資訊檢索與文字探勘導論

作業四

會計四
B06702064
林聖硯

1. 執行環境

Anaconda Spyder

2. 程式語言

Python 3.6.8

3. 執行方式

(1) 需要 import numpy 及 python 內建的 copy 模組

使用 pip install numpy

(2) 名為「result」的子資料夾內放入作業二得到每篇文章 unit-tfidf 的結果(名稱是 doc1.txt 到 doc1095.txt)

(3) 執行 120 到 130 行即可得到結果(三個文件分別存在與 py 檔同一個資料夾內)

```
120  num_doc = 1095
121  tfidf_matrix = generate_tfidf_matrix(num_doc)
122  similarity_matrix = get_cosine_sim_matrix(num_doc, tfidf_matrix)
123
124  #clustering
125  K_list = [8, 13, 20]
126  for K in K_list:
127      available_cluster = get_available_cluster_list(num_doc)
128      similarity_matrix_2 = copy.copy(similarity_matrix)
129      result = HAC(K, num_doc, similarity_matrix_2, tfidf_matrix, available_cluster)
130      save_result(K, result)
```

4. 程式處理邏輯

(0) import 需要套件

```
import numpy as np
import copy
```

----------前處理----------

(1) retrieve_index_and_tfidf_from_txt

從 txt 檔案內拿出每個字的 index 以及相對應的 tfidf

```
def retrieve_index_and_tfidf_from_txt(doc_index):
    '''
    get term indexes and their unit tfidfs from specified documents
    '''
    index = []
    unit_tfidf = []
    pos = './result/doc' + str(doc_index) + '.txt'
    with open(pos) as f:
        count = 0
        for line in f.readlines():
            count+= 1
            if (count == 1 or count == 2):
                continue
            else:
                index.append(int(line.split()[0]))
                unit_tfidf.append(float(line.split()[1]))
    index = np.array(index)
    unit_tfidf = np.array(unit_tfidf)
    return index, unit_tfidf
```

(2) generate_tfidf_matrix

將上一步得到得 tfidf 向量，存到 numpy array 裡面做出 tfidf matrix

```
#14299是從pa2得來的bag-of-word model的長度
def generate_tfidf_matrix(num_doc, len_BOW = 14299):
    tfidf_matrix = np.zeros((num_doc, len_BOW))
    for doc_index in range(1, num_doc + 1, 1):
        indexes, unit_tfidf = retrieve_index_and_tfidf_from_txt(doc_index)
        count = 0
        for index in indexes:
            #print(index, unit_tfidf[count])
            tfidf_matrix[doc_index - 1][index - 1] = unit_tfidf[count]
            count += 1
    return tfidf_matrix
```

(3) get_cosine_sim_matrix

使用 tfidf matrix 計算文章間的 cosine similarity，並且存入 shape 為 (1095, 1095)的 matrix(稱為 similarity_matrix)

```python
def compute_pairwise_cosine_similarity(array_1, array_2):
    cosine_similarity = np.dot(array_1, array_2)
    return cosine_similarity

def get_cosine_sim_matrix(num_doc, tfidf_matrix):
    cosine_sim_matrix = np.zeros((num_doc, num_doc))
    for i in range(num_doc):
        for j in range(num_doc):
            if j < i:
                similarity = compute_pairwise_cosine_similarity(tfidf_matrix[i], tfidf_matrix[j])
                cosine_sim_matrix[i][j] = similarity
                cosine_sim_matrix[j][i] = similarity
            elif j == i:
                cosine_sim_matrix[i][j] = 0
                cosine_sim_matrix[j][i] = 0
            else:
                break
    return cosine_sim_matrix
```

----------clustering----------

(1) get_available_cluster_list

製造一個程度為 1095 的 list(全部放入 1)，代表還可以被 cluster 的文章。

```python
def get_available_cluster_list(num_doc):
    available_cluster = []
    for i in range(num_doc):
        available_cluster.append(1)
    return available_cluster
```

(2) HAC 演算法

a. result (list)：每一個元素放入文章的代碼的 list(為了使用 extend function)

b. 當剩下的 cluster 大於給定的 K 時就執行 HAC 演算法。

c. index 為 similarity matrix 裡面最大的值的 row 及 column index，並以此 index 來更新 available_cluster 及 similairty_matrix

d. 這裡針對 cluster 及 cluster 之間的相似度採用 GACC(Group-average Agglomerative Clustering)

```python
def HAC(K, num_doc, sim_matrix, tfidf_matrix, available_cluster):
    result = []
    for i in range(num_doc):
        result.append([i+1])

    while(sum(available_cluster) > K):
        index = np.unravel_index(np.argmax(sim_matrix, axis=None), sim_matrix.shape)
        row_index = index[0]
        column_index = index[1]

        result[row_index].extend(result[column_index])
        result[column_index] = 0
        available_cluster[column_index] = 0

        #update similarity with row_index
        for j in range(num_doc):
            if available_cluster[j] == 0:
                new_sim = 0
            elif (j == row_index):
                new_sim = 0
            else:
                new_sim = sim_gacc(j, row_index, tfidf_matrix, result)
            sim_matrix[j][row_index] = new_sim
            sim_matrix[row_index][j] = new_sim

            #change similarity that contains "column_index" to zero
            sim_matrix[j][column_index] = 0
            sim_matrix[column_index][j] = 0

    return result
```

(3) sim_gacc

計算 gacc 相似度

```
def sim_gacc(clus_iter, clus_survive, tfidf_matrix, result):
    '''
    compute similarity for two documents by using gacc
    '''
    len_clus_iter = len(result[clus_iter])
    len_clus_survive = len(result[clus_survive])
    index_iter = np.array(result[clus_iter]) - 1
    index_survive = np.array(result[clus_survive]) - 1
    sum_iter_cluster = np.sum(tfidf_matrix[index_iter], axis = 0)
    sum_iter_survive = np.sum(tfidf_matrix[index_survive], axis = 0)
    s = np.sum([sum_iter_cluster, sum_iter_survive], axis = 0)
    similarity_gacc = (np.dot(s,s) - len_clus_iter - len_clus_survive) /  ((len_clus_iter + len_clus_survive) * (len_clus_iter + len_clus_survive - 1
    return similarity_gacc
```

(4) save_result

將結果存入 txt 檔案內

```
def save_result(K, result):
    location = "./{}.txt".format(K)
    with open(location, 'w') as f:
        for cluster in result:
            if cluster != 0:
                cluster = sorted(cluster)
                for doc in cluster:
                    f.write("{}".format(doc))
                    f.write("\n")
                f.write("\n")
```