1. **執行環境**

- Jupyter Notebook

2. **程式語言&版本**

- 語言：Python
- 版本：3.6.6

3. **執行方式**

打開 cmd，cd 到檔案位置

```
C:\Users\hp>cd C:\Users\hp\Desktop\IR\HW2_r07725044

C:\Users\hp\Desktop\IR\HW2_r07725044>
```

再輸入以下指令，即可執行並顯示 output：

```
C:\Users\hp\Desktop\IR\HW2_r07725044>python3 HW2_r07725044.py
```

藍色部分填入 python.exe 及其路徑，或是如上圖輸入在 python 開啟的指令，python3 是我在環境變數下設定 python 3.6.6 版本的指令

（在環境變數先加入 python.exe 的 path，再更改檔名為 python3.exe, 即可直接啟動）

紅色填入檔名 HW2_r07725044.py

得到 output：

```
C:\Users\hp\Desktop\IR\HW2_r07725044>python3 HW2_r07725044.py
0.18924533981387867
```

4. 作業處理邏輯說明

1. Create dictionary

- **Import 套件**

將所有要用的套件 import ，用途如註解

```
#用以定義dictionary的資料結構
from collections import defaultdict

#同作業一，斷出term的
import nltk
import string
# from collections import Counter
from nltk.stem.porter import PorterStemmer #import porter algorithm的套件
from nltk.corpus import stopwords

#打開文檔並讀取
import glob
import re
import os
import operator
import sys

#計算用的sqrt ()
import math
```

- **定義斷出 term 的 function**

以作業一的方式，做 tokenize - stemming - stopwords 作爲之後計算 idf 時的 function

```
def term(word):
    res = nltk.word_tokenize(word)
    porter = PorterStemmer()  #定義方法
    stemmer = [ porter.stem(element) for element in res]  #stemming
    stop = set(stopwords.words('english'))
    final = []
    for s in stemmer:
        if s not in stop:
            if s not in final:
                final.append(s)
    return final
```

- **計算 Document Frequency**

呼叫 term()後，在 DF 這個 dictionary 找是否存在以這個 term 的 key，有的話則 df + 1，無的話建立這個 term 的 key，將 df 設爲 1

```
path = 'IRTM/IRTM'  #文件集的 path
N = len(glob.glob1(path,"*.txt")) #計算文件個數 N
DF = defaultdict(int) #儲存每個term出現在幾篇文章的 dictionary

for filename in glob.glob(os.path.join(path, '*.txt')): #分別讀取每個文件
    words = open(filename, 'r').read().lower()  #lowercase
    word = words.translate(str.maketrans(string.punctuation, ' '*len(string.punctuation))) #將標點符號換成 whitespace，方便處理
    final = term(word)   #呼叫 term ()，斷出文章的所有 term

    for w in set(final): #判斷 term是否是字母組成，是的話 Document Frequency +1
        if w.isalpha():
            DF[w] += 1
```

- **DF 排序**

新建一 dictionary 存放排序后的結果，呼叫 sort()進行排序，並建立

```
sort_dict = {}  #存放排序後的 dictionary

#存放dictionary的key，用以排序
temp = list()
for d in DF:
    temp.append(d)
temp.sort()  #排序

#依序存入新的 dictionary
for t in temp:
    sort_dict[t] = DF[t]

#將每個term建立index
index = {}
count = 1
for s in sort_dict:
    index[s] = count
    count += 1
```

每個 term 的 index

- **存入 dictionary.txt**

定義每個 column 的 title，分別爲 t_index、term、df

```
f = open('dictionary.txt', 'w') #建立新檔案，名爲dictionary.txt
f.write('t_index'+ " "*(10 - len('t_index')) + 'term'.ljust(15," ") + 'df'.rjust(10," ") + '\n')
#寫入list的每個項目，並定義每個column的內容

for element in sort_dict:
    f.write(str(index[element]) + " "*(10 - len(str(count))) + element.ljust(15," ") + str(sort_dict[element]).rjust(10," ") + '\
    #寫入list的每個項目，並換行

f.close() #寫完，關閉檔案儲存
```

2. *convert a set of documents into tf-idf vectors*

- **計算 idf**

根據公式 $idf_t = \log_{10} \frac{N}{df_t}$ 寫計算的程式，其中 doccounter 代表文章總數，sort_dict[word]代表 term 的 df，最後的 10 則是設定 logarithm 的 base 為 10.

```
IDF = dict()
for word in sort_dict:
    IDF[word] = math.log((N / sort_dict[word]), 10) #logarithm base 10
```

- **定義計算 TF 的 function**

先以作業 1 的邏輯將文章的 term 斷出來，再放入 dictionary 的資料結構中，能同時記錄 term 和其 value，在 for loop 中檢查 dictionary 中是否存在該次的 term 作爲 key，若有，則將 value + 1 作爲頻率；若沒有，則是新建這個 term 為 key，並將預設值（0）+ 1.

```
def TF(string):
    tf = defaultdict(int) #建立Dictionary的資料結構，以term作爲key，頻率做value，e.g. 'word' : 3
    #hw1的方式產出term
    res = nltk.word_tokenize(string)
    porter = PorterStemmer()
    stemmer = [ porter.stem(element) for element in res]  #stemming
    stop = set(stopwords.words('english'))
    final = []
    for s in stemmer:
        if s not in stop:
            if s.isalpha():       #判斷是否為英文字母
                final.append(s)
    for t in final:               #將斷出來的字統計為term的次數
        tf[t] += 1

    return tf
```

註：和 term()的程式碼重複性高，但因在計算 idf 值時要待所有文件跑完才能確定 idf，無法直接計算 tf*idf，若在 term 中直接記錄 tf，則會需要定義太多變數，浪費空間，因此才分開寫 term()和 TF()

- **Normalize 並儲存成 txt（轉成 unit vector）**

用一樣的邏輯計算每個 document 的 tf，根據公式

tfidf = tf * idf 計算原始的 tfidf 值，並在讀取每篇文章的 term 時計算 document length（每個 term 的 tfidf 的平方和開根號）

最後將原始的 tfidf 除上 document length

計算完畢後面按照每個 term 對應的 index 排序

排序 function：

```
import operator
def sortdict(x):
    new = {}
    for word in x:
        new[word] = index[word]
    sort = sorted(new.items(), key=operator.itemgetter(1))  #根據dictionary的value來排序
    sort = dict(sort)
    return sort
```

計算：

```
path = 'IRTM/IRTM'
savepath = 'tfidf'
for filename in glob.glob(os.path.join(path, '*.txt')):
    #print(filename)
    #定義每個 doc的 document length，從 0 開始累加
    normal = 0
    #同前面 read file
    words = open(filename, 'r').read().lower()
    word = words.translate(str.maketrans(string.punctuation, ' '*len(string.punctuation)))

    #呼叫 TF（），計算詞頻
    tf = TF(words)

    #用以儲存 normalize過後的 tfidf 的 dictionary
    tfidf = {}

    for w in tf: #讀取這個 doc中有哪些 term，之後取出其 tfidf 值
        tfidf[w] = (tf[w] * IDF[w])   #計算原始的 tfidf
        normal = (tfidf[w]) ** 2 + normal #累加 document length
    normal = math.sqrt(normal)  #開根號

    for w in tfidf:    #normalize
        tfidf[w] = tfidf[w] / normal

    sort = sortdict(tfidf)
    #存取成以ID命名的txt
    scount = filename.replace('IRTM/IRTM',"").replace("\\","")
    filename = os.path.join(savepath, scount)

    f = open(filename, 'w')
    f.write('t_index'.ljust(15," ") + 'tf-idf'.rjust(10," ") + '\n') #寫入list的每個項目，並定義每個column的內容

    for element in sort:
        f.write(str(sort[element]).ljust(15," ") + str(tfidf[element]).rjust(10," ") + '\n') #寫入list的每個項目，並換行
    f.close() #寫完，關閉檔案儲存
```

3. Cosine similarity.

- **定義 cosine similarity.的 function**

可接受 document id，比較兩者相似度

先將 id 加上 '.txt' 轉成檔名，再分別讀取出 2 個 documents 的 unit vector。以第一份 document 的 term 作爲基礎尋找在第二份 document 是否有相同的 term，若沒有，則將該 term 設在第二份 document 的 unit vector 中，值為 0，若存在相同 term，將這個 term 在 2 份 document 中的值相乘，以此累加，最後的出的值就是 2 份 document 的相似度

```
def cosine(d1, d2):  #傳入數值，為document id，這次作業以 1，2為計算對象
    path = 'tfidf'  #讀取 d1,d2 兩份文件的位置
    #將document id 轉為對應的檔名
    xfile = str(d1) + '.txt'
    yfile = str(d2) + '.txt'

    #定義存取兩個文件的 unit vector
    x = {}
    y = {}

    c = 1
    #打開第一份文件讀取
    fix = os.path.join(path, xfile)
    with open(fix, 'r') as fx:
        for line in fx:    #由於在儲存時，將 first line作為該column的title，因此需從 second line讀
            if c == 1:
                c +=1
                continue
            (key, val) = line.split()  #讀取 term index 和 normalize過後的 tfidf
            x[key] = val

    #第二份文件的處理，同第一份文件
    cou = 1
    fiy = os.path.join(path, yfile)
    with open(fiy, 'r') as fy:
        for line in fy:
            if cou == 1:
                cou +=1
                continue
            (key, val) = line.split()
            y[key] = val

    summ = 0.0   #用以累加個內積的值

    #以第一份文件找第二份文件有無對應到的term
    for s in x:
        if s not in y:   #若 x存在一 term是 y沒有的，則將這個 term設給 y，值為 0，方便之後做內積
            y[s] = 0
        summ = summ + float(x[s])*float(y[s])  #unit vector內積，極爲 2 documents的相似度
    return summ
```

- **產生 document 1 & 2 的 similarity**

傳入 document id 1 和 2，得到相似度為 0.18924533981387867

```
sim = cosine(1, 2)
print(sim)
```

```
0.18924533981387867
```