

HW4 Report

R07725021 資管碩 1 洪靖雯

1.執行環境

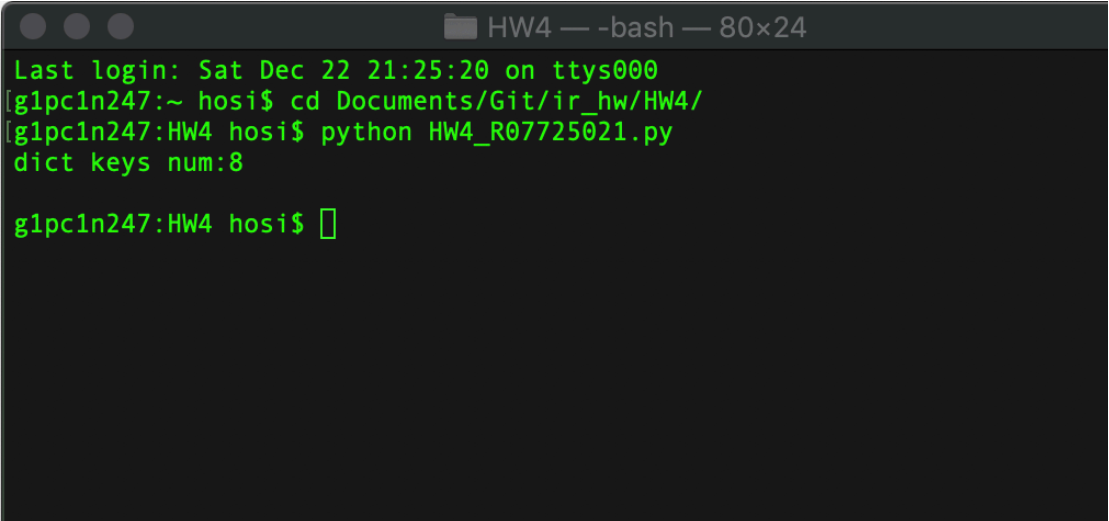
- Sublime
- Cmder

2.程式語言

- Python 3.6.6

3.執行方式

- 套件: nltk、bs4、defaultdict、numpy
- 編譯方式: 使用 `python HW4_R07725021.py` 即可

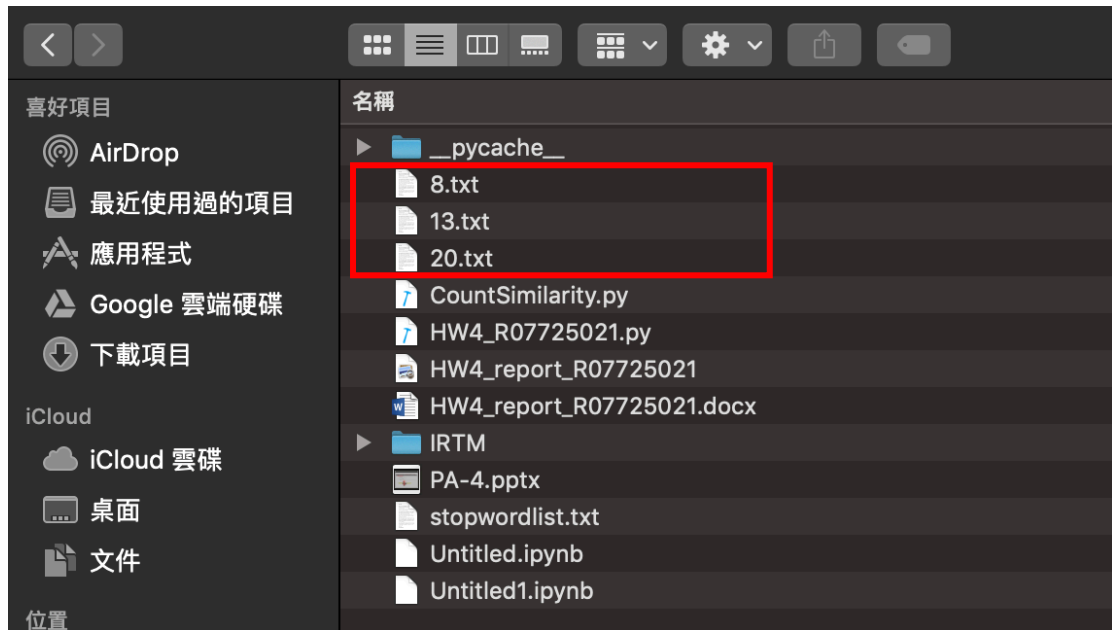


```
HW4 — -bash — 80x24
Last login: Sat Dec 22 21:25:20 on ttys000
[g1pc1n247:~ hosi$ cd Documents/Git/ir_hw/HW4/
[g1pc1n247:HW4 hosi$ python HW4_R07725021.py
dict keys num:8

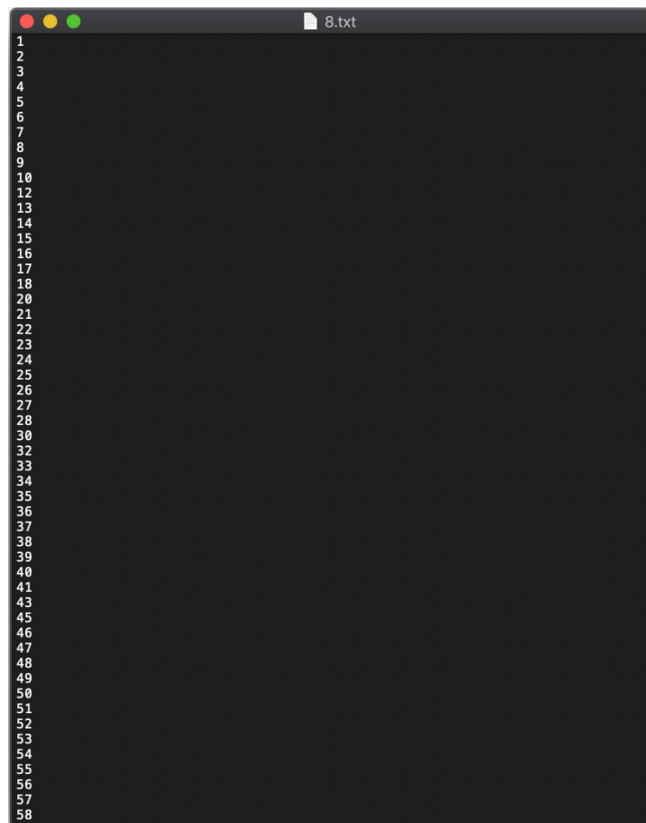
g1pc1n247:HW4 hosi$
```

- 輸出結果

- x.txt (總共有三個檔案，分別是群體數目為 8、13、20 個的結果)



ii. 8.txt 結果示意圖



4.作業處理邏輯說明

本次作業有兩個檔案 - CountSimilarity.py 及 HW4_R07725021.py，以

下分別介紹：

i. CountSimilarity.py

CountSimilarity.py 其實是把 HW2 的程式寫成 Class，供主程式呼叫。

主要做的事情為將 1095 個 document 做 tokenize，個別計算 normalized tf-idf vectors，最後輸入任兩文章 docid，便可得到這兩個文章的 cosine similarity。

➤ Import 必要的套件

```
1 import requests
2 from bs4 import BeautifulSoup
3 import re
4 from nltk.tokenize import word_tokenize
5 from nltk.stem import PorterStemmer
6 import math
7 from collections import defaultdict
8
```

➤ 各文章 tokenize，計算各文章各個 term 的 normalized tf-idf vectors 值，用 nested dict 方式儲存，key 為 **docid**, subkey 為 **term**, value 為 **tf-idf** 值。

◆ 主程式

```
9 class CountSimilarity():
10     global dict_tf_idf
11     dict_tf_idf = defaultdict(dict) #key為docid,value為term和sqrt_tfidf (di
12
13     def main(self): #主要運作流程，計算出tf-idf
14         dict_df_all,dict_tf_each_doc = self.tokenize() #dict_tf_all:得到1095
15         for i in range(1,1096): #計算tf-idf
16             vectorLength = self.countDocVectorLength(dict_tf_each_doc[i]) #
17             for term,tf in dict_tf_each_doc[i].items():
18                 tf_idf = self.findIDF(dict_df_all,term,tf) #count tf-idf
19                 sqrt_tfidf = float(tf_idf/vectorLength) #get tf-idf unit ve
20                 dict_tf_idf[i][term] = sqrt_tfidf
21
```

◆ Tokenize

```

22 def tokenize(self):
23     #initialnize
24     dict_df_all = {}
25     dict_tf_each_doc = defaultdict(dict)
26
27     # 讀取document collection
28     for i in range(1,1096):
29         file = open("IRTM/"+str(i)+".txt","r")
30         # file to string array
31         data = file.read()
32         file.close()
33
34         #tokenize(改用nltk的tokenizer)
35         tokendata = word_tokenize(data)
36         regex = [c for c in tokendata if c.isalpha()] #去除non-alphanumeric
37
38         #lowercasing
39         lowerString = [elements.lower() for elements in regex]
40
41         #stemming
42         stemmer = PorterStemmer()
43         stemString = [stemmer.stem(lowerstring) for lowerstring in lowerString]
44
45         #stopword removal
46         #因為會有回應時間過長問題，直接抓下來讀取檔案
47         file2 = open("stopwordlist.txt","r")
48         stopwordList = file2.read()
49         file2.close()
50         removeStopword = [word for word in stemString if word not in stopwordList]

```

◆ 計算 tf-idf，再計算 vector length，得到 normalized tf-idf vector

```

81 def findIDF(self,dict_df,term,tf): #該term的df，，將df變idf
82     #在dictionary.txt找到該term的df，將df變idf
83     df = dict_df.get(term)
84     idf = math.log10(1095/float(df))
85     tf_idf = float(tf)*float(idf)
86     return float(tf_idf)

```

```

74 def countDocVectorLength(self,dict): #計算vector length
75     squareNum = 0
76     for term,tf in dict.items():
77         squareNum += int(tf) * int(tf) #tf^2相加
78     sqrtNum = math.sqrt(squareNum) #最後squart roots
79     return sqrtNum

```

◆ 最後再給任兩個 docid，計算這兩篇文章的 cosine similarity 並回傳值

```

88     def cosineSimilarity(self, dict_tf_idf, dict1, dict2): #計算兩篇文章的cosi
89         count = 0
90         for term_1 in dict1: #比較dict2是否有包含dict1的key，有表示含有同樣的te
91             if term_1 in dict2:
92                 tf_idf_1 = dict1[term_1] #dict1的value(tf-idf)
93                 tf_idf_2 = dict2[term_1] #dict2的value(tf-idf)
94                 count += float(tf_idf_1*tf_idf_2) #兩者相乘，加起來
95         return count #回傳結果
96
97     def countCosineSimilarity(self, doc1, doc2): #計算cosine similarity
98         dict_1 = dict_tf_idf[int(doc1)] #key為term,value為tf-idf
99         dict_2 = dict_tf_idf[int(doc2)] #key為term,value為tf-idf
100         answer = self.cosineSimilarity(dict_tf_idf, dict_1, dict_2)
101         return answer

```

ii. HW4_R07725021.py

實作 bottom up HAC，並輸出群數為 8 群、13 群及 20 群的結果

➤ Import 必要的套件，以及 CountSimilarity.py Class

```

HW4_R07725021.py x CountSimilarity.py x
1 from CountSimilarity import CountSimilarity #import CountSimilarity.py
2 from collections import defaultdict
3 import numpy as np
4

```

➤ HAC 部分

◆ 先將一些必要值、array 做 initialize

- I. clusterSimilarity 為 numpy array，為 2D ndarray，1096*1096 matrices，因為 index 想要直接是 docid，所以多了一維度浪費掉。
- II. existCluster 為現在 docid 是否還存在，1 為存在，0 表示已經 merge 掉。
- III. clusterDict 是儲存現在分群的狀態，預設是 key 和 value 都是 docid。

```

5 def simpleHAC():
6     clusterSimilarity = np.zeros(shape=(1096,1096)) #浪費空間，index比較好看
7     existCluster = np.ones(1096) #每個cluster是否還存活著，1存在、0不存在，預設都存在
8     clusterDict = defaultdict(list) #存現在merge的文章們(dict型態，key為cluster的頭，value是
9     for num in range(1,1096): #initial, key和value都是docid
10         clusterDict[num].append(num)
11

```

◆ 再從 CountSimilarity.py 那得到 1095 篇文章任兩篇的 cosine similarity

```

12     #得N*N matrix，計算任兩文章的similarity
13     for i in range(1,1096):
14         for j in range(1,1096):
15             if(i == j): #避免self-similarity
16                 continue
17             clusterSimilarity[i][j] = cosineSimilarity.countCosineSimilarity(i,j)
18

```

◆ 接著是做 merge 的動作

- I. `k` 為 merge 次數，每次為兩兩 node 做 merge，總共 merge N-1 次。
- II. `targetCluster` 為我們預計要分幾群，是看 `clusterDict` 的 keys 剩多少，一開始為 1095 個 keys，慢慢減少成我們要的群數(bottom up)
- III. 將上面的 `clusterSimilarity` 用 numpy array 裡的 `argsort` 得到這個 matrices 中 similarity 最大的 indices，用 `max_list` 儲存，並從大到小做排序。
- IV. `i` 和 `m` 各為每個 indices 中的 `index[0]`和 `index[1]`
- V. 判斷 `i` 是否不等於 `m`，且 doc `i` 和 doc `m` 是否都還活著
- VI. 將 `m` merge 到 `i` 中，並將 `m` 在 `clusterDict` 資料刪除
- VII. 接著更新其他與 `i` 有關的 `j` 的 similarity。這裡是用 **complete linkage** 的方式去做。最一開始每個 document 都是一個點，所以任兩個 document 的 cosine similarity 便是他們間的距離。而 complete link

看兩個 cluster 中最遠的點間的距離，所以更新 i 與 j 的間的距離，便是比較 i 對 j 與 m 對 j 間的 complete link 誰比較短，就取比較短的距離！(min)

VIII. 最後把 m 的 existCluster 設為 0

IX. 全部運算完後回傳 clusterDict 為答案

```
19 #merge, 把每個doc都先看成一個點, 任兩點的cosine similarity就是他們之間的距離, 先跟據哪兩個點距離最小
20 #距離 = 1-cosineSimilarity, 所以cosine similarity大, 距離短
21 #合併過後, single linkage是看[i][j]和[m][j]哪個比較短(sim比較大)
22 #complete linkage則是看[i][j]和[m][j]哪個比較長(sim比較小)
23 for k in range(1,1095): # merge次數, 每次都是兩兩merge
24     if(len(clusterDict.keys())<=targetCluster): #決定要幾群
25         break
26     #取得i,m,similarity, 比較similarity, 拿sim最大的i和m來做merge
27     max_list = np.dstack(np.unravel_index(np.argsort(-clusterSimilarity.ravel()), (1096
28     for num in max_list[0]:#取得i,m
29         i = num[0]
30         m = num[1]
31         if((i != m) and (existCluster[i] == 1) and (existCluster[m] == 1)): #i不等於j, 且
32             if(i > m): #i永遠是比較小的
33                 num = i
34                 i = m
35                 m = num
36
37         #把j merge到i去
38         clusterDict[i].extend(clusterDict[m]) #merge m的底下list到i去
39         clusterDict.pop(m, None) #如果dict裡有key為m, 要刪掉(因為m已經merge到i去了)
40
41         #找sim最大的(距離最遠, complete linkage)
42         for j in range(1,1096):
43             if((j!=i) and (j!=m)):
44                 clusterSimilarity[i][j] = min(clusterSimilarity[i][j],clusterSimila
45                 clusterSimilarity[j][i] = min(clusterSimilarity[i][j],clusterSimila
46         existCluster[m] = 0
47         break
48
49     return clusterDict
50
```

➤ 將答案依據目標群數，輸出成 x.txt (8,13,20 群)

```
51 global cosineSimilarity
52 global targetCluster #目標分成幾群
53 targetCluster = 8
54 cosineSimilarity = CountSimilarity() #引用CountSimilarity.py class
55 cosineSimilarity.main() #初始化, 先算出tf-idf值
56 answer = simpleHAC()
57 print("dict keys num:"+str(len(answer.keys()))+"\n")
58 with open(str(targetCluster)+'txt','w') as f: #列印結果, 印出每一群分佈
59     for cluster_boss,cluster_subs in answer.items():
60         for cluster_sub in sorted(cluster_subs):
61             f.write("%d\n" % (cluster_sub))
62         f.write("\n\n")
```