y
(iv) x OR NOT y

We have implemented 5 functions AND, OR, NOT, AND NOT, OR NOT to perform operations on queries.

NOT, OR OPERATION

```
################################################## PART C ##############################################################
def not_operation(var):                         #function to perform not operation
  v = set(range(len(paths)))
  return list(v.difference(var))

def or_operation(var1,var2):                    #function to perform or operation
  n = len(var1)                                 #store length of var1 and var2 into n and m.
  m = len(var2)
  Result = []
  comparisons = 0
  i = 0
  j = 0

  while i<n and j<m:            #perform union in two set X and Y and count number of comparision.
    if var1[i]==var2[j]:
      Result.append(var1[i])
      i=i+1
      j=j+1
      comparisons=comparisons+1
    elif var1[i]<var2[j]:
      Result.append(var1[i])
      i=i+1
      comparisons=comparisons+1
    else:
      Result.append(var2[j])
      j=j+1
      comparisons=comparisons+1

  while i<n:                      #when only var1 set left then append into result without any comparision.
    Result.append(var1[i])
    i=i+1

  while j<m:                      #when only var2 set left then append into result without any comparision.
    Result.append(var2[j])
    j=j+1

  return Result, comparisons           #return union of var1 and var2 and number of comparision.
```

AND, AND NOT, OR NOT OPERATION

```
def and_operation(var1,var2):        #function to perform and operation
  n = len(var1)                      #store length of var1 and var2 into n and m.
  m = len(var2)
  Result = []
  comparisons =0
  i = 0
  j = 0

  while i<n and j<m:           #perform union in two set var1 and var2 and count number of comparision.
    if var1[i]==var2[j]:
      Result.append(var1[i])
      i=i+1
      j=j+1
      comparisons=comparisons+1
    elif var1[i]<var2[j]:
      i=i+1
      comparisons=comparisons+1
    else:
      j=j+1
      comparisons=comparisons+1
  return Result, comparisons     #return intersection of var1 and var2 and number of comparision.

def or_not_operation(var1, var2):                   #function to perform or Not operation
    var2 = not_operation(var2)                      #find not of var2.
    Result, comparisons = or_operation(var1, var2)  #after finding not of var2 then perform OR of var1 and var2 and find no of comparision.
    return Result, comparisons                      #return the result after var1 or Not var2 and number of comparision.

def and_not_operation(var1, var2):                  #function to perform and not operation
    var2 = not_operation(var2)                      #find not of var2.
    Result, comparisons = and_operation(var1, var2) #after finding not of Y then perform and of var1 and var2 and find number of comparision.
    return Result, comparisons                      #return the result after var1 or Not var2 and number of comparision.
```

Next we have implemented a function called cal_result() which drives the main query processing. It takes document list for each word and perform operation on them. It uses the and, or, not ,and not, or not function by calling them and them storing the result.

```
def cal_result(Query_docIds):
  op_pos = 0                        #iterator for positions
  Total_comp = 0                    #to store total comaprisons
  X = Query_docIds[0]
  for operand in Query_docIds[1:]:
    Y = operand

    operator = operations[op_pos]
    op_pos = op_pos+1

    if(operator == 'OR'):      #if operator is OR then call Function_or and find resultant list and number of comparision.
      X, No_of_compare = or_operation(X, Y)
      Total_comp = Total_comp + No_of_compare

    elif(operator == 'AND'):       #if operator is AND then call Function_or and find resultant list and number of comparision.
      X, No_of_compare = and_operation(X, Y)
      Total_comp = Total_comp + No_of_compare

    elif(operator == 'OR NOT'):       #if operator is OR NOT then call Function_or and find resultant list and number of comparision.
      X, No_of_compare = or_not_operation(X, Y)
      Total_comp = Total_comp + No_of_compare

    elif(operator == 'AND NOT'):      #if operator is AND NOT then call Function_or and find resultant list and number of comparision.
      X, No_of_compare = and_not_operation(X, Y)
      Total_comp = Total_comp + No_of_compare

    else:
      print("You Entered Wrong Operations.....!!!")

```

Main function is the starting function of the program. This function is executed first. Firstly it stores all paths in a path list , which is then passed to the create_inverted_index() function to create index. After this input to the program from user is taken. Query and operations to be performed are taken and after doing operations result is shown to the user.

CREATING INDEX

```
if __name__ == "__main__":

  files_path = "/content/drive/MyDrive/Humor,Hist,Media,Food"

  # find all files those are stored into files_path and store into path.
  paths = []
  for (dirpath, dirnames, filenames) in os.walk(str(files_path)):
    for i in filenames:
       paths.append(str(dirpath)+str("/")+i)

  create_inverted_index(paths)    #for creating index
  store_filename = []             #for storing filenames

  # read the index and filename
  with open('fn.pickle', 'rb') as handle:
    inverted_index = pickle.load(handle)
  store_filename=pd.read_pickle("sf")
```

TAKING INPUT AND PERFORMING OPERATIONS AND PRINTING RESULT

```
18
19  # Our input query
20  N = int(input("Enter Number of Queries you want to test:-"))
21  while N>0:
22    Query = input("Enter the query:-")
23    operations  = input("Enter operations with comma between them:-")
24    operations=operations.upper()
25    operations=operations.split(",")
26    operations = [i.strip() for i in operations]
27
28  # Query Pre-processing
29    pre_processed_query = preprocess_data(Query)
30    pre_processed_query = pre_processed_query.flatten()
31    pre_processed_query = np.char.split(pre_processed_query[0])
32    pre_processed_query = pre_processed_query.tolist()
33
34  #add word documents id to query document ids in sorted order
35    Query_docIds = []
36
37    for r in pre_processed_query:
38      if r not in inverted_index.keys():
39        print(r+" is not in any document. Please enter valid query")
40        break
41      word_docIds = inverted_index[r]
42      Query_docIds.append(sorted(word_docIds))
43
44    #check if processed query documents are 1 less than number of operations
45    if(len(Query_docIds)-1 == len(operations)):
46      cal_result(Query_docIds)
47    else:
48      print("Wrong Input")
49    N-=1
```

OUTPUT:-

```
Enter Number of Queries you want to test:-2
Enter the query:-lion stood thoughtfully for a moment
Enter operations with comma between them:-or,or,or


Number of documents matched:  211
No. of comparisons required:  336
Documents retreived are:-
['conan.txt', 'coyote.txt', 'incarhel.hum', 'cartoon_.txt', 'ivan.hum', 'lbinter.hum', 'murphys.txt', 'li
Enter the query:-telephone,paved, roads
Enter operations with comma between them:-or not,and not


Number of documents matched:  996
No. of comparisons required:  2244
Documents retreived are:-
['bbq.txt', 'gohome.hum', 'mothers.txt', 'harmful.hum', 'hate.hum', 'herb!.hum', 'mowers.txt', 'hell.jok'
```

Question 2

(a) Carry out the following preprocessing steps on the given dataset

Before preprocessing, we manually extracted the folder and then retrieved all files and stored their paths in the list.

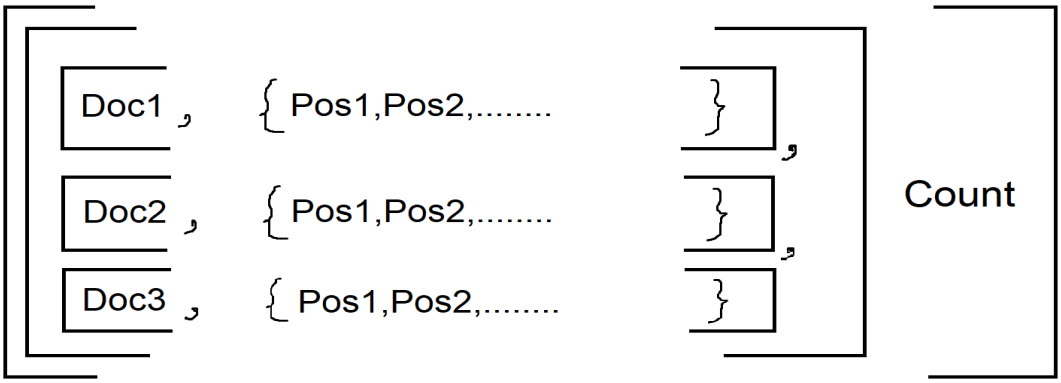
```
[9]  files_path = "/content/drive/MyDrive/Humor,Hist,Media,Food"

     #Store path of all files in paths
     paths = []
     for (dirpath, dirnames, filenames) in os.walk(str(files_path)):
       for i in filenames:
          paths.append(str(dirpath)+str("/")+i)

     print(len(paths))

     1133
```

(i). In preprocessing, first converted the text of each file into lower case.
(ii). Then converted this text into tokens using word_tokenize from nltk library.
(iii). Removed all stopwords from these tokens using stopwords of nltk library.
(iv). Removed punctuations from above filtered tokens using re.sub() which replaced all the char except a-z A-Z 0-9 _
(v). Finally removed all the blank spaces from these tokens.

```
###################################   part a   ##########################################

#Convert the text to lower case
def convert_lower_case(text):
    return np.char.lower(text)

#Perform word tokenization
def word_tokenization(text):
  tokens = word_tokenize(str(text))
  return tokens

#Remove stopwords from tokens
def remove_stop_words(tokens):
    stop_words = stopwords.words('english')
    filtered_tokens = []
    for word in tokens:
        if word not in stop_words:
            filtered_tokens.append(word)
    return filtered_tokens

#Remove punctuation marks from tokens
def remove_punctuation(words):
    filtered_words = []
    for word in words:
        filtered_word = re.sub(r'[^\w\s]', ' ', word)    #\w for string containing char a-z,A-Z,0-9,_, \s for whitespaces, ^for except these
        if filtered_word != '':
            filtered_words.append(filtered_word)
    return filtered_words

#Remove blank space tokens
def remove_blank_space_tokens(words):
  words=' '.join(words).split()
  return words

#Perform preprocessing
def preprocess_data(text):
    text = convert_lower_case(text)
    tokens = word_tokenization(text)
    filtered_tokens = remove_stop_words(tokens)
    filtered_tokens = remove_punctuation(filtered_tokens)
    filtered_tokens = remove_blank_space_tokens(filtered_tokens)
    return filtered_tokens
```

(b) Implement the positional index data structure

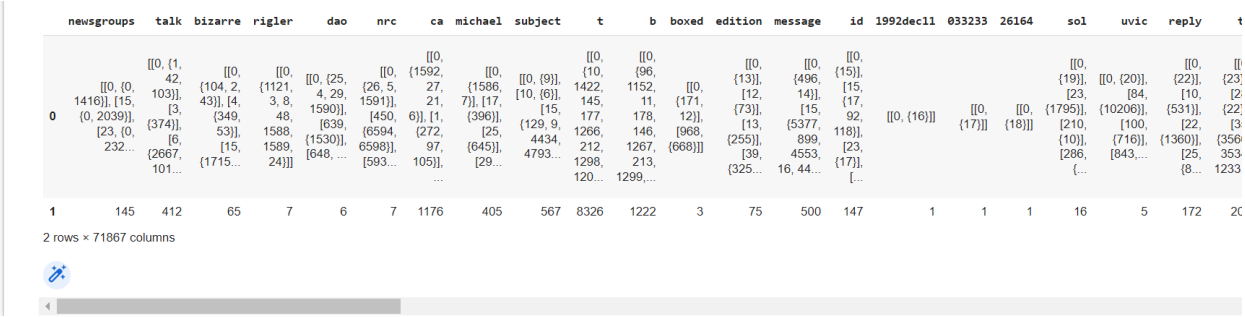
Created the positional index using the dictionary that has a list as a value for each token.
The list itself has two elements:
(i)the list of lists having doc_no and tokens positions set in that doc
(ii)the total occurrence of the token in all the docs
Also maintained dictionary doc_map to store doc_no, filename as key-value pair.



So iterated for each file, read the text, and preprocessed it. Then for all the tokens, updated the positional_index

```
positional_index = {}
doc_map = {}

for path in paths:
    file = open(path, 'r', encoding='ISO-8859-1')
    text = file.read().strip()
    file.close()
    tokens = preprocess_data(text)

    for position, term in enumerate(tokens):
        if term in positional_index :
            positional_index[term][1] += 1
            posting = positional_index[term][0]                    #posting list of token
            pos = find_pos_of_list_having_doc(posting,doc_no)
            if pos != "" :                                         #if doc already added
                positional_index[term][0][pos][1].add(position)    #add position to set
            else:                                                  #if doc added first time
                positional_index[term][0].append([])               #create new list
                positional_index[term][0][-1].append(doc_no)       #append doc_no to the newly appended list(-1 is position of new list)
                positional_index[term][0][-1].append(set())        #append set for positions to the newly appended list
                positional_index[term][0][-1][1].add(position)     #add position to this newly appended set
        else:
            positional_index[term] = []                            #new list as a value for token
            positional_index[term].append([])                      #list at 1st pos for storing posting lists for each doc
            positional_index[term].append(1)                       #occurence count of token at 2nd position
            positional_index[term][0].append([])                   #create new list
            positional_index[term][0][-1].append(doc_no)           #append doc_no to the newly appended list(-1 is position of new list)
            positional_index[term][0][-1].append(set())            #append set for positions to the newly appended list
            positional_index[term][0][-1][1].add(position)         #add position to this newly appended set
    filename = os.path.basename(path)
    doc_map[doc_no] = filename
    doc_no+=1

df = pd.DataFrame(positional_index)
display(df)
```

Positional index after processing all docs and corresponding tokens.



	newsgroups	talk	bizarre	rigler	dao	nrc	ca	michael	subject	t	b	boxed	edition	message	id	1992dec11	033233	26164	sol	uvic	reply	t
0	[[0, {0, 1416}], [15, {0, 2039}], [23, {0, 232...	[[0, {1, 42, 103}], [3, {374}], [6, {2667, 101...	[[0, {104, 2, 43}], [4, {349, 53}], [15, {1715...	[[0, {1121, 3, 8, 48, 1588, 1589, 24}]]	[[0, {25, 4, 29, 1590}], [639, {1530}], [648, ...	[[0, {26, 5, 1591}], [450, {6594, 6598}], [593...	[[0, {1592, 27, 21, 6}], [1, {272, 97, 105}], ...	[[0, {1586, 7}], [17, {396}], [25, {645}], [29...	[[0, {9}], [10, {6}], [15, {129, 9, 4434, 4793...	[[0, {10, 1422, 145, 177, 1266, 212, 1298, 120...	[[0, {96, 1152, 11, 178, 146, 1267, 213, 1299,...	[[0, {171, 12}], [968, {668}]]	[[0, {13}], [12, {73}], [13, {255}], [39, {325...	[[0, {496, 14}], [15, {5377, 899, 4553, 16, 44...	[[0, {15}], [15, {17, 92, 118}], [23, {17}], [...	[[0, {16}]]	[[0, {17}]]	[[0, {18}]]	[[0, {19}], [23, {1795}], [210, {10}], [286, {...	[[0, {20}], [84, {10206}], [100, {716}], [843,...	[[0, {22}], [10, {531}], [22, {1360}], [25, {8...	[[
1	145	412	65	7	6	7	1176	405	567	8326	1222	3	75	500	147	1	1	1	16	5	172	2(

2 rows × 71867 columns

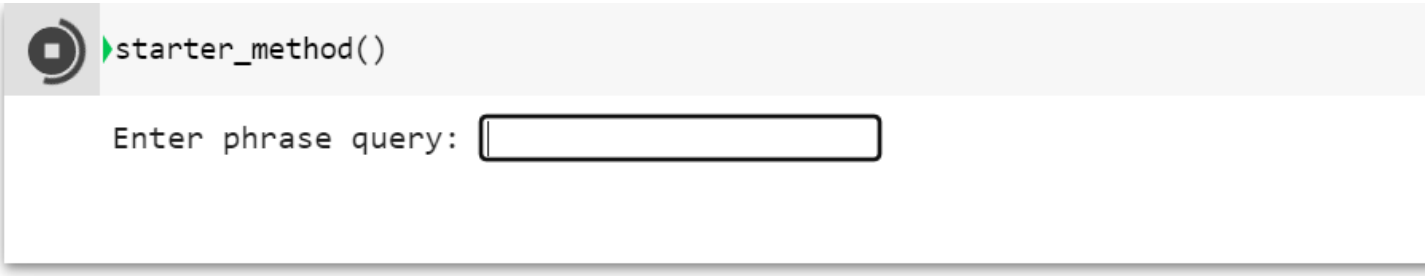
(c) Provide support for the searching of phrase queries

For processing phrase query search, first, we preprocessed the query with the same steps as we did for file processing. Now if the filtered query tokens have only one word then print the docs in its posting list and the count of these docs. For phrase query for more than one word we first find the list of (document, position) pair for first word using **`find_doc_position_pair_list()`** helper method. Now iterate for all remaining words in the phrase query and find if the next word is also in same document but has position one more than that of the previous word.If so then append the doc_no in the result doc list and finally return it as a set.

```
def positional_func(first_word_doc_position_set, query_tokens):
    matched_docs = []
    for a in first_word_doc_position_set:
        doc = a[0]
        pos = a[1]
        token_count = 0

        for token in query_tokens:                          #for all words after first word
            pos = pos+1                                     #to check if next word is on next position
            token_posting = positional_index[token][0]      #posting list of next word
            token_docs = [a[0] for a in token_posting]      #docs list of next word
            if doc in token_docs:                           #if same doc as of first word in next word also
                doc_positions = find_word_positions_in_doc(token_posting, doc)
                if pos in doc_positions:
                    token_count += 1
                else:
                    token_count += 1
                    break
            if token_count == len(query_tokens):
                matched_docs.append(a[0])

    return set(matched_docs)
```

```
###################################   part c   ##########################################

def check_tokens_in_index(tokens):
  for token in tokens:
    if token not in positional_index:
      return False
  return True

def find_word_positions_in_doc(posting_list, doc):
    for a in posting_list:
        if a[0] == doc:
            return a[1]            #return set of word positions in doc
    return {}

def find_doc_position_pair_list(word):
    doc_position_pair_list = []
    word_postings = positional_index[word][0]
    for a in word_postings:
        for position in a[1]:      #for each position of doc a[0]
            doc_position_pair_list.append((a[0], position))
    return doc_position_pair_list
```

Outputs:




```
starter_method()

Enter phrase query: University of Maryland Baltimore County
The number of documents retrieved: 2
The list of document names retrieved: ['top10st1.txt', 'top10st2.txt']
```

```
starter_method()

Enter phrase query: welcome
The number of documents retrieved: 94
The list of document names retrieved: ['st_silic.txt', 'top10st2.txt', 'top10st1.txt', 'top10.txt',
```