Assessment 3

-17BCB0102

-Vaibhav Bhandari

A) Consider the following two sentences

1. **Term frequency matrix is important for ranking docs.**
2. **TFIDF is more important than Term frequency matrix for the same.**

- **Find TF MATRIX, IDF values of each term and finally TF*IDF MATRIX.**
- **Find cosine similarity also.**

Ans)

Code-

```
from collections import Counter

import pandas as pd

def computeTF(wordDict,bow):

    tfDict={}

    bowCount=len(bow)

    for word, count in wordDict.items():

        tfDict[word]=count/float(bowCount)

    return tfDict

def computeIDF(docList):

    import math

    idfDict = {}

    N = len(docList)

    idfDict = dict.fromkeys(docList[0].keys(), 0)

    for doc in docList:

        for word, val in doc.items():

            if val > 0:

                idfDict[word] += 1

    for word, val in idfDict.items():
```

```
        idfDict[word] = math.log10(N / float(val))
    return idfDict
def computeTFIDF(tfBow,idfs):
    tfidf={}
    for word,val in tfBow.items():
        tfidf[word]=val*idfs[word]
    return tfidf
S1="Term frequency matrix is important for ranking docs."
S2="TFIDF is more important than Term frequency matrix for the same"
bowA=S1.split(" ")
bowB=S2.split(" ")
wordSet=set(bowA).union(set(bowB))
wordDictA = dict.fromkeys(wordSet, 0)
wordDictB = dict.fromkeys(wordSet, 0)
for word in bowA:
    wordDictA[word]+=1
for word in bowB:
    wordDictB[word]+=1
print()
print("The TF matrix is::")
print(pd.DataFrame([wordDictA,wordDictB]))
tfBowA=computeTF(wordDictA,bowA)
tfBowB=computeTF(wordDictB,bowB)
print()
print("The IDF values of each term::")
print(tfBowA)
print()
print(tfBowB)

idfs = computeIDF([wordDictA,wordDictB])
```

```
tfidfBowA = computeTFIDF(tfBowA, idfs)

tfidfBowB = computeTFIDF(tfBowB, idfs)

print()

print("The TF*IDF Matris::")

print(pd.DataFrame([tfidfBowA, tfidfBowB]))
```

Output—

```
The TF matrix is::
   TFIDF  Term  docs.  for  frequency  important  is  matrix  more  ranking  same  than  the
0      0     1      1    1          1          1   1       1     0        1     0     0    0
1      1     1      0    1          1          1   1       1     1        0     1     1    1

The IDF values of each term::
{'frequency': 0.125, 'TFIDF': 0.0, 'the': 0.0, 'than': 0.0, 'same': 0.0, 'docs.': 0.125, 'is': 0.125, 'm
ore': 0.0, 'ranking': 0.125, 'Term': 0.125, 'for': 0.125, 'matrix': 0.125, 'important': 0.125}

{'frequency': 0.09090909090909091, 'TFIDF': 0.09090909090909091, 'the': 0.09090909090909091, 'than': 0.0
9090909090909091, 'same': 0.09090909090909091, 'docs.': 0.0, 'is': 0.09090909090909091, 'more': 0.090909
09090909091, 'ranking': 0.0, 'Term': 0.09090909090909091, 'for': 0.09090909090909091, 'matrix': 0.090909
09090909091, 'important': 0.09090909090909091}

The TF*IDF Matris::
      TFIDF  Term     docs.  for  frequency    ...        more   ranking      same      than       the
0  0.000000   0.0  0.037629  0.0        0.0    ...    0.000000  0.037629  0.000000  0.000000  0.000000
1  0.027366   0.0  0.000000  0.0        0.0    ...    0.027366  0.000000  0.027366  0.027366  0.027366

[2 rows x 13 columns]


------------------
(program exited with code: 0)

Press any key to continue . . .
```

B) Implement PAGE RANK ALGORITHM. Take input for adjacency matrix (no need to visualise the directed graph), find stochastic matrix, find transpose of it. Consider dumping factor 0.7. Consider initial P values as all 1s. You can consider 5 nodes. Calculate page rank until 2 iterations and display the ranks.

Ans)

Code-

```
import numpy as np

print("Enter the nuber of rows and columns::")

a=int(input())

print("Enter the adjacency matrix(row wise)(in the form of a matrixi)::\n")

mat=np.zeros((a,a))

for i in range(a):

	mat[i]=input().split(" ")

print("\nThe entry is::\n")

print(mat)

print("\nEnter the number of iterations::\n")

n=int(input())

print("\nEnter the dumping factor::\n")

m=float(input())


for i in range(a):

	flag=1

	for j in range(a):

		if(mat[i,j]!=0):

			flag=0

	if flag==1:

		for j in range(a):

			mat[i,j]=1

print("\nThe adjacency matrix is::\n")

print(mat)

for i in range(a):

	count=0

	for j in range(a):

		if(mat[i,j]==1):

			count+=1

	for j in range(a):
```

```
                    if(mat[i,j]==1):
                              mat[i,j]/=count
print("\nThe schotastic matrix is::\n")
print(mat)
B=mat.transpose()
print("\nThe transpose Schotastic matrix is::\n")
print(B)
print("\nThe matrix C is::\n")
C=(1.0-m)*(1/a)+(m)*B
print(C)
print("\nThe initial value of P is::\n")
P=np.array([[1],[1],[1],[1],[1]]);
print(P)
for i in range(n):
          P=np.dot(C,P)
          print("\nThe ranks in iteration ",i+1,"are::\n")
          print(P)
```

Output—

```
Enter the nuber of rows and columns::
5
Enter the adjacency matrix(row wise)(in the form of a matrixi)::

0 1 0 1 0
1 1 1 1 0
0 0 0 0 1
1 0 0 1 1
0 0 0 0 0

The entry is::

[[0. 1. 0. 1. 0.]
 [1. 1. 1. 1. 0.]
 [0. 0. 0. 0. 1.]
 [1. 0. 0. 1. 1.]
 [0. 0. 0. 0. 0.]]

Enter the number of iterations::

2

Enter the dumping factor::

0.7

The adjacency matrix is::

[[0. 1. 0. 1. 0.]
 [1. 1. 1. 1. 0.]
 [0. 0. 0. 0. 1.]
 [1. 0. 0. 1. 1.]
 [1. 1. 1. 1. 1.]]

The schotastic matrix is::

[[0.         0.5        0.         0.5        0.        ]
 [0.25       0.25       0.25       0.25       0.        ]
 [0.         0.         0.         0.         1.        ]
 [0.33333333 0.         0.         0.33333333 0.33333333]
 [0.2        0.2        0.2        0.2        0.2       ]]
```

```
The transpose Schotastic matrix is::

[[0.         0.25       0.         0.33333333 0.2       ]
 [0.5        0.25       0.         0.         0.2       ]
 [0.         0.25       0.         0.         0.2       ]
 [0.5        0.25       0.         0.33333333 0.2       ]
 [0.         0.         1.         0.33333333 0.2       ]]

The matrix C is::

[[0.06       0.235      0.06       0.29333333 0.2       ]
 [0.41       0.235      0.06       0.06       0.2       ]
 [0.06       0.235      0.06       0.06       0.2       ]
 [0.41       0.235      0.06       0.29333333 0.2       ]
 [0.06       0.06       0.76       0.29333333 0.2       ]]

The initial value of P is::

[[1]
 [1]
 [1]
 [1]
 [1]]

The ranks in iteration  1 are::

[[0.84833333]
 [0.965     ]
 [0.615     ]
 [1.19833333]
 [1.37333333]]

The ranks in iteration  2 are::

[[0.94075278]
 [0.95805833]
 [0.66114167]
 [1.23766944]
 [1.20237778]]


------------------
(program exited with code: 0)

Press any key to continue . . .
```

C) Implement Ellias Gamma, Ellias Delta and Golomb coding

Ans)

```
from math import log,ceil

log2 = lambda x: log(x,2)

def binary(x,l=1):

    fmt = '{0:0%db}' % l

    return fmt.format(x)

def unary(x):

    return x*'1'+'0'

def elias_generic(lencoding, x):

    if x == 0: return '0'

    l = 1+int(log2(x))

    a = x - 2**(int(log2(x)))

    k = int(log2(x))

    return lencoding(l) + binary(a,k)

def golomb(b, x):

    q = int((x) / b)

    r = int((x) % b)

    l = int(ceil(log2(b)))

    #print q,r,l

    return unary(q) + binary(r, l)

def elias_gamma(x):

    return elias_generic(unary, x)

def elias_delta(x):

        return elias_generic(elias_gamma,x)

print("Gamme Coding,   \tElias Coding, \tGoulomb Coding")

for i in range(11):

    print ("%5d: %-10s\t : %-10s\t : %-10s\t" %(i,elias_gamma(i),elias_delta(i), golomb(3,i)))
```

Output—

```
Gamme Coding,            Elias Coding,    Goulomb Coding
    0: 0                  : 0              : 000
    1: 100                : 1000           : 001
    2: 1100               : 11000          : 010
    3: 1101               : 11001          : 1000
    4: 111000             : 110100         : 1001
    5: 111001             : 110101         : 1010
    6: 111010             : 110110         : 11000
    7: 111011             : 110111         : 11001
    8: 11110000           : 111000000      : 11010
    9: 11110001           : 111000001      : 111000
   10: 11110010           : 111000010      : 111001


------------------
(program exited with code: 0)

Press any key to continue . . .
```