

1. 執行環境

- Jupyter Notebook

2. 程式語言&版本

- 語言：Python
- 版本：3.6.6

3. 執行方式

打開 cmd，cd 到檔案位置

```
C:\Users\hp>cd C:\Users\hp\Desktop\IR\assignment\HW4
```

再輸入以下指令，即可執行並顯示 output：

```
C:\Users\hp\Desktop\IR\assignment\HW4>python3 r07725044_HW4.py
```

藍色部分填入 python.exe 及其路徑，或是如上圖輸入在 python 開啟的指令，python3 是我在環境變數下設定 python 3.6.6 版本的指令

（在環境變數先加入 python.exe 的 path，再更改檔名為 python3.exe，即可直接啟動）

紅色填入檔名 HW4_r07725044.py

得到 output：

在當前目錄下會產出名為 8.txt、13.txt、20.txt 的檔案，是分群後的結果

此外還會產生一個名為 matrix.dat 的檔案，是記錄文件之間相似度的 matrix

4. 作業處理邏輯說明

1. 前處理

- Import 套件

將所有要用的套件 import ，用途如註解

```
#用以定義dictionary的資料結構
from collections import defaultdict

#同作業一，斷出term的
import nltk
import string
# from collections import Counter
from nltk.stem.porter import PorterStemmer #import porter algorithm的套件
from nltk.corpus import stopwords

#打開文檔並讀取
import glob
import re
import os
import operator
import sys

#random select
import random

#計算用的sqrt () log ()
import math

#csv寫入
import csv

#矩陣用
import numpy as np

#計算程式運行時間
import time
```

- 加入前幾次作業的 function

將 document 轉換成 term 的 function

```
def term(word):
    res = nltk.word_tokenize(word)
    porter = PorterStemmer() #定義方法
    stemmer = [porter.stem(element) for element in res] #stemming
    stop = set(stopwords.words('english'))
    final = []
    for s in stemmer:
        if s not in stop:
            if s not in final:
                final.append(s)
    return final
```

計算 term frequency 的 function :

```
def TF(string):
    tf = defaultdict(int) #建立Dictionary的資料結構，以term作為key，頻率做value，e.g. 'word' : 3
    #hw1的方式產出term
    res = nltk.word_tokenize(string)
    porter = PorterStemmer()
    stemmer = [porter.stem(element) for element in res] #stemming
    stop = set(stopwords.words('english'))
    final = []
    for s in stemmer:
        if s not in stop:
            if s.isalpha(): #判斷是否為英文字母
                final.append(s)
    for t in final: #將斷出來的字統計為term的次數
        tf[t] += 1

    return tf
```

計算各文件之間的相似度：

```
def cosine(d1, d2): #傳入數值，為document id，這次作業以 1,2為計算對象
    path = 'tfidf' #讀取 d1,d2 兩份文件的位置
    #將document id 轉為對應的檔名
    xfile = str(d1) + '.txt'
    yfile = str(d2) + '.txt'

    #定義存取兩個文件的 unit vector
    x = {}
    y = {}

    c = 1
    #打開第一份文件讀取
    fix = os.path.join(path, xfile)
    with open(fix, 'r') as fx:
        for line in fx: #由於在儲存時，將 first line作為該column的title，因此需從 second line讀
            if c <= 2:
                c += 1
                continue
            (key, val) = line.split() #讀取 term index 和 normalize過後的 tfidf
            x[key] = val

    #第二份文件的處理，同第一份文件
    cou = 1
    fiy = os.path.join(path, yfile)
    with open(fiy, 'r') as fy:
        for line in fy:
            if cou <= 2:
                cou += 1
                continue
            (key, val) = line.split()
            y[key] = val

    summ = 0.0 #用以累加個內積的值
    #以第一份文件找第二份文件有無對應到的term

    for s in x:
        if s not in y: #若 x存在一 term是 y沒有的，則將這個 term設給 y，值為 0，方便之後做內積
            y[s] = 0
        summ = summ + float(x[s])*float(y[s]) #unit vector內積，極為 2 documents的相似度

    return summ
```

- 建立各文件之間相似度的 matrix

運用 cosine similarity 的方法，建立文件之間相似度關係的 matrix，值為 0~1，若是文件對自己本身的關係，則設為負無限大，如此一來不會在 argmax 的時候取到

```
start_time = time.time()

matrix = np.zeros((1095, 1095))
I = []
#建立矩陣
for n in range(0, 1095):
    for i in range(0, 1095):
        if(n == i): # n 跟 i 相等，也就是自己與自己的相似度一定會很大，因此設成，日後不會被挑到
            matrix[n][i] = float('-inf')
        else:
            x = n + 1
            y = i + 1
            matrix[n][i] = cosine(x, y) #index從0開始，要記得加一
    I.append(1) #存有哪些
print(matrix)

print("time of building matrix: ")
print("--- %s seconds ---" % (time.time() - start_time))

[[ -inf 0.20113796 0.30104544 ... 0.03777052 0.02942304 0.0497299 ]
 [0.20113796 -inf 0.20126398 ... 0.02986856 0.01167086 0.02317333]
 [0.30104544 0.20126398 -inf ... 0.04507046 0.02447444 0.04011235]
 ...
 [0.03777052 0.02986856 0.04507046 ... -inf 0.13256605 0.02298185]
 [0.02942304 0.01167086 0.02447444 ... 0.13256605 -inf 0.01967375]
 [0.0497299 0.02317333 0.04011235 ... 0.02298185 0.01967375 -inf]]
--- 1593.7882721424103 seconds ---
```

```
matrix.dump("matrix.dat") #存入本地端
```

2. Clustering

- *Hierachical Clustering + Complete Link*

Argmax 的 function：

2 個 cluster 必須都是 available 的，才能被取值

```
#取最大值
def arg(mtx, I):
    maxima = float('-inf')

    for x in range(0, 1095):
        for y in range(0, 1095):
            if (I[x] + I[y]) == 2 and x != y and mtx[x][y] > maxima:
                maxima = mtx[x][y]
                re = [x, y]

    return re[0], re[1]
```

Clustering

```

def HAC_CompleteLink(k):
    I = [1] * 1095
    A = [] #結果 List
    K = 1095 - k

    #加載建立好的 matrix
    matrix = np.load("matrix.dat")

    #clustering
    for k in range(0, K):

        #取最大值的 2個 cluster
        x, y = arg(matrix, I)
        sets = set([x, y]) #存成 set

        flag = True #看 結果List 有沒有可以合併的，合併成功改成 false

        concat = [] # 合併成功的結果
        delete = [] # 原本參與合併的要刪掉
        for item in A:
            if len(item.intersection(sets)) > 0: #有交集的item記錄下來
                temp = item.union(sets)
                concat.append(temp)
                delete.append(item)
                flag = False

        if flag: #沒有合併成功，加入新的 pair
            A.append(sets)
        else: #合併成功就加入合併結果，并把原本參與合併的item刪除
            for dlt in delete:
                A.remove(dlt)
            add = set()
            for con in concat:
                add = add.union(con)
            A.append(add)

        #更新cluster之間的距離
        for j in range(0, 1095):
            matrix[x][j] = min(matrix[x][j], matrix[y][j])
            matrix[j][x] = min(matrix[j][x], matrix[j][y])

        #標記為已被合併
        I[y] = 0

    return A

```

取出最大值的兩個對應的 cluster，把編號較大的合併進小的，且用 set 的資料結構，將包含 2 個 cluster 其中的任意一個的 cluster 全部合併，最後更新 cluster 之間的相似度關係，使用 complete link 所以取最小相似度，再更新被合併的 cluster 為 not available

- Call function and save result

儲存至 file 的 function：

因為前面的資料結構是以 0 為開始的，所以所有編號要加 1 才是正確的文件編號，每個 cluster 中間隔 1 行


```
#分群結果存入 file
def save(clusters):
    filename = str(len(clusters)) + '.txt'
    file = open(filename, "wt")
    for c in clusters:
        for doc in sorted(c):
            file.write(str(doc + 1) + '\n')
        file.write('\n')
    file.close()
```




呼叫 function，分別分出 8 群、13 群、20 群：

```
cluster8 = HAC_CompleteLink(8)
save(cluster8)

cluster13 = HAC_CompleteLink(13)
save(cluster13)

cluster20 = HAC_CompleteLink(20)
save(cluster20)
```

最後當前目錄會產生出 3 個以群數為命名的檔案，就是分群結果

-  8.txt
-  13.txt
-  20.txt