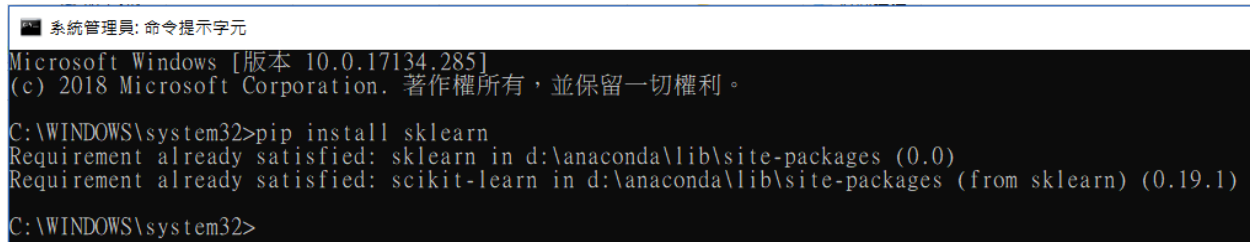


作業二 李妮燁

1. 執行環境 & 作業系統

本次採用的作業系統為 win 10 64 位元，執行環境為 Visual Studio Code，需要預先安裝 Natural Language Toolkit tool、sklearn tool。首先要先在電腦用 cmd 安裝各個 tool 的工具包，用以下指令：

```
pip install -U sklearn
```



```
系統管理員: 命令提示字元
Microsoft Windows [版本 10.0.17134.285]
(c) 2018 Microsoft Corporation. 著作權所有，並保留一切權利。
C:\WINDOWS\system32>pip install sklearn
Requirement already satisfied: sklearn in d:\anaconda\lib\site-packages (0.0)
Requirement already satisfied: scikit-learn in d:\anaconda\lib\site-packages (from sklearn) (0.19.1)
C:\WINDOWS\system32>
```

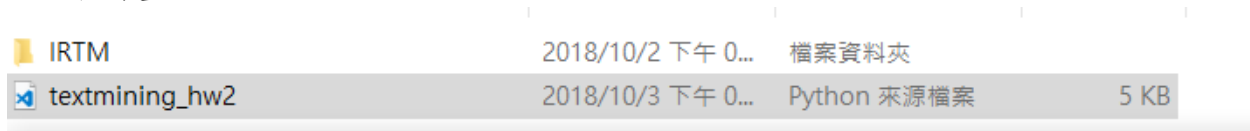
2. 程式語言, 版本

使用的程式語言和版本為 python 3.7。

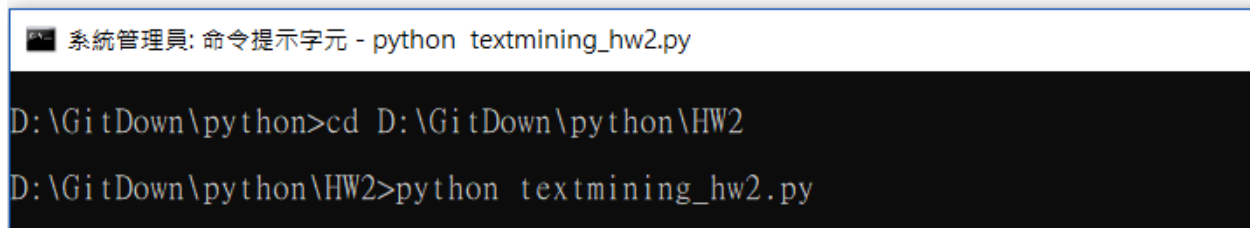
3. 執行方式

先將要處理的 1095 個文字檔案存到 IRTM 的資料夾，並且和 python source code 存放在同一個資料夾，使用 Visual Studio Code 編寫完後，cmd 執行 python 程式。

● 執行畫面



IRTM	2018/10/2 下午 0...	檔案資料夾
textmining_hw2	2018/10/3 下午 0...	Python 來源檔案 5 KB



```
系統管理員: 命令提示字元 - python textmining_hw2.py
D:\GitDown\python>cd D:\GitDown\python\HW2
D:\GitDown\python\HW2>python textmining_hw2.py
```

4. 作業處理邏輯說明

首先要 import 多項套件，例如：

```
import math
import string
from nltk.stem import PorterStemmer
from nltk.tokenize import WordPunctTokenizer
from nltk.tokenize import word_tokenize
from nltk.tokenize import wordpunct_tokenize
from nltk.corpus import stopwords
```

● 統計各個 term 出現在幾個文件

先將各文件的 text 處理成 term 後，**set(term)**讓重複的 term 去掉，使用 **union** 聯集所有文件的 term，最後 **sort** 過後存成一個 term 的 list；**df_dict={}**為一個字典，在每個文件轉成

作業二 李妮燁

term 後不重複的 term 只要字典還沒有就加入字典，有的話在 value 加一，df_dict={} 就可以統計出每個 term 的 df 數值。

```
doc_non_repeat=list(set(words)) #每個文件不重複的 term 的 list
for i in range(len(doc_non_repeat)):
    if doc_non_repeat[i] not in df_dict:
        df_dict[doc_non_repeat[i]]=1
    else:
        df_dict[doc_non_repeat[i]]+=1
```

最後把 print 結果寫入到 **dictionary.txt** 中，同時建立一個查詢編號的字典。

```
dict={} #紀錄 1095 個文件出現過的 term 的 t_index 編號
fp = open("dictionary.txt", "a+", encoding='utf-8')
for i in range(len(union)):
    #df_dict[word] 的 value 是 term 的 df 值
    print(i+1,union[i],"df",df_dict[union[i]],file=fp)
    dict[union[i]]=str(i+1) #取得這個 term 的編號
```

- 產生各別文件裡 term 的紀錄和顯示 tfidf unit vector 的數值

```
dict_idf={} #term 的 idf 值存成 dict
for word in dict:
    dict_idf[word]=math.log10(1095/df_dict[word])
```

先將每個 term 的 idf 計算出來

```
doc_num_tf_list=[] #存放每個文件裡面的 term 字典(字典的 value 為 term 和出現的個數)
#被用來 sort
for i in range(len(doc_num_tf[i])): #doc_num_tf[i] 為一個 dict
    doc_num_tf_list.append(list(doc_num_tf[i]))
    doc_num_tf_list[i].sort()
```

doc_num_tf 為一個陣列，裡面一個 element 就存放一個字典，字典紀錄每個文件所有出現過的 term 他們個別的出現次數。

```
#doc_num_tf[i] 為一個 dict
for word in doc_num_tf[i]:
    #取出 dict 的 term 的 tf 數值
    tf_idf[word]=doc_num_tf[i][word]*(dict_idf[word])
    count=count+((tf_idf[word])**2)
count=math.sqrt(count)
for word in tf_idf:
    tf_idf[word]=tf_idf[word]/count #normalize 後的數值
```

計算 tf-idf 的 **unit vector 數值**，先將 term sort 過後存成 list 整理後生成文件的規格，將每個文件以迴圈的方式寫進文件檔案

```
tf_idf_list.append(word) #sort 過後的 term
tf_idf_list.sort()
for i in range(len(tf_idf_list)):
    #dict[tf_idf_list[i]] 為 term 的編號
    #tf_idf_list[i] 為排序過後的 term
    #tf_idf[tf_idf_list[i]] 為此 term 的 if idf 值
```

作業二 李妮燁

```
print  
("t_index",dict[tf_idf_list[i]],tf_idf_list[i],tf_idf[tf_idf_list[i]],file=fp)
```

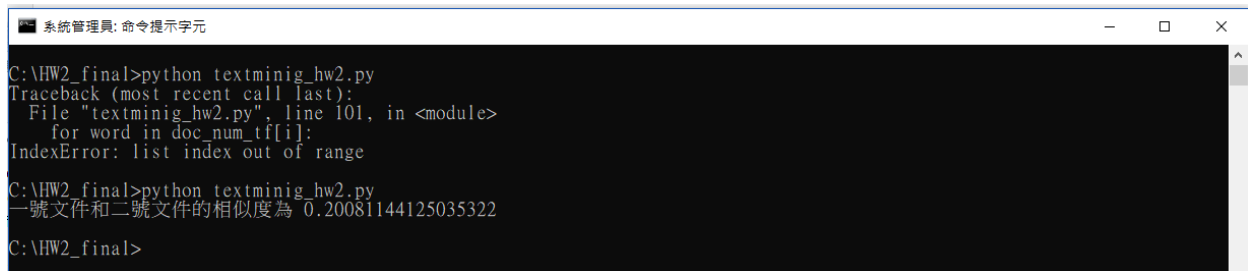
- 計算兩個文件之間的 cosine similarity

```
if i==0:  
    tf_idf_0[word]=tf_idf[word]  
if i==1:  
    tf_idf_1[word]=tf_idf[word]
```

在前面產生文件檔時事先記錄了文件一號跟二號的 tf-idf unit vector 的值和他們各自擁有的 term

```
cosine_sim=0  
for word in tf_idf_0: #第一號文件出現的 Term  
    if word in tf_idf_1: #如果第二號文件也有同一個 term 兩者的 tf-idf-unit vector  
        乘起來相加  
        cosine_sim=cosine_sim+(tf_idf_0[word]*tf_idf_1[word])  
print("一號文件和二號文件的相似度為",cosine_sim)
```

如果在文件一號出現的 term 也在二號出現，就將他們內積，計算出 cosine similarity 得到的相似度為 0.200811



```
系統管理員: 命令提示字元  
C:\HW2_final>python textminig_hw2.py  
Traceback (most recent call last):  
  File "textminig_hw2.py", line 101, in <module>  
    for word in doc_num_tf[i]:  
IndexError: list index out of range  
C:\HW2_final>python textminig_hw2.py  
一號文件和二號文件的相似度為 0.20081144125035322  
C:\HW2_final>
```

補充:也用 cosine_similarity 的 function 測試計算過，相似度同上數值。

```
def cosine_similarity(v1,v2):  
    #"compute cosine similarity of v1 to v2: (v1 dot v2)/{||v1||*||v2||}"  
    sumxx, sumxy, sumyy = 0, 0, 0  
    for i in range(len(v1)):  
        x = v1[i]; y = v2[i]  
        sumxx += x*x  
        sumyy += y*y  
        sumxy += x*y  
    return sumxy/math.sqrt(sumxx*sumyy)
```

5. 任何在此作業中的心得

本次作業大量運用到 tf、idf 相關的計算，運用相關套件之前需要花許多時間查找資料，也要將上次作業一 extract 出來的 term 經過整理、改變型態等，也建立了很多相關的字典型態，最後的數值結果也需要經過一番整理才能創立檔案做相關的紀錄。其中碰到一些問題例如：字典塞進 list 存取、公式的計算、log 基底更改等等都花了一定時間閱讀，最後才能順利得到資料，此次作業讓我更了解文件之間相似性分析步驟與原理。