R10725011 陳佑甄

作業四

1. 執行環境

本機開 Anaconda，建立虛擬環境，下 python 去跑.py 檔。

用 Visual Studio Code 撰寫 python 程式碼。

2. 程式語言 (請標明版本)

Python 3.7.11

3. 執行方式

本機開啟 Anaconda Powershell Prompt，Anaconda 版本是 conda 4.10.3，

使用 nltk 套件。

- 創建虛擬環境：**(以下我有先創虛擬環境，但也可以不創建)**

```
$conda create --name IR python=3.7
```

下`$conda env list`可以看到剛剛創建的虛擬環境。

```
(base) PS C:\Users\[redacted]> conda env list
# conda environments:
#
base                  *  D:\anaconda
AIHW1                    D:\anaconda\envs\AIHW1
IR                       D:\anaconda\envs\IR
pyhon3.7                 D:\anaconda\envs\pyhon3.7
```

用`$ conda activate IR`可以啟動剛剛創好的虛擬環境。

- 安裝 nltk 套件：**(若沒有 nltk 套件一定需要 install)**

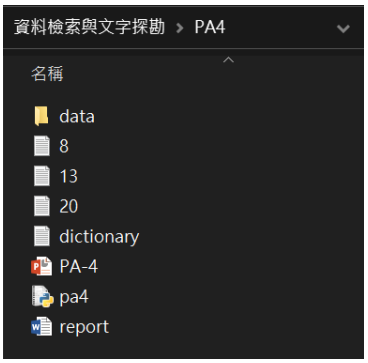
然後由於有用到 nltk 套件抓 stopword list ，

所以要下`$conda install -c anaconda nltk`。

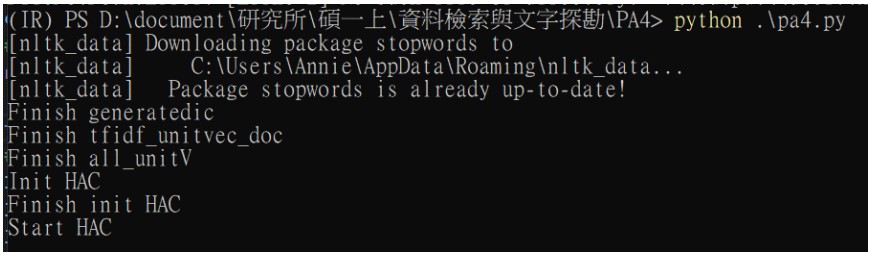
- 執行程式：

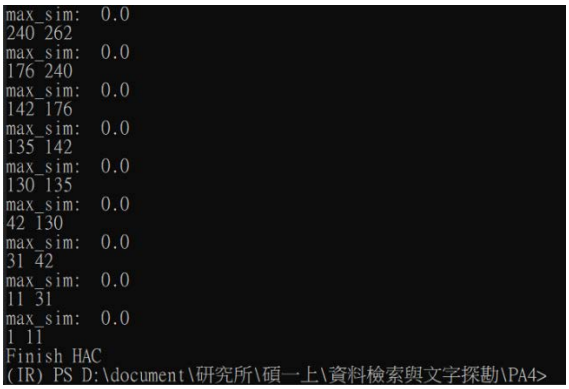
去到放 pa4.py 的資料夾，`$python .\pa4.py` 就可以執行程式碼。

由於會用到先前的資料，放在與 py 檔同一層資料夾中的 data 資料夾。

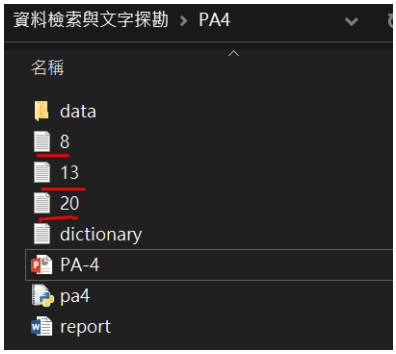


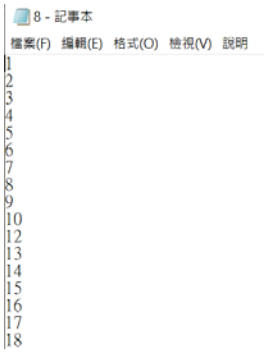
執行結果如下圖，會印出執行到哪一階段，分別是 Finish generatedic、Finish tfidf_unitvec_doc、Finish all_unitV、Init HAC、Finish init HAC、Start HAC、一連串的 max_sim 跟當下合併的兩個 cluster ID，最後印出 Finish HAC 就全部結束，全部跑完大概花 5 分半。

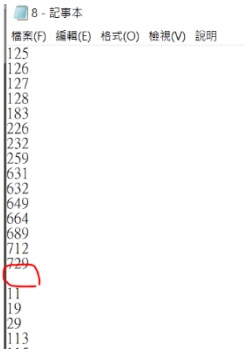




印出 Finish HAC 代表程式執行完成，過程中會在與 py 檔的同一層資料夾產生出 8.txt、13.txt、20.txt（dictionary 是中間產物）。點開 txt 檔部分輸出如圖所示，Cluster 間會用空格區分。







4. 作業處理邏輯說明

- import

```
import nltk
import glob
import os
import math
import numpy as np
from nltk.stem import PorterStemmer
nltk.download('stopwords')
```

- PA2 範圍

```
  # Generate Dictionary
  d, TFtds = generatedic()
print("Finish generatedic")
  # Transfer each document into a tf-idf unit vector
  ALL_doc_unit = tfidf_unitvec_doc(d, TFtds)
  # ALL_doc_unit = {docID: [[index], [doc unit]]}
  print("Finish tfidf_unitvec_doc")
  # get all unitV
  ALL_unitV = all_unitV(ALL_doc_unit)
  # ALL_unitV = {docID: [unitV]}
  print("Finish all_unitV")
```

以上都屬於 pa2 的範圍，從讀檔一直到產生出 ALL_unitV 的所有 doc 的補過 0 的 vocabulary size 的 unit vector。

- Build max heap：呼叫可將 list 變成 root 也就是第一個 element 的 similarity 最大的 list。

```
def max_heapify(P,k):
    l = left(k)
    r = right(k)
```

```
    if l < len(P) and P[l] > P[k]:
        largest = l
    else:
        largest = k
    if r < len(P) and P[r] > P[largest]:
        largest = r
    if largest != k:
        P[k], P[largest] = P[largest], P[k]
        max_heapify(P, largest)
def left(k):
    return 2 * k + 1
def right(k):
    return 2 * k + 2
def build_max_heap(P):
    n = int((len(P)//2)-1)
    for k in range(n, -1, -1):
        max_heapify(P,k)
```

- HAT initialization

```
  # Simple HAT with complete link
  print("Init HAC")
  C = []
  P = []
  I = []
  A = []
  N = len(TFtds)
  K = [8, 13, 20]
  for n in range(N):
    C.append([])
    P.append([])
    for i in range(N):
      sim = cosine(ALL_unitV[n+1], ALL_unitV[i+1])
      C[n].append([sim, i+1])
    I.append(1)
    P[n] = C[n].copy()
    P[n].remove(C[n][n])
    build_max_heap(P[n])
# P: n[ i[ [1.0, 1], [0.2705026724104795, 2],...], ...] (sorted by sim)
# C: n[ i[ [1.0, 1], [0.2705026724104795, 2],...], ...]
```

```
  print("Finish init HAC")
```

初始化 C 跟 P，並移除掉 P 當中的 self similarity。利用 build_max_heap 將 P 轉成 priority queue。

- HAC

```
print("Start HAC")
  for k in range(N-1):
    # 從所有還活著的 cluster 中找到最大的 similarity 抓出 k1 k2 做合併
    max_sim = 0
    for j in range(N):
      if I[j] == 1:
# P[j][0] => 在 P[j=docID]中 sim 最高的 pair => P[j][0] = [sim, with docID]
        sim = P[j][0][0]
        if max_sim <= sim:
          max_sim = sim
          k1 = min((j+1), P[j][0][1])
          k2 = max((j+1), P[j][0][1])
    print("max_sim: ", max_sim)
    print(k1, k2)
    A.append([k1, k2])
    I[k2-1] = 0
    # 當發現剩下 8, 13, 20 個 cluster 活著的時候，呼叫 Kdoc 產出 K.txt 檔。
    # K = 8, 13, 20
    if sum(I) in K:
      Kdoc(I, A)
    # 將 k1 的 Priority queue 初始化
    P[k1-1] = []
    # 進行 update
    for j in range(N):
      if I[j] == 1 and j != (k1-1):
        P[j].remove(C[j][k1-1])
        P[j].remove(C[j][k2-1])
        # Complete Linked 使用 min 來做 complete linked
        newsim = min(C[j][k1-1][0], C[j][k2-1][0])
        C[j][k1-1][0] = newsim
        P[j].append([newsim, k1])
        build_max_heap(P[j])
        C[k1-1][j][0] = newsim
```

```
        P[k1-1].append([newsim, j+1])
        build_max_heap(P[k1-1])
  print("Finish HAC")
```

- Kdoc：用來產出 K.txt 文件。

```
def Kdoc(I, A):
  # 利用 I 來抓出最後還存活的 cluster ID 並初始化 cluster dictionary
  # ex：live = [0, 2] => 初始化 cluster = { 1:[1], 3:[3] }
  I_np = np.asarray(I)
  live = np.where(I_np == 1)[0]
  live = list(live)
  cluster = {}
  for l in live:
    cluster[l+1] = [l+1]
  # 將 A 複製一份並倒過來
  # ex: A = [[8,9], [3,7], [3,6], [1,8]]  ([當次活下來的, 被吃掉的])
  # A_r = [[1,8], [3,6], [3,7], [8,9]]
  A_r = A.copy()
  A_r.reverse()
  for a in A_r:
# 若當次活下來的最後沒有活下來，就去找他被誰吃掉過了，把這次被吃掉的加進那個 cluster。
    if a[0]-1 not in live:
      for key, vallist in cluster.items():
        if a[0] in vallist:
          cluster[key].append(a[1])
# 若當次活下來的最後活下來了，把這次被吃掉的加進他的 cluster。
    else:
      cluster[a[0]].append(a[1])
  kdoc = ""
  filename = str(sum(I))
  for key, item in cluster.items():
    item = sorted(item)
    for i in item:
      kdoc += str(i)+ '\n'
    kdoc += '\n'
  with open('.\\'+ filename + '.txt','w') as f:
    f.write(kdoc)
```