Information Retrieval Assignment- 1

Group No. 78
Nellore Bhavana (MT21131)
Prachi Gupta (MT21135)

Importing necessary libraries

```
import os
import glob
import nltk
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.stem.porter import PorterStemmer
import re
import string
import contractions
import json
from textblob import TextBlob, Word
import joblib
from nltk.tokenize import word_tokenize

nltk.download('punkt')
nltk.download('wordnet')
nltk.download('stopwords')

lema = WordNetLemmatizer()
```

QUESTION 1

Methodology:

Read files

- To read files present in "`Humor,Hist,Media,Food`" folder
  `file_list = os.listdir(r"/content/drive/MyDrive/Datasets/Humor,Hist,Media,Food")`
- To open a particular file
  `rf = open(path,'r',errors = 'ignore')`
- To read content of file
  `rf = rf.read()`

a) Preprocessing steps for the given data :

- Firstly to preprocess the data we have created a cleaning method which takes all the documents and performs the following steps. Function is useful for preprocessing of queries as well as files.
    - Converting the text to the lower case
      - by using .lower() method
    - Removing Punctuations and replacing them with the spaces
      - by using .translate() method
    - Removing Stop words and doing lemmatization
    - Performing tokenization
      - by using word_tokenize()

```
doc = {}
lp = len(string.punctuation)
#Cleaning the data
def cleaning(flag, path, name):
  #For pre-processing queries
  if(flag==0):
    #Expand contractions
    #path = contractions.fix(path)
    #Convert to lower case
    path = path.lower()
    #Remove punctuations and replace with space
    path = path.translate(str.maketrans(string.punctuation, " "*lp, ""))
    path = re.sub('[^A-Za-z\s\n ]+', ' ',path)
    #Function for tokenization
    path = word_tokenize(path)
    #Function for stop words removal and lemmatization
    path = [lema.lemmatize(s) for s in path if s not in stopwords.words('english') and s.isalpha and len(s)>1]
    return path

  else:
    #For pre-processing of files
    rf = open(path,'r',errors = 'ignore')
    #Read the file
    rf = rf.read()
    #Convert to lower case
    clean = rf.lower()
    #Remove punctuations and replace with space
    clean = clean.translate(str.maketrans(string.punctuation, " "*lp, ""))
    clean = re.sub('[^A-Za-z\s\n ]+', ' ',clean)
    #Function for tokenization
    clean = word_tokenize(clean)
    #Function for stop words removal and lemmatization
    clean = [lema.lemmatize(s) for s in clean if s not in stopwords.words('english') and s.isalpha and len(s)>1]
    doc[name] = clean
```

b) Implementation of Unigram Inverted Index data structure :

- We created a dictionary to store the inverted index which contains the terms and their posting lists

```
invertedIndex = {}
final = []
#Final set of words
file = open('vocabulary.txt','w')
for doc_name in doc:
    for word in doc[doc_name]:
        final.append(word)
final = set(final)
for word in final:
   file.write(word+" , ")
file.close
#invertedIndex (dictinary) contains term and posting list
#Build posting list containing all doc id's where term exist
post = open('posting.txt','w')
for term in final:
    postingList = []
    post.write("\n"+term + " - ")
    for id in doc:
        if term in doc[id]:
            postingList.append(id)
            post.write(id+" , ")
    invertedIndex[term] = postingList
```

- After completing all the preprocessing steps we created 2 files
    - Vocabulary.txt - This document contains all the final words

vocabulary.txt

```
1 foray , criminologist , knitting , patched , scmitt , pastu
  nocturnal , crabgrass , saline , rained , nationwide , chol
  occlude , coarsley , nuance , nessman , demonews , buddy ,
  , hpa , toadie , mansour , kiljander , footpounds , ediing
  prostitute , 404 , ccny , paraphrase , uky , norma , inordi
  unwarranted , ski , periodic , enters , solitaire , biodyna
  shwartzenegger , fenneman , aolers , 88693 , murukh , punta
  uniquely , nikki , aghast , simulacrum , nicer , spangled ,
  rolling , afullam , ruggieri , avalanche , ganged , peugh ,
  , charter , picalilli , pointless , 1st , magoo , burrito ,
  , intently , rhyme , pamphlet , parading , mccarthyism , 01
  sfrpg , flawless , 8077 , pleasepleasepleasepleaseplease ,
  alpha , distended , flecked , clerical , methrin , alpert ,
  concocts , highlight , handy , sesms , chalkboard , molest
  electron , illustration , bassoon , conceptualized , acos ,
  continuity , hef , 0192 , garnering , ience , rinsing , inn
  , dreading , vtvm , mashed , casaleggi , gently stream ,
  extend , usss , babs , michener , cush , straczynski , disp
  february , cannister , dvm , trunk , compusex , diplomatic
```

- posting.txt - This document contains all the documents Id's where the terms exists

posting.txt

```
1
2 foray - kanalx.txt , hackmorality.txt , radexposed.txt ,
3 criminologist - cybrtrsh.txt ,
4 knitting - college.hum , arnold.txt ,
5 patched - prac3.jok , practica.txt , hacktest.txt , humor9
6 scmitt - manspace.hum ,
7 pasture - macsfarm.old , all_grai ,
8 lucipher - college.hum ,
9 food - the_math.hel , thecube.hum , skincat , top10st2.txt
10 atkinson - hell.jok , blackadd , hbo_spec.rev ,
11 hengst - crzycred.lst ,
12 so - hacktest.txt , llamas.txt , acronyms.txt ,
13 witching - oldtime.sng ,
14 peritoneal - antibiot.txt ,
15 regularly - taping.hum , top10st1.txt , racist.net , quote
16 deunan - stuf10.txt ,
17 thomajan - missdish ,
18 lookup - humor9.txt , acronyms.txt ,
19 tbls - bakebred.txt , batrbred.txt , egg-bred.txt , engmut
```

- To Store all the data obtained we used joblib.dump()

```
[ ] #dump the invertedIndex using joblib
    joblib.dump(invertedIndex, '/content/drive/MyDrive/invertedIndex')

    ['/content/drive/MyDrive/invertedIndex']
```

- Loaded the inverted index in posting using joblib.load()

```
[ ] #Load invertedIndex using joblib in postings
    invertedIndex = joblib.load('/content/drive/MyDrive/invertedIndex')
    postings = invertedIndex
```

c) Implementation of Queries :

Preprocessing input queries and operators

```
[ ] #Pre-process queries and operators
    #Use flag = 0
    qp=[]
    def qprocess(operator, query):
      #Pre-process the given operator
      operator = re.sub("[^a-zA-Z\s\n\,]", " ",operator)
      op = operator.split(",")
      op = [x.strip(" ") for x in op]
      op = [x.upper() for x in op]
      #Pre-process the given query
      qp = cleaning(0, query, " ")
      return op, qp
```

(i) x OR y:

Take two words at a time and find their posting list. Append those to the final list and remove the intersection.

```
if op[i] == 'OR':
  if i == 0 :
    if (w1 in postings.keys()):
     doc1 = postings[w1].copy()
    if (w2 in postings.keys()):
     doc2 = postings[w2].copy()
  else :
    doc1=[]
    doc1 = finalList
    finalList = []
    w2 = qfinal[i+1]
    if (w2 in postings.keys()):
        doc2=[]
        doc2 = postings[w2].copy()
  for item in doc1:
      c += 1
      finalList.append(item)
      if item in doc2:
        doc2.remove(item)
  for item in doc2:
    c += 1
    finalList.append(item)
```

(ii) x AND y:

Similarly take two words and find their posting list. Perform intersection and add to the final list.

```
#Take two words at a time, find posting list, find intersection and add to the final doc
if op[i] == 'AND':
  if i == 0 :
    if (w1 in postings.keys()):
     doc1 = postings[w1].copy()
    if (w2 in postings.keys()):
     doc2 = postings[w2].copy()
  else :
    doc1=[]
    doc1 = finalList
    finalList = []
    w2 = qfinal[i+1]
    if (w2 in postings.keys()):
        doc2 = []
        doc2 = postings[w2].copy()
  for item in doc1:
      if item in doc2:
        c +=1
        finalList.append(item)
```

(iii) x AND NOT y:

Find a posting list of two words then add the posting list of first word in the final list. Now, removing the intersection of both the words.

```
#Find posting list of two words, add posting list of w1 in final doc, remove inte
if op[i] == 'AND NOT':
  if i == 0 :
    if (w1 in postings.keys()):
     doc1 = postings[w1].copy()
    if (w2 in postings.keys()):
     doc2 = postings[w2].copy()
    for item in doc1:
      c += 1
      finalList.append(item)
      if item in doc2:
        finalList.remove(item)
  else :
    doc1=[]
    doc1 = finalList
    finalList = []
    w2 = qfinal[i+1]
    if (w2 in postings.keys()):
        doc2=[]
        doc2 = postings[w2].copy()
    for item in doc1:
      finalList.append(item)
      if item in doc2:
        finalList.remove(item)
        c += 1
```

(iv) x OR NOT y:

Find posting for first word, find postings for second word where it doesn't occur. Perform OR operation.

```
#Find posting list of w1 and find postings in which 2 is not present, then do OR o
if op[i] == 'OR NOT':
  if i == 0 :
    if (w1 in postings.keys()):
     doc1 = postings[w1].copy()
    for item in postings.keys():
       if item != w2:
         for u in postings[item]:
            if u not in doc2:
              doc2.append(u)
  else :
    doc1 = []
    doc1 = finalList
    finalList = []

    w2 = qfinal[i+1]
    doc2 = []
    for item in postings.keys():
       if item != w2:
         for u in postings[item]:
              if u not in doc2:
                 doc2.append(u)
  for item in doc1:
      c += 1
      finalList.append(item)
      if item in doc2:
        doc2.remove(item)

  for item in doc2:
    c += 1
    finalList.append(item)
```

Query Evaluation:

```
[ ]  print("Number of queries: ")
     N = int(input())
     for j in range(N):
       print("Input query: ")
       query = input()
       print("Input operation sequence: ")
       operation = input()
       op, qfinal = qprocess(operation, query)
       merge_pl(op, qfinal)
```

d) Queries Output :

```
Number of queries:
2
Input query:
lion stood thoughtfully for a moment
Input operation sequence:
OR, OR , OR
Number of documents matched:  213
No. of comparisons required:  414
['murphys.txt', 'llong.hum', 'onetotwo.hum', 'deep.txt', 'drunk.txt', 'hecomes.jok', 'wag
Input query:
telephone,paved, roads
Input operation sequence:
OR NOT, AND NOT
Number of documents matched:  998
No. of comparisons required:  1272
['cartoon_.txt', 'looser.hum', 'losers84.hum', 'cartoon.law', 'raven.hum', 'radiolaf.hum',
```

Assumptions

- We assumed that user enters the valid query and operations among “AND, OR, AND NOT, and OR NOT”
- The operations on query are computed from **Left to right**
- After pre-processing of input query, if length of tokenizing string of query is m then length of input query operator should be (m-1)

QUESTION 2

Methodology:

Read files

- To read files present in "`Humor,Hist,Media,Food`" folder
  `path = "/content/drive/MyDrive/Datasets/Humor,Hist,Media,Food"`
- To open a particular file
  `rf = open(path,'r',errors = 'ignore')`
- To read content of file
  `rf = rf.read()`

a) Preprocessing steps for the given data :

(i) Converting the text to lower case
(ii) Word Tokenization
(iii) Removing the stop words from tokens
(iv) Removing the punctuation marks from tokens
(v) Removing the blank spaces tokens

```
[ ]  ln = len(string.punctuation)
     #Cleaning the data
     def process(content):
       #Convert the text to lower case
       content = content.lower()
       #Remove punctuation marks from tokens
       content = content.translate(str.maketrans(string.punctuation, " "*ln,''))
       #Perform word tokenization
       ctokens = word_tokenize(content)
       #Remove stopwords from tokens and do lemmatization
       #Checking length, if length = 1
       ctokens = [lema.lemmatize(s) for s in ctokens if s not in stopwords.words('english') and s.isalpha and len(s)>1]
       return ctokens
```

b) Implementing the Positional Index Data Structure

- For each word we stored the document list and position of each word in each document in a dictionary

- All the obtained positional indexes are stored in a json file `lposting.json` to easily fetch for further processing

```
[ ]  #Function to create posting list
     #Positional index
     pol={}
     def lposting(c, tl):
       i = 0
       for t in tl:
         i = i+1
         if t in pol:
           d = pol[t][1]
           if c in d:
             d[c].append(i)
           else:
             pol[t][0] = (pol[t][0]+1)
             d[c] = [i]
           pol[t][1] = d
         else:
           pol[t] = []
           pol[t].append(1)
           pol[t].append({})
           pol[t][1][c] = [i]
```

```
[ ]  #Iterate over the dataset for preprocessing and indexing
     c = 0
     for file in glob.glob("*"):
       c = c+1
       #Reading the file
       with open(file,'r',encoding = 'utf8', errors = 'ignore') as f:
         rfile = f.read()
         doc[c] = file
         tl = process(rfile)
         lposting(c,tl)

     print(c)
```

Check the posting list

```
[ ]  #Print terms in posting list
     for t in pol:
       print(t,pol[t][0],len(pol[t][1]))
```

Output of the terms in Posting List:

```
aafte 1 1
iers 1 1
orxogrefkl 1 1
riform 1 1
lojikl 1 1
kohirnt 1 1
xrewawt 1 1
ingliy 1 1
werld 1 1
iorz 1 1
feixfuli 1 1
yilz 1 1
spontaniously 1 1
crist 1 1
demean 1 1
generosity 1 1
havard 1 1
analyse 1 1
brookings 1 1
stephenson 1 1
diemer 1 1
carothers 1 1
```

Save the posting list for further use

```
[ ] #Dumping the posting index to avoid making posting index from scratch every time
    save = open('lposting.json','w')
    save = json.dump(pol,save)
```

```
[ ] #Loading the posting index
    load_file = open('posting_doc.json','w')
    load_file = json.dump(pol,load_file)
```

c) Support for searching of phrase queries

```
[ ] #Checking common documents for the query in the list
    def dcommon(qf):
      n = len(qf)
      cd = []
      l = pol[qf[0]][1]
      for d in l:
        c=1
        for i in range(1, n):
          if d in pol[qf[i]][1]:
            c = c+1
        if c == n:
          cd.append(d)
      return cd
```

```
[ ]  #Retrieving the list of positions for words from the common documents
     def word_pos(qf,cd):
       pol1 = {}
       for d in cd:
         for t in qf:
           if d in pol1:
             pol1[d].append(pol[t][1][d])
           else:
             pol1[d] = [pol[t][1][d]]

       return pol1
```

d) Queries Output

Query Processing :

` Preprocess the input queries before evaluation

```
[ ]  #Function for query processing
     def qprocess(qf):
       f_doc = []
       print(qf)
       cd = dcommon(qf)
       plist1 = word_pos(qf,cd)
       for d in cd:
         f = False
         tp = plist1[d]
         tp1 = tp[0]
         for i in range(len(tp1)):
           f1 = True
           c = tp1[i]
           for j in range(1,len(tp)):
             c = c+1
             if c not in tp[j]:
               f1 = False
               break
           if(f1):
             f = True
         if(f):
           f_doc.append(d)
         f_doc = list(set(f_doc))
       return f_doc
```

Query Evaluation:

```
[ ]  #Evaluate query
     print("Enter Query")
     #Input the query
     query = input()
     #preprocessing input query
     qf = process(query)
     #Processing input query
     d = qprocess(qf)
     #Listing documents
     final_doc = []
     for i in d:
       final_doc.append(doc[i])

     print(f"\nThe number of documents retrieved : {len(d)}")
     print("The list of document names retrieved : ")
     print(final_doc)
```

Output:

```
Enter Query
Good Day
['good', 'day']

The number of documents retrieved : 9
The list of document names retrieved :
['xibovac.txt', 'tnd.1', 'onetoone.hum', 'horoscop.jok', 'horoscop.txt', 'lozeuser.hum', 'humor9.txt', "terrmcd'.hum", 'valujet.txt']
```