HW1 TF-IDF 實作

一、 前置作業

使用雲端資料

```
[1]  from google.colab import drive
     drive.mount("/content/drive",force_remount=True)
     !mkdir -p drive
     !google-drive-ocamlfuse drive
```

```
Mounted at /content/drive
/bin/bash: google-drive-ocamlfuse: command not found
```

Import&path

```
[3]  import numpy as np
     import pandas as pd
     import copy
     from gensim.models import Word2Vec
     import keras
     from numpy import dot
     import math
     from sklearn.preprocessing import normalize

     file_path = './drive/My Drive/IR/HW1/'
```

由於使用 colab 先讓我們能取用 google 雲端的資料，安裝好套件後 import 需要的套件及指定檔案路徑

二、 資料讀取

```
print(query_list)
print(doc_list)
print(all_query[0])
print(all_doc[0])
```

```
['301', '302', '303', '304', '305', '306', '307',
['FBIS3-10082', 'FBIS3-10231', 'FBIS3-10243', 'FBI
['intern organ crime']
['languag f p 105 spanish f articl type bfn text s
```

```
doc_list.remove('LA072189-0048')
for i in range(len(doc_list)):
  if (doc_list[i]=='LA072189-0048'):
    print(i)
print(all_doc[3643])
del all_doc[3643:3644]
print(len(all_doc))
```

```
[]
4190
```

將資料讀取成上方形式，在過程中發現有個檔案為空白，因此將其移除，在輸出答案時再將其放置到 list 的末端

三、 TF-IDF 設計

```
##會return 所有得分跟所有欄位
def my_tfidf(all_input):
  column_name = []
  all_data_split = [] ##分割字串
  for i in range(len(all_input)):
    all_data_split.append(all_input[i][0].split())
  ## 讀入並掃次數
  voc_dic = {}
  score_dict_list = [] ##拿來存tf的分數

  example_dic = {} ##拿來乘的空dict
  for i in range(len(all_data_split)): ##幾篇文章
    score_dic = {} ##算tf用的
    updated = {} ##已經算過的字
    for j in range(len(all_data_split[i])): ##所有的字
      if (voc_dic.get(all_data_split[i][j]) != None): ##已經存在
        if (all_data_split[i][j] not in updated):    ##這篇還沒看到
          voc_dic[all_data_split[i][j]] += 1.0
          updated[all_data_split[i][j]] = 1.0
```

```
      else:         ##沒有發現過這個字
        voc_dic[all_data_split[i][j]] = 1.0   #初始化
        updated[all_data_split[i][j]] = 1.0   #加入更新過的陣列
      if score_dic.get(all_data_split[i][j]) == None:
        score_dic[all_data_split[i][j]] = 1.0 ##建一個
      else:
        score_dic[all_data_split[i][j]] += 1.0 ##計數加一
      example_dic[all_data_split[i][j]] = 0
    score_dict_list.append(score_dic) ##存了所有tf有值的分數
  column_name = list(voc_dic.keys())
  ## idf
  idf_dic = {} ##idf權重
  N = len(all_input) ##總共幾篇
  for i in voc_dic:
    idf_dic[i] = math.log(1+((N+1)/(voc_dic[i]+1))) ##sklearn文檔中的smooth_idf
  result = []
```

```
  for i in range(N):
    for value in score_dict_list[i]:
      example_dic[value] =idf_dic[value]*(1+math.log(score_dict_list[i][value]))
    score = list(example_dic.values())
    result.append(score)
    for value in score_dict_list[i]:
      example_dic[value] = 0.0
  result_score = normalize(result,norm ='l2')
  return result_score,column_name
```

將資料傳入自訂的 Function，首先會進行**字串分割**，因此可以將每個 element 存為 dict 裡的 key，以加快速度，其中各 dic 的用途如下：

- **example_dic**：記錄所有 key 值的 dict，方便所有 tf 進行運算時統一維度，為暫存 tf 運算結果的 dict
- **score_dic(tf)**：記錄該篇文章中出現過的字及次數，將其 append 至 list 得到所有 Query 及 Doc 的初步 tf 值
- **voc_dic**：記錄所有 key 值出現在多少篇文章中，單篇最多計算一次
- **updated**：記錄在此篇文章中出現過的 key 以限制單篇不重複計算
- **idf_dic**：記錄 idf 權重的 dict

TF 公式：

用 sklearn 套件中的 sublinear_tf =True 的公式，

把初步 tf 值進行運算：**1 + log (tf)** 得到的結果，

能讓 MAP 評估分數從 0.68 上升至 0.71，因此選擇此公式，

我認為這個公式讓 tf 不會為 0 是提升的關鍵。

IDF 公式：

用 sklearn 套件中的 smooth_idf =True 的公式，

把 idf 值進行運算：**log (1 + (N+1) / (ni+1))** 得到的結果，

與上面的理由相同，我認為他讓 idf 避免除 0 及避免 idf 為 0 是採用的原因

Return：

先將分數的結果進行 L2 正則化，Function 返回值會為所有文章的 TF-IDF 分數及 column 名稱

四、 實際測試

```
[ ] ## 所有文檔 包括query跟doc
    all_input = all_query + all_doc
    result_score,column_name = my_tfidf(all_input)
    df_tfidf = pd.DataFrame(result_score,columns=column_name, index=(query_list+doc_list)) ##弄成dataframe
    df_tfidf
```

▾ 處理成斷詞的格式

```
[ ] query_data = []
    doc_data = []
    for i in range(len(all_query)):
      query_data.append(all_query[i][0].split())

    for i in range(len(all_doc)):
      if len(all_doc[i]) != 0:
        doc_data.append(all_doc[i][0].split())
    print(query_data[0])
    print(doc_data[0])
```

```
['intern', 'organ', 'crime']
['languag', 'f', 'p', '105', 'spanish', 'f', 'articl',
```

將函式的回傳結果轉成 dataframe，並將所有讀取的 query、doc 進行字串分割得到所有 term

```
[ ] ##找出Index
    def find_index(input):
      max = -1.0
      id = 0
      for i in range(len(input)):
        if (input[i] > max):
          max = input[i]
          id = i
      return id
```

尋找 index 的 function

```
ans_list = []
for i in range(len(query_data)): ##總共幾個query
  score_list = []       ##每個cos_sim的得分
  query_ans_list = []   ##排序輸出結果
  query_vec = np.zeros((len(query_data[i]))) ##每個query都有N個字當作n維向量
  doc_vec = np.zeros((len(doc_data),len(query_data[i]))) ##然後把doc的每個維度讀出來
  for j in range(len(query_data[i])): ## query有幾個詞
    query_vec[j] = df_tfidf.iloc[i][query_data[i][j]] ##query中每個字的分數存成vec
    for k in range(len(doc_data)):    ##總共幾篇doc
      doc_vec[k][j] = df_tfidf.iloc[k+50][query_data[i][j]] ##拿query中的字去看每篇doc的得分 弄成vec
  ##都存好開始算每個分數
  for j in range(len(doc_data)):
    cos_sim = 0
    for k in range(len(query_data[i])):
      cos_sim += dot(dot(query_vec[k],doc_vec[j][k]),query_vec[k]) ## query_vec的權重平方
    cos_sim /= len(query_data[i])
    score_list.append(cos_sim)
  for k in range(len(doc_data)):##用function找出從高到低的分數並記錄
    pos = find_index(score_list)
    query_ans_list.append(doc_list[pos])
    score_list[pos] = -2
  ans_list.append(query_ans_list)
```

在評估相似度部分，並未採用 cos_similarity 的公式，原因下方會詳細說明，

- **評估策略**

由於評估方法為 MAP 評估，經過查詢後發現高質量的答案對於評估結果較為重要，因此不拿取整個 query 向量，而是只取要查詢的 query 中的出現的 term 當作維度。

例如：[‘intern’,’crime’]即為 2 維，對於每篇 doc 會去查找 doc 向量中這兩個的值作為評估標準

由於 doc 向量的評估值為少數的 term，因此在文章中不曾出現的機率較高 (TF-IDF 值為 0)，導致 norm(doc)接近 0，造成分母為 0 的情況

$$cos(\theta) = \frac{\vec{q} \cdot \vec{d}_j}{|\vec{q}||\vec{d}_j|} = \frac{\sum_{w_i \in V} w_{i,q} \times w_{i,j}}{\sqrt{\sum_{w_i \in V} w_{i,q}^2} \times \sqrt{\sum_{w_i \in V} w_{i,j}^2}}$$

因此將分母移除，只保留分子的點積，並且實驗發現將 query 向量點積兩

次，能得到更好的成果(0.66 → 0.68)

我認為這個方法加大了 query 的影響，使的與 query 相近的資料獲得更高的評分，從而提高了整體表現。

五、 輸出

▾ 弄出結果

```
[ ]  error_name = 'LA072189-0048' ##錯誤的檔案 把它塞在最低分
     with open(file_path+'hw1_result_mytfidf_check.txt','w') as f:
       f.write('Query,RetrievedDocuments\n')
       for i in range(50):
         f.write(query_list[i])
         f.write(',')
         for j in range(len(ans_list[0])):
           f.write(ans_list[i][j])
           f.write(' ')
         f.write(error_name)
         f.write('\n')
       f.close()
```

輸出成指定格式，並將錯誤檔案補在最後方。

● 總結與心得

在這次的作業中，一開始還不知道不能使用套件，因此在套件的各種參數上實驗才得到超過 baseline 的成績。寫完才知道要 TF-IDF 部分要自己實作。

因此任務變成：還原套件使用的公式及速度，公式部分在多方查找及比對後很快得到了解決，但是速度方面始終相差甚遠，套件只需要 5 秒，一開始完成的版本卻需要 10 分鐘，後來將 tf 的計算併入 idf 的迴圈中存成陣列才將速度加快成 1 分鐘，不過在這次的作業讓我更熟悉 dic 的運用及好處，也對 TF-IDF 有了更深入的理解。