ASSIGNMENT-1

Q1.

Code file for Q1 is with name 'IR_assignment1_Q1'.

For implementing inverted index structure:

Step1: We import and download some python and nltk libraries

```
#importing python libraries
import os
import re
import numpy as np
import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
#nltk.download('punkt')
#nltk.download('stopwords')
#nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer
```

Step2: Load the complete data set.

PART A

Step3: Done **pre-processing** like removing punctuation, removing stop words, apostrophe's, converting complete text to lower case, do stemming, converting digit to words, word tokenize etc on complete dataset.

```
#cleaning/preprocessing of data
def cleandata(dt, query):
    dt = converttolower(dt)      #call function for converting to lowercase
    dt = eliminateapos(dt)       #call function for removing to appostrophe
    dt = digittoword(dt)         #call function for converting digit to words
    dt = eliminatestopwords(dt)      #call function for removing stopwords
    dt = eliminatepunctuation(dt)    #call function for removing punctuation
    dt = eliminateonechar(dt)        #call function for eliminating word with one character
    dt = doportStemming(dt)          #call port stemming
    return dt
```

PART B

Step4: Create inverted index data structure with the help of dictionary in python with keys as unique words in the dataset and values will be the list of documents in which these words are present.

```
: Docnumber = 0          #represents current document processed
  docsarray = {}         #posting of complete dataset
  for file in filearray:
      with open(file, 'r', encoding = 'utf-8', errors = 'ignore') as f:
          content = f.read().strip()
          f.close()

          cleantext = cleandata(content, False)              #clean data present in file
          wordarray = word_tokenize(str(cleantext))          #tokenize the data

          for word in set(wordarray):
              if word in docsarray:
                  docsarray[word].append(Docnumber)          #If word already present then append its document number
              else:
                  docsarray[word] = [Docnumber]              #If word not present then create list for it and insert document no
          Docnumber += 1
```

Docsarray will be the required inverted index data structure.

Example: boost word is present in document number 0,117,362 and so on.

```
'boost': [0, 117, 362, 378, 414, 424, 725, 732, 919, 970, 1034, 1035],
```

PART C

Step5: To support different types of queries 4 functions are created:

a) To support 'OR' query.

```
#Or function
def calculateOr(fDoc,sDoc):
    ans = []
    postlist1 = list(docsframe[fDoc])
    postlist2 = list(docsframe[sDoc])
    index1=0
    index2=0
    comparisonmade = 0
    while index1<len(postlist1) or index2<len(postlist2):
        if index1<len(postlist1) and index2<len(postlist2) and (postlist1[index1]==postlist2[index2]):
            comparisonmade +=1
            ans.append(postlist1[index1])
            index1+=1
            index2+=1
        elif index1 < len(postlist1)and index2<len(postlist2) and (postlist1[index1]<postlist2[index2]):
            comparisonmade +=1
            ans.append(postlist1[index1])
            index1+=1
        elif index1 < len(postlist1)and index2<len(postlist2) and (postlist1[index1]>postlist2[index2]):
            comparisonmade +=1
            ans.append(postlist2[index2])
            index2+=1
        elif index1 < len(postlist1):
            ans.append(postlist1[index1])
            index1+=1
        elif index2<len(postlist2):
            ans.append(postlist2[index2])
            index2+=1
    return comparisonmade,set(ans)
```

b) To support 'AND' query.

```
#And function
def calculateAnd(fDoc,sDoc):
    ans = []
    postlist1 = list(docsframe[fDoc])
    postlist2 = list(docsframe[sDoc])
    index1=0
    index2=0
    comparisonmade = 0
    while index1<len(postlist1) and index2<len(postlist2):
        if (postlist1[index1]==postlist2[index2]):
            ans.append(postlist1[index1])
            index1+=1
            index2+=1
        elif (postlist1[index1]<postlist2[index2]):
            index1+=1
        else:
            index2+=1
        comparisonmade+=1
    return comparisonmade,set(ans)
```

c) To support 'NOT' query.

```
#Not function
def calculatenot(dt):
    tempval = set()
    for i in setarray:
        if i not in dt:
            tempval.add(i)
    return tempval
```

d) To support 'AND NOT' query.

```
#AND NOT function
def calculateAndNot(fDoc,sDoc):
    notsDoc = docsframe[sDoc]
    notsDoc = calculatenot(notsDoc)
    return calculateAnd(fDoc,notsDoc)
```

e) To support 'OR NOT' query.

```
#OR NOT function
def calculateOrNot(fDoc,sDoc):
    notsDoc = docsframe[sDoc]
    notsDoc = calculatenot(notsDoc)
    return calculateOr(fDoc,notsDoc)
```

Step6: Sample output of some queries run.

```
Enter no of cases: 2
Please enter your input : I like jeans and shirt
please enter your query : [OR, AND NOT]
['like', 'jean', 'shirt']
['OR', ' AND NOT']
like OR jean
tempResult  AND NOT shirt
Number of documents matched:  717
No. of comparisons required:  704
Please enter your input : I like little half
please enter your query : [OR, AND]
['like', 'littl', 'half']
['OR', ' AND']
like OR littl
tempResult  AND half
Number of documents matched:  794
No. of comparisons required:  793
```

Here, **tempResult** denotes the intermediate calculated result.

Assumption: If given query dataset contain a single word which is not present in our index then it prints '**Data not found**'

Q2.

Code file for Q2 is with name ‘IR_Assignment_1_QUES2’.

PRE-PROCESSING OF THE TEXT:

For this step, we have stripped off URL’s, brackets, stopwords, non-ascii characters, punctuations, underscores (more than three underscores) – all these are parts of denoise the data. Then lowercase the data, converted numbers, tokenization, lemmatization, normalization and stemming was performed.

CREATING THE DATA STRUCTURE:

For a given term: eg.’unbelievable’

positional_index[’unbelievable’] looks like a dictionary with keys as the documents having this term in it with value as TermDetails(‘doc_id’: , ‘positions’:[Where the word exists in the processed the text], term_frequency:[no. of appearance])

{‘doc_name’ : TermDetails(‘doc_id’: , ‘positions’:[Where the word exists in the processed the text], term_frequency:[no. of appearance])}

```
# Function to search the string ####
def searchDocs(string):
  raw_text = denoise_text(string)

  raw_text = replace_contractions(raw_text)


  words = nltk.word_tokenize(raw_text)

  words = normalize(words)
  # print(words)
  stems, lemmas = stem_and_lemmatize(words)
  # print(lemmas)
  print(phrase_query(lemmas))
```

```
##### CREATING DATA STRUCTURE #####
current_directory=pathlib.Path(os.path.abspath(""))
path=current_directory / pathlib.Path("/content/drive/MyDrive/IR_A1_Dataset")
for filename in path.glob("**/*"):
  if not filename.is_file():
    continue
  with filename.open('rb') as f:
    doc_id=filename.name
    raw_text = f.read()
    raw_text = denoise_text(raw_text)
    raw_text = replace_contractions(raw_text)
    words = nltk.word_tokenize(raw_text)
    words = normalize(words)
    stems, lemmas = stem_and_lemmatize(words)
    cleaned_text = " ".join(lemmas)
    for lemma in lemmas:
      positions = []
      for match in re.finditer(lemma,cleaned_text):
        positions.append(match.start())
      if lemma not in positional_index:

        positional_index[lemma]={doc_id:TermDetails(doc_id=doc_id, positions=positions, term_frequency=len(positions))}
      else:
        if doc_id in positional_index[lemma]:
          positional_index[lemma][doc_id].positions.extend(positions)
          positional_index[lemma][doc_id].term_frequency += len(positions)
        else:
          positional_index[lemma][doc_id] = TermDetails(doc_id=doc_id, positions=positions, term_frequency=len(positions))
    # print(filename)
```

```
## FUNCTION TO FIND THE DOCUMENTS FROM THE DATA STRUCTURE ####
def phrase_query(phrase):
  starting_dict_keys=set(positional_index[phrase[0]].keys())
  # print(starting_dict_keys)
  # common_keys=copy.deepcopy()
  for i in range(1,len(phrase)):
    starting_dict_keys=set(positional_index[phrase[i]].keys()).intersection(set(starting_dict_keys))
    # print(starting_dict_keys)
    removed_keys=set()
    for key in starting_dict_keys:

      for k in positional_index[phrase[i]][key].positions:
        # print(k)
        for j in positional_index[phrase[i-1]][key].positions:
          # print(j,k)
          if(k==j+len(phrase[i-1])+1):
            # print(j)
            k+=1
            break
        else:
          continue
        break
      else:
        removed_keys.add(key)
    for el in removed_keys:
      starting_dict_keys.remove(el)
  print("number of documents retrieved: ",len(starting_dict_keys))
  print("list of documents retrieved:")
  print(starting_dict_keys)
```

FOR RUNNING THE FILE:

THE FILE BEST WORKS ON GOOGLE COLAB

Run all the cells in sequence and the last cell asks for a string input.

Example String: ‘came out’