作業四 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 。

2. 程式語言, 版本

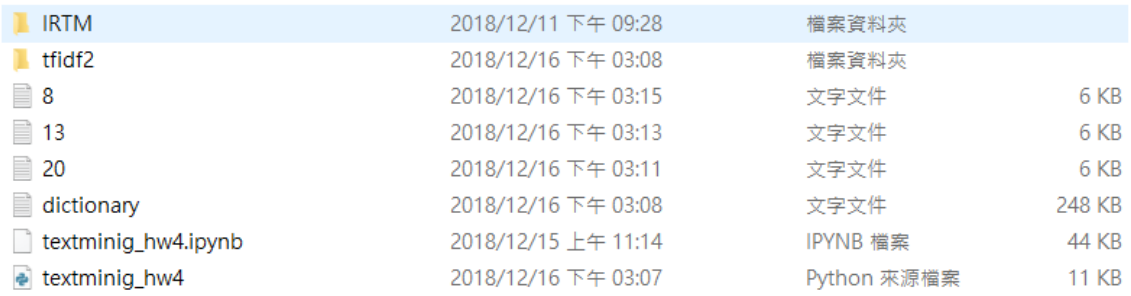
使用的程式語言和版本為 python 3.7。

3. 執行方式

先將要處理的 1095 個文字檔案存到 IRTM 的資料夾，一個空的"tfidf2"的資料夾-tfidf2.txt 與 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

- 執行結果



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
import nltk
```

運用作業二算出的 cosine similarity 公式改寫成 function，return 回兩個文件的相似度

```
def cosine_sim_num(doc1,doc2):
    cosine_sim=0
    for word in tf_idf_all[doc1-1]:  #第一號文件出現的 Term
```

作業四 李妮燁

```
        if word in tf_idf_all[doc2-1]:  #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_all[doc1-
1][word]*tf_idf_all[doc2-1][word])
    return cosine_sim
```

A. 將 1095 號文件對應的 similarity 存入 sim_sum 二維陣列中，本身對於本身的相似度設為 0，建立一個字典存放 1095 個 list，每個 list 為 merge 的情況

```
for i in range(1,1096):
        sim_record=[]
        for j in range(1,1096):
            sim_record.append(cosine_sim_num(i,j))
        sim_sum.append(sim_record)
for i in range(0,1095):
        sim_sum[i][i]=0    #自己和自己相似度設為 0
    for i in range(1095):
        cluster_dict[i]=[]
```

B. 從 1095 個文件內找最大的相似度，max_sim_now[0]為相似度數值，max_sim_now[1]和 max_sim_now[2]為兩個文件編號各減一(因為存放在 list 從 0 開始)

```
def max_sim_info():
    max_sim_now=[]
    max_sim_now.append(max(find_max_sim))#從 1095 個文件內找最大的相似度
    max_sim_now.append(find_max_sim.index(max(find_max_sim)))

max_sim_now.append(sim_dict_doc_num[find_max_sim.index(max(find_max_sim))])
    # max(find_max_sim)  #sim 最高的數值
    return max_sim_now
```

C. 把 2 個文件編號較大的 merge 到較小的編號，進行兩個文件對其他文件的相似度比較，取較小的(較遠，這邊採用 **complete link 的方式**)，update 文件 merge 後的相似度，將相似度較大的(距離較近)和其他文件相似度設定為 0(不會再比較到)，全部更新完後，把文件較大的編號從字典裏面刪除。

```
def update_act():

cluster_dict[min(max_sim_now[1],max_sim_now[2])].append(max(max_sim_now[
1],max_sim_now[2]))
    if cluster_dict[max(max_sim_now[1],max_sim_now[2])]!=[]:  #後 merge
的 cluster 為非一個文件的 cluster(裡面還有先前 merge 的文件)
        for x in
range(len(cluster_dict[max(max_sim_now[1],max_sim_now[2])])):

cluster_dict[min(max_sim_now[1],max_sim_now[2])].append(cluster_dict[max
(max_sim_now[1],max_sim_now[2])][x])
```

作業四 李妮燁

```
    #將 cluster 用字典的方式儲存，以編號小的為 key，value 就是目前 cluster 裡面
的文件編號
    #這裡"還沒有包含本身"，cluster 的編號就是以他為群體
    for i in range(0,1095):
        if sim_sum[max_sim_now[1]][int(i)] >
sim_sum[max_sim_now[2]][int(i)]:
#
sim_sum[max_sim_now[2]][int(i)]=cosine_sim_num(max_sim_now[2]+1,int(i)+1
)

sim_sum[max_sim_now[1]][int(i)]=sim_sum[max_sim_now[2]][int(i)]
            sim_sum[i][1]=sim_sum[max_sim_now[2]][int(i)]
                    #相似度大代表比較近，cluter 裡面的點分別跟外面的點的相似
度取較小的
        else:

sim_sum[max_sim_now[2]][int(i)]=sim_sum[max_sim_now[1]][int(i)]
            sim_sum[i][2]=sim_sum[max_sim_now[1]][int(i)]
        sim_sum[max(max_sim_now[1],max_sim_now[2])][i]=0
        sim_sum[i][max(max_sim_now[1],max_sim_now[2])]=0
    del cluster_dict[max(max_sim_now[1],max_sim_now[2])]
```

D. 依照剛剛上面的步驟，如果要產出 20 個分群，需要執行 1075 次(一次會消掉一個 cluster，一開始為 1095 個)

```
sim_sum=[]
cluster_dict={}
remerge()
for x in range(1075):
    sim_dict_doc_num={}
    for i in range(1095):
        sim_dict_doc_num[i]=sim_sum[i].index(max(sim_sum[i]))
        #字典 key 為 0， value 就是對應最大相似度的另一個文件編號
    find_max_sim=[]
    for i in range(1095):
        find_max_sim.append(sim_sum[i][sim_dict_doc_num[i]])
    #從第一號文件找出每號文件對應最大的相似度存為陣列
    max_sim_now=max_sim_info()
    update_act()
```

E. 因為儲存在字典內的編號需加一才為真正的文件編號，所以在這邊進行每一個陣列的元素加一，然後 sort 排序

```
for x in cluster_dict:
    cluster_dict[x]=[i+1 for i in cluster_dict[x]]  #文件編號為 sim_sum
裡面的編號+1，例如 sim_sum100 為文件 101 號
```

```
#     cluster_dict[x].append(x+1)  #加入開頭的最小的文件編號
    cluster_dict[x].sort()
```

F. 將分群結果印出

```
fp = open('20.txt', "a+", encoding='utf-8')
for x in cluster_dict:
    print(x+1,file=fp)
    for i in range(len(cluster_dict[x])): #cluster_dict[x] 是一個 list
        print(cluster_dict[x][i],file=fp)
    print('',file=fp)
fp.close()
```

重複步驟 A 到 F 來取得 13 群和 8 群(更改 merge cluster 的次數)

5. 任何在此作業中的心得

這次作業中我選擇 complete link 的方法去分 cluster，過程中需要釐清一些觀念，例如在寫 merge 的地方原本不太清楚 merge 後需要存放到哪個文件編號、或是針對兩個 cluster 各不只有一個文件時的狀況需要做的處理等等，整體來說此次的作業讓我更理解分群的的過程與步驟。