R12725049 徐尚淵 作業四

1. 執行環境：Jupyter Notebook
2. 程式語言：Python (版本 3.11.4)
3. 執行方式：

   直接使用 Jupyter Notebook GUI Run code

   ▶ Run

4. 作業處理邏輯說明

   1. 讀取需使用的 package

```
import os
import re
import pandas as pd
import numpy as np
```

   2. 創建一個空的 List 名為 total_tfidf_df_list，裡面將用 List 儲存每個文件的用字

      intex 以及其 tfidf 值，並且每個文件用 dataframe 的格式來儲存

```
total_tfidf_df_list = list()

for i in range(0,1095):
    t_index_list = []
    tfidf_list = []

    path = "./output/" + str(i + 1) + ".txt"
    with open(path, 'r', encoding='utf-8') as file:
        # 跳過Header
        next(file)
        for line in file:
            parts = line.strip().split()
            if len(parts) == 2:
                t_index, tfidf = int(parts[0]), float(parts[1])
                t_index_list.append(t_index)
                tfidf_list.append(tfidf)

    single_doc_data = {'t_index': t_index_list, 'tf_idf': tfidf_list}
    single_doc = pd.DataFrame(single_doc_data)
    total_tfidf_df_list.append(single_doc)
```

   3. 定義 normalize function

計算 unit vector，並定義 cosine function 計算任兩文件的 cosine similarity (同作業二)

```
# 計算其unit vector
def norm(tfidf_List):
    norm_vector = sum(tfidf ** 2 for tfidf in tfidf_List) ** 0.5
    return norm_vector
```

```
def cosine(doc1_df, doc2_df):
    # 以t_index做outer merge
    merged_df = pd.merge(doc1_df, doc2_df, on='t_index', how='outer').fillna(0)

    # 轉換為unit vector
    doc1_norm = norm(merged_df['tf_idf_x'].to_list())
    doc2_norm = norm(merged_df['tf_idf_y'].to_list())

    # 計算cosine similarity
    similarity = np.dot(merged_df['tf_idf_x']/doc1_norm, merged_df['tf_idf_y']/doc2_norm)

    return similarity
```

4. 用名為 C 的 dataframe 來儲存兩兩文件所計算出的 similarity 結果，因為對稱矩陣所以只須計算上半三角，下半三角可直接對稱

```
column_c = list(range(0,1095))
row_c = list(range(0,1095))
```

```
C = pd.DataFrame(columns=column_c, index=row_c)
for i in range(0,1095):
    for j in range(i,1095):
        C.iat[i,j] = cosine(total_tfidf_df_list[i],total_tfidf_df_list[j])
        C.iat[j,i] = C.iat[i,j]
```

5. 定義 sim_matrix function

用以更新此 Cluster 演算法中 Similarity matrix 的 function，每次迭代都實作一次，首先確保收到的 id 為 int 格式，而後利用 np.minimium 找出兩個 cluster 中的最小值，更新到 cluster1 的欄位，並更新對稱矩陣以及設定 cluster1 自己跟自己比的 similarity 為負無限大，cluster2 亦同以免日後受到其干擾

```
def sim_matrix(similarity_matrix, cluster1_id, cluster2_id):
    # 確保其收到的資料格式為int
    cluster1_id = int(cluster1_id)
    cluster2_id = int(cluster2_id)

    # 使用廣播找出兩個索引對應的最小相似度值
    similarity_matrix[cluster1_id, :] = np.minimum(similarity_matrix[cluster1_id, :], similarity_matrix[cluster2_id, :])
    similarity_matrix[:, cluster1_id] = similarity_matrix[cluster1_id, :]  # 利用對稱性更新列
    similarity_matrix[cluster1_id, cluster1_id] = -np.inf  # 設定 cluster1 內部的值為負無窮大

    # 將 cluster2 對應的行和列設為負無窮大，表示已經合併
    similarity_matrix[cluster2_id, :] = -np.inf
    similarity_matrix[:, cluster2_id] = -np.inf
```

6. 定義 Max_pos function

   能夠找到 Test_C 這個餘弦相似矩陣中的最大元素的 index，也就是最相似的兩個文件 or 分群

```
def max_pos(test_C):
    max_position = np.unravel_index(np.argmax(test_C), test_C.shape)
    return max_position
```

7. 定義 Apply_HAC Function

   大致上遵照講義的虛擬碼、以及 complete-link 的邏輯來實作，在達到 k 類之前都會持續循環，找到最相似的兩個 cluster 的索引並合併，之後清空 k_max 表示其已經被合併，最後更新相似度矩陣並 return 結果

```
def apply_HAC(clusters, k):

    while k < len(test_C):
        k_max = max(max_pos(test_C))
        k_min = min(max_pos(test_C))
        clusters[k_min] += clusters[k_max]
        clusters[k_max] = []
        sim_matrix(test_C, k_min, k_max)  # 更新similarity matrix
        k += 1

    return clusters
```

8. 定義 Write_file Function

   用以寫入文件

```
def write_file(cluster, k):
    file_name = f"./{k}.txt"
    with open(file_name, "w") as f:
        for group in cluster:
            if group:

                f.write('\n')
                f.write('\n'.join(map(str, sorted(group))))
                f.write('\n')
```

9. 主程式部分

   For 迴圈條件設定為 8、13、20 三群，先初始化日後會用到的 output_k、test_C 以及 clusters，並將 C 複製起來以便儲存，再直接利用 apply_HAC 實作，並排序後寫入文件中。

```
# 分為8、13、20群
for k in [8, 13, 20]:
    output_k = k
    copy_C = C.copy()
    test_C = copy_C.to_numpy()
    np.fill_diagonal(test_C, 0)  # 將sim matrix對角線值設為零

    clusters = [[i + 1] for i in range(len(test_C))]  # 將每個文章都分為一個cluster

    clusters = apply_HAC(clusters, k)

    clusters = [group for group in clusters]  # 去除空的cluster

    for group in clusters:
        group.sort()  # 對每個群組的文章id進行排序

    # 寫入
    write_file(clusters, output_k)
```