Information Retrieval and Text Mining: Homework #2

R08725008 資管碩二 周若涓

執行環境

- Jupyter Notebook

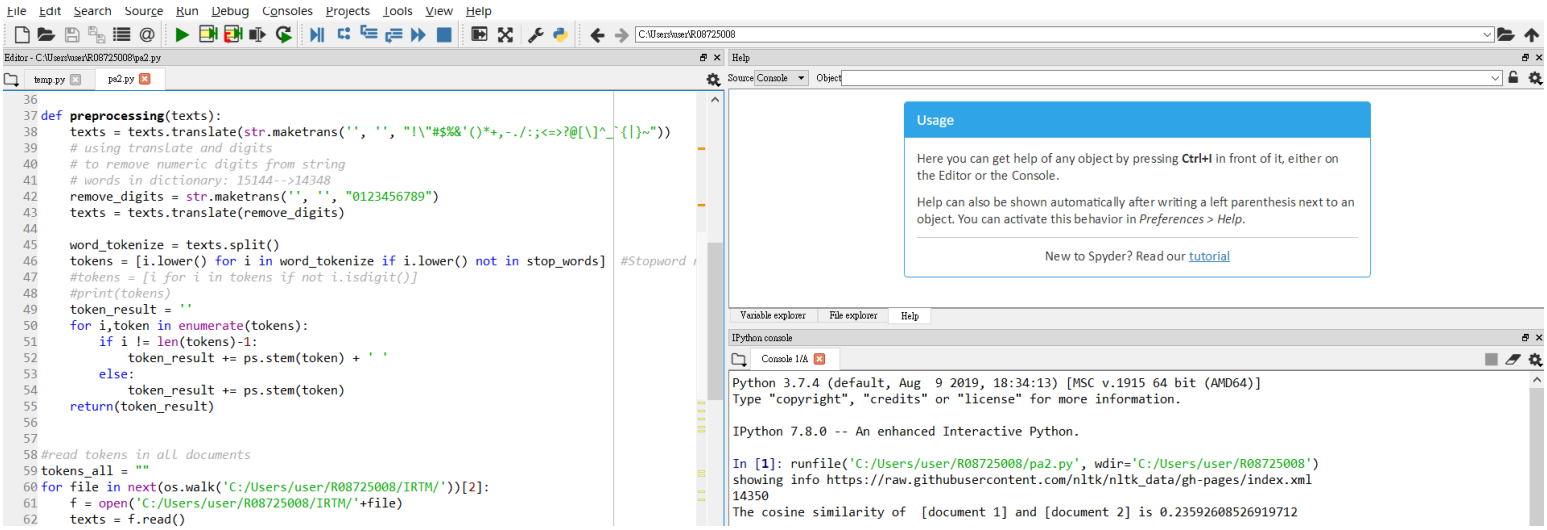
程式語言

- Python 3

執行方式

- 在執行前需安裝 nltk 套件 (command: pip install nltk)

- 安裝 pandas 套件 (command: pip install pandas)
- 安裝 numpy 套件 (command: pip install numpy)
- 以下說明 2 種執行環境:
  1. 可以利用 Spyder 開啟 pa2.py，並執行



2. 或可利用 python3 pa2.py 直接執行 python 檔案

```
(base) C:\Users\user\R08725008>python pa2.py
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml
14347
The cosine similarity of  [document 1] and [document 2] is 0.23592608526919712
```

- 確保提供的 IRTM 資料夾放於 C:/Users/user/R08725008/ 目錄下
- 產出的 dictionary.txt 預設放於 C:/Users/user/R08725008/ 目錄下
- 產出的 doc1.txt – doc1095.txt 預設放於 C:/Users/user/R08725008/tf_idf 目錄下

作業邏輯說明

Part 1: Construct a dictionary

1.

- 利用 nltk 套件初始化 PorterStemmer
- import stopwords 套件，並增加 list，其為觀察到可能為 stopwords

```
# install NLTK
import nltk
#import string
# install related NLTK packages
nltk.download()
# Porter's algorithm
from nltk.stem.porter import *
#stopwords package
from nltk.corpus import stopwords
# to read all files in folder
```

```
# Stemming using Porter's algorithm
ps = PorterStemmer()
# Stopword lists
stop_words = set(stopwords.words('english')) #Stopword
stop_words.update(['.', ',', '"', "'", '?', '!', ':', ';', '(', ')', '[', ']', '{', '}',
                   '\'s','\'m','\'re','\'ll','\'d','n\'t','shan\'t','that\'s',"'at",
         "_","``","\'\'","--","```",".","//",":","___",'_the','-',"'em",".com","...","\'ve",'u'])
# print(stop_words)
```

2. 定義 preprocessing function，有以下功能

- 去除 punctuation
- [更新] 與上次 preprocessing 不同在於，加上去除數字的處理
- Tokenize
- 將讀入的文件轉換為小寫，如果不在 stop words 當中的 word 才會保留
- 最後將每個 token 進行 stemming，並加回字串當中
- 把結果輸出 token_result

```
def preprocessing(texts):
    texts = texts.translate(str.maketrans('', '', "!\"#$%&'()*+,-./:;<=>?@[\]^_`{|}~"))
    # using translate and digits
    # to remove numeric digits from string
    # words in dictionary: 15144-->14348
    remove_digits = str.maketrans('', '', "0123456789")
    texts = texts.translate(remove_digits)

    word_tokenize = texts.split()
    tokens = [i.lower() for i in word_tokenize if i.lower() not in stop_words]  #Stopword removal
    #tokens = [i for i in tokens if not i.isdigit()]
    #print(tokens)
    token_result = ''
    for i,token in enumerate(tokens):
        if i != len(tokens)-1:
            token_result += ps.stem(token) + ' '
        else:
            token_result += ps.stem(token)
    return(token_result)
```

3. 讀入所有資料夾中的 documents，經過前處理後的所有 terms 收集至 tokens_all

```
#read tokens in all documents
tokens_all = ""
for file in next(os.walk('C:/Users/user/R08725008/IRTM/'))[2]:
    f = open('C:/Users/user/R08725008/IRTM/'+file)
    texts = f.read()
    f.close()
    tokens_all += preprocessing(texts)
```

4. 建立 dictionary list
   - 透過觀察，去除長度小於 3 的 terms，因為這類字通常無意義，例如: 單位(kg, km) 等
   - 將剩下的 terms 經過排序，形成 dictionary list

```
#print(tokens_all.split(' '))
token_set = set(tokens_all.split(' ')) #extract distinct words
token_list = list(token_set) #order word list
#print(token_list)
for token in token_list: #obervation: the term that less than three words is uaually meaningless e.g. and, or, km, kg...
    if len(token)<3:
        token_list.remove(token)
token_list = sorted(token_list)
print(len(token_list))
token_list
```

5.

- import pandas 做 dataframe 的運算
- token list(即上個步驟得到的 dictionary list)，放到 term 欄位
- t_index 從 1 開始，故將 index + 1
- 初始化 df 欄位(document frequency, 紀錄有幾個 document 出現這個 term)
- 並調整 dataframe 欄位順序，由左至右依序分別為 t_index、term、df

```
import pandas as pd
```

```
# use term list and  index to a DataFrame
df = pd.DataFrame(pd.Series(token_list),columns=['term'])
#index starts from one
df['t_index'] = df.index+1
#initialize the document frequency of each term
df['df'] = 0
#adjust the order of columns
df = df[['t_index','term','df']]
df
```

	t_index	term	df
0	1	aan	0
1	2	aaron	0
2	3	aback	0

6.

- 依序讀入 1.txtx 至 1095.txt
- 比對 token list(dictionary)，如果 token list 中的 term 有出現在 document 中，df 值就加 1

```
for file in tqdm(next(os.walk('C:/Users/user/R08725008/IRTM/'))[2]):
    f = open('C:/Users/user/R08725008/IRTM/'+file)
    texts = f.read()
    f.close()
    tokens_doc = preprocessing(texts)
    # Record the document frequency of each term
    for term in token_list:
        if term in tokens_doc:
            df.loc[df['term']==term,'df'] += 1
df
```

```
100%|
```

	t_index	term	df
0	1	aan	2

7. Optimization
   - 去除只有出現在 3 個 documents 以下的 terms
   - 去除 common terms ，即出現在超過 90% 的 documents 的 terms
     - 1000*0.9 約等於 985
   - t_index 從 1 開始，故將 index + 1

```
#optimization: refined the dictionary by filtering out unimportant words
#remove the low frequency words
df_ = df.drop(df[df.df<3].index)
# remove common words which occurs in 90% of  documents
df_ = df_.drop(df_[df_.df>985].index)
df_.reset_index(drop=True,inplace=True)
df_['t_index'] = df_.index + 1
#Construct the final dictionary
df_
```

	t_index	term	df
0	1	aback	7
1	2	abandon	39
2	3	abc	51

8. 輸出 dictionary.txt

```
#Save the dictionary as a txt file
df_.to_csv('C:/Users/user/R08725008/dictionary.txt',index=False,header=False,sep=' ')
```

Part 2: tf-idf unit vector

1.
   - 建立 tf_idf dataframe，並初始化 tf 值
   - 計算 term frequency 值，term frequency 值為 document level，讀取每一個 document，計算 token_list 中的每一個 term 出現的次數(頻率)

```
tf_list = list(df_.term)
for file in tqdm(next(os.walk('C:/Users/user/R08725008/IRTM/'))[2]):
    tf_idf = df_[['t_index','term']]
    #Initialize term frequency in tf column
    tf_idf['tf'] = 0
    f = open('C:/Users/user/R08725008/IRTM/'+file)
    texts = f.read()
    f.close()
    #Extract tokens of each document
    tokens_all = preprocessing(texts)
    #Calculate tf value:the number of occurrences of the term in the document
    num_of_terms = 0
    for token in tokens_all.split(' '):
        if token in tf_list:
            tf_idf.loc[tf_idf['term']==token,'tf'] += 1
            num_of_terms += 1
```

2.

- import numpy 做 log 計算
- 計算 idf 值，log10(1095/df)
- tf 即為上個步驟算出的 tf_idf.tf 再除 num_of_terms
- tf-idf 即為 tf 乘 idf
- 去除 tf_idf.tf =0 的列: document 中有一些 tokens 不在 token_list 中，所以其 tf_idf.tf 為 0，tf-idf 值也為 0

```
import numpy as np
```

```
#Calculate the inverse document frquency
idf = np.log10(1095 / df_.df)
#Calculate the tf-idf unit vector
tf_idf.tf = (tf_idf.tf / num_of_terms)* idf
#Remove the row whcih tf_idf.tf is zero(words not in dictionary)
tf_idf = tf_idf[tf_idf.tf > 0]
```

3.

- 將所有 tf-idf 相加，再把每一個 tf-idf 除 tf-idf 的總和，做 normailize，形成 tf-idf unit vector
- 將 column 重新命名: tf 改成 tf-idf，以符合輸出格式
- 去除 term 欄位，留下 t_index 與 tf-idf 欄位即可
- 輸出時，將此 document 中在 token_list 中的 terms 的數量記於第一列

```
#normalize to unit vector
tf_idf_sum = np.sum(tf_idf.tf)
tf_idf.tf = (tf_idf.tf)/tf_idf_sum
tf_idf = tf_idf.rename(columns={'tf': 'tf-idf'})
#Remove term column: we only need t_index & tf_idf
tf_idf.drop('term',axis=1,inplace=True)
tf_idf.to_csv('C:/Users/user/R08725008/tf_idf/'+file, index=False, header=False, sep=' ')
with open('C:/Users/user/R08725008/tf_idf/'+file,'r') as fo: unit_vector = fo.read()
with open('C:/Users/user/R08725008/tf_idf/'+file, 'w') as result: result.write(str(len(tf_idf))+'\n'+unit_vector)
```

Part 3: Cosine similarity

1. 定義 similarity function 計算 cosine similarity
    - 讀取 document 1 以及 document 2
    - 去除第一列 (此 document 中在 token_list 中的 terms 的數量)
    - cosine similarity 分子為向量相乘，使用 numpy 中的 dot 運算
    - cosine similarity 分母為向量長度相乘，使用 numpy 中的 norm 運算來計算向量的長度
    - 上述兩點相除，即得到 cosine similarity，此為 function 的 return value

```
from numpy import dot
from numpy.linalg import norm
```

```
def similarity(doc1, doc2):
    #separate the column with space
    d1 = pd.read_csv(doc1,names=['t_index','tf_idf'], sep=' ')
    d2 = pd.read_csv(doc2,names=['t_index','tf_idf'], sep=' ')
    #remove the first row(counter)
    d1 = d1.drop(0)
    d2 = d2.drop(0)
    #d2
    d1_d2 = pd.merge(d1,d2,on='t_index', how='outer')
    d1_d2.fillna(0,inplace=True)
    matrix_product = np.sum(dot(d1_d2.tf_idf_x, d1_d2.tf_idf_y))
    #print(matrix_product)
    #print(norm(d1_d2.tf_idf_x)*norm(d1_d2.tf_idf_y))
    sim = matrix_product / (norm(d1_d2.tf_idf_x)*norm(d1_d2.tf_idf_y))
    return sim
```

2. document 1 與 document 2 的 cosine similarity 為 **<u>0.2359260852691971</u>**

```
sim = similarity('C:/Users/user/R08725008/tf_idf/1.txt','C:/Users/user/R08725008/tf_idf/2.txt')
print("The cosine similarity of  [document 1] and [document 2] is "+str(sim))
```

```
The cosine similarity of  [document 1] and [document 2] is 0.2359260852691971
```