Name : B.M.G.G.K. Rajapaksha

Index No. : S14210

Faculty : Faculty of Science, Bioinformatics

CS 4104 – Data Analytics – Assignment 1

a. Screenshots with an explanation of the tools you used for the above-mentioned *Document Similarity* implementation.

```
# STEP 1 -Import libraries
from sklearn.feature_extraction.text import TfidfVectorizer
from collections import OrderedDict
from sklearn.metrics.pairwise import cosine_similarity
import pandas as pd
```

sklearn (scikit – learn) – Sklearn is a highly recommended and wide using machine learning library package for the python programming. It consists set of machine learning tools that are used in most machine learning problems.

sklearn.feature_extraction.text – It is a module in sklearn library which can be used to extract features in a format supported by machine learning algorithms from datasets consisting of formats such as text.

TfidfVectorizer – It converts a collection of raw documents to a matrix of TE-IDF (Term Frequency Inverse Document Frequency) features.

As per sklearn's online documentation, It uses the below method to calculate tf and idf of a term in a document.

- tf(t) = (No. of times a term occurs in the document)/(No. of terms in the document)
- idf(t) = $\log_e [(1+n)/(1+df(t))]+1$ (**default i:e smooth_idf = True**)
- idf(t) = $\log_e [(n/df(t)]+1$ (when smooth_idf = False)

Where;
n : Total no. of documents available
t : Term for which idf value has to be calculated
df(t) : No. of documents in which the term t appears.

```
# STEP 4 - Fit the corpus in to the vectorizer with preprocessing
parameters

# convert all texts into lowercase and remove stop words and initiate the
TfidfVectorizer
vectorizer = TfidfVectorizer(lowercase=True, stop_words='english')

# fit the corpus into the vectorizer
tf_idf_matrix = vectorizer.fit_transform(corpus)
print('TF-IDF matrix : ', tf_idf_matrix)
```

Output

```
TF-IDF matrix :    (0, 833) 0.04123167956458735
  (0, 760)  0.03524333866473208
  (0, 752)  0.0824633591291747
  (0, 1235) 0.04123167956458735
  (0, 174)  0.04123167956458735
  (0, 1220) 0.030994543179823038
  (0, 862)  0.04123167956458735
  (0, 1093) 0.04123167956458735
  (0, 440)  0.04123167956458735
  (0, 294)  0.0824633591291747
  (0, 1236) 0.04123167956458735
  (0, 574)  0.04123167956458735
  (0, 1244) 0.04123167956458735
  (0, 95)   0.04123167956458735
  (0, 277)  0.04123167956458735
  (0, 961)  0.04123167956458735
  (0, 379)  0.04123167956458735
  (0, 960)  0.04123167956458735
  (0, 525)  0.04123167956458735
  (0, 831)  0.03524333866473208
  (0, 19)   0.030994543179823038
  (0, 292)  0.030994543179823038
  (0, 1112) 0.03524333866473208
  (0, 506)  0.04123167956458735
  (0, 886)  0.04123167956458735
  : :
```

collections – This is a python module which implements specialized container data types providing alternatives to pythons's general purpose built-in containers such as dict, list, set, and tuple.

OrderedDict – This is the dict subclass in collections module that remembers the order entries were added.

```
# take the vocabulary into a sorted dictionary
sorted_dict = OrderedDict(sorted(vectorizer.vocabulary_.items(), key=lambda
x: x[1], reverse=False))
print('Sorted vocabulary\n', sorted_dict, '\n')
print('Length of the vocabulary - ', len(vectorizer.vocabulary_), '\n')
```

Output

```
Sorted vocabulary
 OrderedDict([('000', 0), ('10', 1), ('100', 2), ('11', 3), ('110', 4), ('115', 5), ('12', 6), ('125', 7), ('14', 8), ('140', 9), ('15', 10),

Length of the vocabulary -  1260
```

sklearn.matrics – This module implements functions assessing prediction error for specific purposes.

sklearn.matrics.pairwise – This is a sub module in **sklearn.matrics** which implements utilities to evaluate pairwise distances of affinity of set of samples.

Cosine_similarity – This computes cosine similarity between samples in X and Y as follows;

- Cosine similarity(X,Y) = X*Y/||X||*||Y||

```
cosine_sim = cosine_similarity(tf_idf_matrix, tf_idf_matrix)

print(cosine_sim)
```

Output

```
[[1.         0.06787707 0.03973042 0.08261503 0.06299337 0.08570263
  0.06750272 0.28178071 0.         0.         0.18436755]
 [0.06787707 1.         0.49304659 0.11187574 0.06229914 0.1314421
  0.6613149  0.06118759 0.30461999 0.         0.        ]
 [0.03973042 0.49304659 1.         0.06200186 0.03164802 0.08750003
  0.42699656 0.02899932 0.30172464 0.         0.        ]
 [0.08261503 0.11187574 0.06200186 1.         0.49128319 0.53505921
  0.10786941 0.08390641 0.         0.18571007 0.        ]
 [0.06299337 0.06229914 0.03164802 0.49128319 1.         0.27939714
  0.07936087 0.06927846 0.         0.         0.        ]
 [0.08570263 0.1314421  0.08750003 0.53505921 0.27939714 1.
  0.13834258 0.08407119 0.         0.06696519 0.        ]
 [0.06750272 0.6613149  0.42699656 0.10786941 0.07936087 0.13834258
  1.         0.06105472 0.30722682 0.         0.        ]
 [0.28178071 0.06118759 0.02899932 0.08390641 0.06927846 0.08407119
  0.06105472 1.         0.         0.01444164 0.2322117 ]
 [0.         0.30461999 0.30172464 0.         0.         0.
  0.30722682 0.         1.         0.         0.        ]
 [0.         0.         0.         0.18571007 0.         0.06696519
  0.         0.01444164 0.         1.         0.        ]
 [0.18436755 0.         0.         0.         0.         0.
  0.         0.2322117  0.         0.         1.        ]]
```

pandas – It is a fast powerful, flexible and easy to use open source data analysis and manipulation tool which can be used for python programming. In here it is used to represent the cosine similarity matrix using pandas dataframe structure.

```
# transform the similarity matrix into a dataframe
df = pd.DataFrame(cosine_sim, columns=['doc_1', 'doc_2', 'doc_3', 'doc_4',
'doc_5', 'doc_6', 'doc_7', 'doc_8',
                                          'topic_1', 'topic_2', 'topic_3'],
                   index=['doc_1', 'doc_2', 'doc_3', 'doc_4', 'doc_5',
'doc_6', 'doc_7', 'doc_8',
                          'topic_1', 'topic_2', 'topic_3'])
```

Output

```
Cosine similarity matrix
            doc_1     doc_2     doc_3     doc_4     doc_5     doc_6     doc_7  \
doc_1    1.000000  0.067877  0.039730  0.082615  0.062993  0.085703  0.067503
doc_2    0.067877  1.000000  0.493047  0.111876  0.062299  0.131442  0.661315
doc_3    0.039730  0.493047  1.000000  0.062002  0.031648  0.087500  0.426997
doc_4    0.082615  0.111876  0.062002  1.000000  0.491283  0.535059  0.107869
doc_5    0.062993  0.062299  0.031648  0.491283  1.000000  0.279397  0.079361
doc_6    0.085703  0.131442  0.087500  0.535059  0.279397  1.000000  0.138343
doc_7    0.067503  0.661315  0.426997  0.107869  0.079361  0.138343  1.000000
doc_8    0.281781  0.061188  0.028999  0.083906  0.069278  0.084071  0.061055
topic_1  0.000000  0.304620  0.301725  0.000000  0.000000  0.000000  0.307227
topic_2  0.000000  0.000000  0.000000  0.185710  0.000000  0.066965  0.000000
topic_3  0.184368  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000

            doc_8   topic_1   topic_2   topic_3
doc_1    0.281781  0.000000  0.000000  0.184368
doc_2    0.061188  0.304620  0.000000  0.000000
doc_3    0.028999  0.301725  0.000000  0.000000
doc_4    0.083906  0.000000  0.185710  0.000000
doc_5    0.069278  0.000000  0.000000  0.000000
doc_6    0.084071  0.000000  0.066965  0.000000
doc_7    0.061055  0.307227  0.000000  0.000000
doc_8    1.000000  0.000000  0.014442  0.232212
topic_1  0.000000  1.000000  0.000000  0.000000
topic_2  0.014442  0.000000  1.000000  0.000000
topic_3  0.232212  0.000000  0.000000  1.000000
```

Then the pandas dataframe is used to order the documents and there topic.

```
# Identify documents related to the each topic
print('Identify documents related to each topic')
row = 0
for row in range(8):
    if df.iloc[row:row + 1, 8:11].max(axis=1)[0] == 0:
        check_most_sim_doc = df.iloc[row:row + 1, 0:8]
        most_sim_doc = check_most_sim_doc.T.apply(lambda x:
x.nlargest(2).idxmin())[0]

        while df.iloc[df.index == most_sim_doc, 8:11].max(axis=1)[0] <= 0:
            check_most_sim_doc = df.iloc[df.index == most_sim_doc, 0:8]
            most_sim_doc = check_most_sim_doc.T.apply(lambda x:
x.nlargest(2).idxmin())[0]

        print(df.columns[row], '-', df.iloc[df.index == most_sim_doc,
8:11].idxmax(axis=1)[0])

    else:
        print(df.columns[row], '-', df.iloc[row:row + 1,
8:11].idxmax(axis=1)[0])

# END
```

Output

```
Identify documents related to each topic
doc_1 - topic_3
doc_2 - topic_1
doc_3 - topic_1
doc_4 - topic_2
doc_5 - topic_2
doc_6 - topic_2
doc_7 - topic_1
doc_8 - topic_3
```

b. Brief explanation of the pre-processing steps you followed.

```
# convert all texts into lowercase and remove stop words and initiate the
TfidfVectorizer
vectorizer = TfidfVectorizer(lowercase=True, stop_words='english')
```

lowercase = True (default) **-** This converts all the terms into lower case in order to take the unique words only.

stop_words = 'english' (default) – All the terms are passed to check and the remove frequently used unimportant (do not help to take information) words from the corpus. There are pre-defined set of stop words.

norm = l2 (default) – The norm to use to normalize each non zero sample. They rescale the representation of each document to have Euclidean norm 1. Hence the length of the document does not change the vectorized representation. Therefore the results do not depend on the lengths of the documents.

Here the numerical values in the documents were not remove. Because they might be useful when comparing document similarity.

c. List of .txt documents related to each news topic.

Document similarities

Cosine similarity matrix

	doc_1	doc_2	doc_3	doc_4	doc_5	doc_6	doc_7	doc_8
doc_1	1.000000	0.067877	0.039730	0.082615	0.062993	0.085703	0.067503	0.281781
doc_2	0.067877	1.000000	0.493047	0.111876	0.062299	0.131442	0.661315	0.061188
doc_3	0.039730	0.493047	1.000000	0.062002	0.031648	0.087500	0.426997	0.028999
doc_4	0.082615	0.111876	0.062002	1.000000	0.491283	0.535059	0.107869	0.083906
doc_5	0.062993	0.062299	0.031648	0.491283	1.000000	0.279397	0.079361	0.069278
doc_6	0.085703	0.131442	0.087500	0.535059	0.279397	1.000000	0.138343	0.084071
doc_7	0.067503	0.661315	0.426997	0.107869	0.079361	0.138343	1.000000	0.061055
doc_8	0.281781	0.061188	0.028999	0.083906	0.069278	0.084071	0.061055	1.000000

According to the document similarity scores

- **Document 1, 8**
- **Document 2, 3 and 7**
- **Document 4, 5, and 6** are much similar to each other than others.

Document topics

	topic_1	topic_2	topic_3
doc_1	0.000000	0.000000	0.184368
doc_2	0.304620	0.000000	0.000000
doc_3	0.301725	0.000000	0.000000
doc_4	0.000000	0.185710	0.000000
doc_5	0.000000	0.000000	0.000000
doc_6	0.000000	0.066965	0.000000
doc_7	0.307227	0.000000	0.000000
doc_8	0.000000	0.014442	0.232212

According to the topic similarity,

- **Document 2, 3 and 7 belong to the topic_1**
- **Document 4, and 6 belong to the topic_2**
- **Document 1, 8 belong to the topic_3**

But document 5 does not belong to any topic. Because the words in the document 5 are not including any of the topics. Since document 5 is more similar to the document 4 with the similarity score of 0.491283, it can be concluded that the document 5 belongs to topic 2.

Therefore,

- Text documents related to the news topic**: *Hurricane Gilbert Heads Toward Dominican Coast*** are **doc 2.txt, doc 3.txt and doc 7.txt**
- Text documents related to the news topic: ***IRA terrorist attack*** are **doc 4.txt, doc 5.txt and doc 6.txt**
- Text documents related to the news topic: ***McDonald's Opens First Restaurant in China*** are **doc 1.txt, doc 8.txt**

```
# Identify documents related to the each topic
print('Identify documents related to each topic')
row = 0
for row in range(8):
    if df.iloc[row:row + 1, 8:11].max(axis=1)[0] == 0:
        check_most_sim_doc = df.iloc[row:row + 1, 0:8]
        most_sim_doc = check_most_sim_doc.T.apply(lambda x:
x.nlargest(2).idxmin())[0]

        while df.iloc[df.index == most_sim_doc, 8:11].max(axis=1)[0] <= 0:
            check_most_sim_doc = df.iloc[df.index == most_sim_doc, 0:8]
            most_sim_doc = check_most_sim_doc.T.apply(lambda x:
x.nlargest(2).idxmin())[0]

        print(df.columns[row], '-', df.iloc[df.index == most_sim_doc,
8:11].idxmax(axis=1)[0])

    else:
        print(df.columns[row], '-', df.iloc[row:row + 1,
8:11].idxmax(axis=1)[0])
```

Output

```
Identify documents related to each topic
doc_1 - topic_3
doc_2 - topic_1
doc_3 - topic_1
doc_4 - topic_2
doc_5 - topic_2
doc_6 - topic_2
doc_7 - topic_1
doc_8 - topic_3
```

d. Append your full code lines at the end of the PDF file.

```
"""
Name          : B.M.G.G.K. Rajapaksha
Faculty       : Faculty of Science, Bioinformatics
Index NO.     : S14210
Date          : 25/07/2022

Task          : Assignment 1; An algorithm to determine the document
frequency (tfidf classifier)
Input         : 1. A set of .txt documents (8) related to 3 different news
topics
                2. Topics (3) of the documents

Output        : 1. Cosine Similarity matrix
                2. Categorization of the documents according to their
similarity and topics

"""

# STEP 1 -Import libraries

from collections import OrderedDict
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import pandas as pd

# STEP 2 - Open documents

with open('doc 1.txt', 'r') as doc_1:
    doc_1 = doc_1.read()

with open('doc 2.txt', 'r') as doc_2:
    doc_2 = doc_2.read()

with open('doc 3.txt', 'r') as doc_3:
    doc_3 = doc_3.read()

with open('doc 4.txt', 'r') as doc_4:
    doc_4 = doc_4.read()

with open('doc 5.txt', 'r') as doc_5:
    doc_5 = doc_5.read()

with open('doc 6.txt', 'r') as doc_6:
    doc_6 = doc_6.read()

with open('doc 7.txt', 'r') as doc_7:
    doc_7 = doc_7.read()

with open('doc 8.txt', 'r') as doc_8:
    doc_8 = doc_8.read()

# News topics

topic_1 = 'Hurricane Gilbert Heads Toward Dominican Coast'
topic_2 = 'IRA terrorist attack'
topic_3 = 'McDonald\'s Opens First Restaurant in China'

# STEP 3 - take the corpus into a list
corpus = [doc_1, doc_2, doc_3, doc_4, doc_5, doc_6, doc_7, doc_8, topic_1,
```

```
topic_2, topic_3]

# STEP 4 - Fit the corpus in to the vectorizer with preprocessing
parameters

# convert all texts into lowercase and remove stop words and initiate the
TfidfVectorizer
vectorizer = TfidfVectorizer(lowercase=True, stop_words='english')

# fit the corpus into the vectorizer
tf_idf_matrix = vectorizer.fit_transform(corpus)
print('TF-IDF matrix : ', tf_idf_matrix)

# take the vocabulary into a sorted dictionary
sorted_dict = OrderedDict(sorted(vectorizer.vocabulary_.items(), key=lambda
x: x[1], reverse=False))
print('Sorted vocabulary\n', sorted_dict, '\n')
print('Length of the vocabulary - ', len(vectorizer.vocabulary_), '\n')

# STEP 5 - Cosine similarity matrix

cosine_sim = cosine_similarity(tf_idf_matrix, tf_idf_matrix)

print(cosine_sim)

# transform the similarity matrix into a dataframe
df = pd.DataFrame(cosine_sim, columns=['doc_1', 'doc_2', 'doc_3', 'doc_4',
'doc_5', 'doc_6', 'doc_7', 'doc_8',
                                       'topic_1', 'topic_2', 'topic_3'],
                  index=['doc_1', 'doc_2', 'doc_3', 'doc_4', 'doc_5',
'doc_6', 'doc_7', 'doc_8',
                         'topic_1', 'topic_2', 'topic_3'])

pd.set_option('display.max_columns', 11)
print('Cosine similarity matrix\n', df, '\n')

# Identify documents related to the each topic
print('Identify documents related to each topic')
row = 0
for row in range(8):
    if df.iloc[row:row + 1, 8:11].max(axis=1)[0] == 0:
        check_most_sim_doc = df.iloc[row:row + 1, 0:8]
        most_sim_doc = check_most_sim_doc.T.apply(lambda x:
x.nlargest(2).idxmin())[0]

        while df.iloc[df.index == most_sim_doc, 8:11].max(axis=1)[0] <= 0:
            check_most_sim_doc = df.iloc[df.index == most_sim_doc, 0:8]
            most_sim_doc = check_most_sim_doc.T.apply(lambda x:
x.nlargest(2).idxmin())[0]

        print(df.columns[row], '-', df.iloc[df.index == most_sim_doc,
8:11].idxmax(axis=1)[0])

    else:
        print(df.columns[row], '-', df.iloc[row:row + 1,
8:11].idxmax(axis=1)[0])

# END
```

Output

```
OrderedDict([('000', 0), ('10', 1), ('100', 2), ('11', 3), ('110', 4), ('115', 5), ('12', 6), ('125', 7), ('14', 8), ('140', 9), ('15', 10), ('16', 11), ('17', 12), ('18', 13),

Length of the vocabulary -  1260

Cosine similarity matrix
            doc_1     doc_2     doc_3     doc_4     doc_5     doc_6     doc_7  \
doc_1    1.000000  0.067877  0.039730  0.082615  0.062993  0.085703  0.067503
doc_2    0.067877  1.000000  0.493047  0.111876  0.062299  0.131442  0.661315
doc_3    0.039730  0.493047  1.000000  0.062002  0.031648  0.087500  0.426997
doc_4    0.082615  0.111876  0.062002  1.000000  0.491283  0.535059  0.107869
doc_5    0.062993  0.062299  0.031648  0.491283  1.000000  0.279397  0.079361
doc_6    0.085703  0.131442  0.087500  0.535059  0.279397  1.000000  0.138343
doc_7    0.067503  0.661315  0.426997  0.107869  0.079361  0.138343  1.000000
doc_8    0.281781  0.061188  0.028999  0.083906  0.069278  0.084071  0.061055
topic_1  0.000000  0.304620  0.301725  0.000000  0.000000  0.000000  0.307227
topic_2  0.000000  0.000000  0.000000  0.185710  0.000000  0.066965  0.000000
topic_3  0.184368  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000

            doc_8   topic_1   topic_2   topic_3
doc_1    0.281781  0.000000  0.000000  0.184368
doc_2    0.061188  0.304620  0.000000  0.000000
doc_3    0.028999  0.301725  0.000000  0.000000
doc_4    0.083906  0.000000  0.185710  0.000000
doc_5    0.069278  0.000000  0.000000  0.000000
doc_6    0.084071  0.000000  0.066965  0.000000
doc_7    0.061055  0.307227  0.000000  0.000000
doc_8    1.000000  0.000000  0.014442  0.232212
topic_1  0.000000  1.000000  0.000000  0.000000
topic_2  0.014442  0.000000  1.000000  0.000000
topic_3  0.232212  0.000000  0.000000  1.000000
```

```
Identify documents related to each topic
doc_1 - topic_3
doc_2 - topic_1
doc_3 - topic_1
doc_4 - topic_2
doc_5 - topic_2
doc_6 - topic_2
doc_7 - topic_1
doc_8 - topic_3
```