CS4201 Assignment-1

Aakash Ghosh
19MS129

January 2023

1 Package installs

We first import the needed packages. Packages can be installed using `pip` as given below.

```
PS C:\Users\ghosh> pip install nltk
```

On a fresh download of NLTK, run the following command on a python console to download the required modules:

```
>>> import nltk
>>> nltk.download('popular')
```

2 Package imports

We import the following packages:

1. `import csv` for reading csv files
2. `from nltk.tokenize import word_tokenize` for calling the tokenizer
3. `from nltk.stem.porter import *` for calling the porter stemmer

We also create a stemmer object.

```
import csv
from nltk.tokenize import word_tokenize
from nltk.stem.porter import *
stemmer = PorterStemmer()
```

3 Class Creation

We create a class called `film` for easy access to film data which will be read eventually. We provide a constructor for initialization at creation.

```
class film:
    def __init__(self,year,name,origin,director,cast,genre,url,plot):
        self.year=year
        self.name=name
        self.origin=origin
        self.director=director
        self.cast=cast
        self.genre=genre
        self.url=url
        self.plot=plot
```

4 Array Functions

We create two functions: one for performing a binary search and the other for inserting a string in a sorted array in place.

```
def Bin_Serch(arr, key):
    first = 0
    last = len(arr) - 1
    while first <= last:

        mid = (last + first) // 2
        if arr[mid] < key:
            first = mid + 1
        elif arr[mid] > key:
            last = mid - 1
        else:
            return mid
    return -1

def Sort_Insert(arr,key):
    i = 0
    while i<len(arr):
        if arr[i]>key:
            break
        i=i+1
    arr.insert(i,key)
    return arr
```

5 Reading and Tokenizing the data

We create three arrays: `FilmArr,TokArr,WordList`. The i^{th} index of `FilmArr` contains the info about the $i + 1^{th}$ film. The i^{th} index of `TokArr` contains the tokens in the plot of $i + 1^{th}$ film. `WordList` is our bag of words. For each entry, we find if any word is missing in the word list from the list of tokens found and correspondingly update the word list. Since we use our define for search and insertions, the word list always remain sorted.

```
FilmArr=[]
TokArr=[]
WordList=[]
with open("file.csv","r",encoding='utf-8') as file:
    reader=csv.reader(file)
    for line in reader:
        filmOb=film(line[0],line[1],line[2],line[3],line[4],line[5],line[6],line[7])
        tok=[]
        tok=tok+word_tokenize(filmOb.plot)
        l=len(tok)
        i=0
        while(i<l):
            tok[i]=stemmer.stem(tok[i].lower())
            if (Bin_Serch(WordList,tok[i])==-1):
                WordList=Sort_Insert(WordList,tok[i])
            i=i+1
        tok.sort()
        TokArr.append(tok)
        FilmArr.append(filmOb)
```

6 Creation of term incidence matrix

The term incidence matrix `inMat[][]` is created by initializing a zero matrix of required size and then updating each row by going through `TokArr`. As our bag of words is sorted, we can use the functions we have defined for looking up the required positions. `inMAt[i][j]` will contain number of occurance of token `EordList[j]` in `FilmArr[i].plot`

```
inMAt=[[0]*len(WordList)for _ in range(len(TokArr))]
i=0
while(i<len(TokArr)):
    lastTok=""
    lastIndex=-1
    for j in TokArr[i]:
        if j==lastTok:
            inMAt[i][lastIndex]=inMAt[i][lastIndex]+1
        else:
            lastTok=j
            lastIndex=Bin_Serch(WordList,j)
            inMAt[i][lastIndex]=inMAt[i][lastIndex]+1
    i=i+1
```

7 Creation of inverted index

The inverted index `invertedIndex[][]` is created by going through each column of `inMAt[][]` and picking up the non zero entries.

```
invertedIndex=[]
i=0
while(i<len(WordList)):
    imCol=[]
    j=0
    while(j<len(FilmArr)):
        if inMAt[j][i]!=0:
            imCol.append(j)
        j=j+1
    i=i+1
    invertedIndex.append(imCol)
```

8 Creation of Array operations

We define functions for **binary** operations **AND**, **OR** and **ANDNOT**

```
def arrAnd(arr1,arr2):
    arr=[]
    for i in arr1:
        if i in arr2:
            arr.append(i)
    return arr

def arrOr(arr1,arr2):
    arr=[]
    for i in arr1:
        arr.append(i)
    for i in arr2:
        flag=1
        for j in arr1:
            if i==j:
                flag=0
        if flag:
            arr.append(i)
    return arr

def arrAndNot(arr1,arr2):
    arr=[]
    for i in arr1:
        if i not in arr2:
            arr.append(i)
    return arr
```

9 Creation of Boolean Query Retrieval Function

With the proper use of brackets we can assume that the presence of `")"` indicates a completing of a term that can be computed. The array that is passed to this function consists of inverted index

entries of the queried words, query operators and parenthesis.

```
def query(arr):
    try:
        i=0
        stack=[]
        while(i<len(arr)):

            if arr[i]==")":
                if stack[-2]=="AND":
                    temp=arrAnd(stack[-1],stack[-3])
                    stack.pop()
                    stack.pop()
                    stack.pop()
                    stack.pop()
                    stack.append(temp)
                elif stack[-2]=="OR":
                    temp=arrOr(stack[-1],stack[-3])
                    stack.pop()
                    stack.pop()
                    stack.pop()
                    stack.pop()
                    stack.append(temp)
                else:
                    temp=arrAndNot(stack[-1],stack[-3])
                    stack.pop()
                    stack.pop()
                    stack.pop()
                    stack.pop()
                    stack.append(temp)
            else:
                stack.append(arr[i])
            i=i+1
        return stack[0]
    except:
        print("Error in query processing")
        return -1
```

10 The main loop

This is the main loop that will take queries until a numeric entry is provided. We tokenize the input based on the presence of parenthesis and spaces. The query words are identified, stemmed and sent to the function defined above. The returned array contains only those movie indexes that satisfy the query criteria. The name of the movies is accessed from `filmArr[]` and printed

```
while(1):
    str=input("Enter query. Query words are to be given in small case and Query
    ↪  relation operators are to be given in upper case. Query realtion operators
    ↪  are considered to be binary realations. Enter a digit to quit.\n")
```

```
str=str+" "
if(str.isnumeric()):
    break
tok=[]
tempStr=""
i=0
while(i<len(str)):
    if str[i]=="(":
        tok.append(tempStr)
        tok.append("(")
        tempStr=""
        i=i+1
    elif str[i]==")":
        tok.append(tempStr)
        tok.append(")")
        tempStr=""
        i=i+1
    elif(str[i]==" "):
        tok.append(tempStr)
        tempStr=""
        i=i+1
    else:
        tempStr="".join([tempStr,str[i]])
        i=i+1
for i in tok:
    if(len(i)==0):
        tok.remove(i)
i=0
while(i<len(tok)):
    if
    ↪  (tok[i]!="AND")and(tok[i]!="OR")and(tok[i]!="ANDNOT")and(tok[i]!="(")and(tok[i]!=")"):
        tok[i]=stemmer.stem(tok[i].lower())
        r=Bin_Serch(WordList,tok[i])
        if(r==-1):
            tok[i]=[]
        else:
            tok[i]=invertedIndex[r]
    i=i+1

arr=query(tok)
if(arr!=-1):
    print("Match list is:")
    l=len(arr)
    i=0
    while(i<l):
        print(FilmArr[arr[i]].name)
        i=i+1
```

11 Sample use(on a subset of the given corpus)

11.1 Case 1: Single word, with matches

```
Enter query. Query words are to be given in small case and Query relation
↪  operators are to be given in upper case. Query realtion operators are
↪  considered to be binary realations. Enter a digit to quit.
scene
Match list is:
Love by the Light of the Moon
The Little Train Robbery
The Night Before Christmas
Hemlock Hoax, the Detective
What the Daisy Said
```

11.2 Case 2: Single word, without matches

```
Enter query. Query words are to be given in small case and Query relation
↪  operators are to be given in upper case. Query realtion operators are
↪  considered to be binary realations. Enter a digit to quit.
phone
Match list is:
```

11.3 Case 3: Multiple words with relational operators

```
(and AND (scene AND when))
Match list is:
The Little Train Robbery
What the Daisy Said
Enter query. Query words are to be given in small case and Query relation
↪  operators are to be given in upper case. Query realtion operators are
↪  considered to be binary realations. Enter a digit to quit.
(and AND (scene OR when))
Match list is:
Jack and the Beanstalk
Alice in Wonderland
The Little Train Robbery
Dream of a Rarebit Fiend
How Brown Saw the Baseball Game
A Christmas Carol
What the Daisy Said
Love by the Light of the Moon
The Night Before Christmas
Hemlock Hoax, the Detective
```