一、執行環境 & 作業系統

Mac Command Shell

二、程式語言, 版本

Python 3.6.2

三、執行方式

(1) 安裝以下 package：tqdm@4.19.4、nltk@3.2.5、numpy@1.13.3

```
~/Desktop/IRTM/hw/hw3 sudo pip3 install 'tqdm==4.19.4' 'nltk==3.2.5'
```

sudo pip3 install 'tqdm==4.19.4' 'nltk==3.2.5' 'numpy@1.13.3'

(2) 安裝 nltk 裡的 stopWord Corpus

```
ken@Weal222C:~/IRTM/hw/hw3$ python3
Python 3.5.2 (default, Nov 23 2017, 16:37:01)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import nltk
>>> nltk.download()
NLTK Downloader
---------------------------------------------------------------------------
    d) Download   l) List    u) Update   c) Config   h) Help   q) Quit
---------------------------------------------------------------------------
Downloader> d

Download which package (l=list; x=cancel)?
  Identifier> stopwords
    Downloading package stopwords to /home/ken/nltk_data...
      Unzipping corpora/stopwords.zip.
```

python3 > import nltk > nltk.download() > d > stopwords

(3) 執行：**python3 main.py**，耗時約 **5 分鐘**

```
~/Desktop/IRTM/hw4 python3 main.py

Preparing...

[1/5] Document collection & regularization: 100%|| 1095/1095 [00:56<00:00,
[2/5] Establish tf-idf dictionary: 100%|| 1095/1095 [00:08<00:00,
[3/5] Establish cosine_similarity dictionary: 100%|| 1096/1096 [00:39<00:00,
[4/5] Constructing the Heap of all docs' similarity: 100%|| 1095/1095 [00:00<00:00,
[5/5] Start Hierarchical Clustering (Complete-Link): 100%|| 1087/1087 [03:35<00:00,

Done.
```

四、作業處理邏輯說明

主邏輯程式主要分為三個檔案：(1) Normalize.py (2) Vectorize.py (3) main.py

(1) Normalize.py

原身為作業一的程式，主要做的事情為

- 切詞處理(tokenize)
- 將所有字符轉為小寫
- Stemming
- 移除特殊字元
- 移除 stopWords

(2) Vectorize.py

原身為作業二的程式，主要做的事情為

- 利用 **TermToIdxMake** 函式建立 term 與 idx 的映射 dictionary—**term_idx**
- 利用 **TFIDFGet** 函式與 **TransferTermToIdx** 函式建立 tf-idf 的 dictionary—**tf_idf_dicts**
- 根據此 **tf_idf_dicts** 內容，藉由 **vectorize** 函式將所有的 doc 裡所包含的 term 變為同樣長度經過正規劃的 vector
- 利用 **cosine** 函式將所要運算的 doc 做相乘運算回傳 cosine 值
- 回傳一個 N*N 的矩陣，紀錄 doc 彼此之間的 cosine similarity

(3) main.py

- 在 **EfficientHAC**，將每個 doc 視為一個 cluster，並將這些 cluster 之間的 cosine similarity 以 Heap 的資料結構(**heapq**)記錄在 **P dict**

```
heap_list = []
for targetId in range(1, N + 1):
  if (docId == targetId): continue
  heap_list.append((cosine_dict[docId][targetId], targetId))
heapq._heapify_max(heap_list)
P[docId] = heap_list
```

- 接著在 **P dict** 檢視最大的 cluster similarity 組合，將其提出並 merge

```
def selectHighestCosine(P, available):
  # P = { 1: [(cosine, id), (cosine, id), ...], 2: [(cosine, id), (cosine, id), ...], ... }
  # P[docId][0]: Heap最大的tuple:(cosine, id)，P[docId][0][0]：cosine
  idMaxCosList = [ P[docId][0][0] for docId in P ]
  MaxDocId = -1
  CurrentMax = -9999
  for i, val in enumerate(idMaxCosList):
    docId = i + 1 # 因為docId沒有0
    if (val > CurrentMax and available[docId] == True):
      CurrentMax = val
      MaxDocId = docId
  return MaxDocId
```

- 更新 **Clusters**、**Cluster_Progress**、**available**，記錄最新分群過程及結果

```
Clusters[docId] = Clusters[docId].union(Clusters[targetId])
del Clusters[targetId]
Cluster_progress[len(Clusters)] = copy.deepcopy(Clusters) #
```

- 針對所有的分群(包含新合併的)更新 **P dict**，新合併的分群使用 Complete-Link 計算 similarity

```
P[docId] = []

for i, existBool in enumerate(available):
  if (i == docId or i == 0 or existBool == False): continue
  P[i] = lazyDel(P[i], [docId, targetId])

  newMutualSim = getNewSim(Clusters[i], Clusters[docId], cosine_dict)

  cosine_dict[i][docId] = newMutualSim
  cosine_dict[docId][i] = newMutualSim

  P[i].append((cosine_dict[i][docId], docId))
  P[docId].append((cosine_dict[docId][i], i))
```

- 根據 **Cluster_Progress** 輸出 **K.txt**。(K 為不同的分群：8, 13, 20)

```
for idx, k in enumerate([8, 13, 20]):
  res = Cluster_progress[k]
  content = ''
  for _, ids in res.items():
    for i in sorted(ids):
      content += '{}\n'.format(i)
    content += '\n'
  with open(str(k) + '.txt', 'w') as f:
    f.write(content.strip())
```