1. 執行環境：Jupyter Notebook
2. 程式語言：Python 3
3. 執行方式：
    1. $ pip install nltk (for python)
    2. ~~$ pip install scipy (for python)~~
       from numpy import dot
       from numpy.linalg import norm
    3. Open 'pa1.py' and run
        - on terminal: $ python3 pa2.py

    4. Output (print): 為了方便觀察執行過程，會印出當前正在處理的 txt 檔案 id 以及其包含的 term, tf , idf , tf-idf
    5. Output (file): （藍色粗體）

```
.
├── data
├── dictionary.txt: 來自所有文件 (1.txt～1095.txt) 的 term。
├── output: ~~Doc{id}~~ -> doc{id}.txt: 紀錄所有文件 (1.txt～1095.txt) 中個別的 term 數量、term 與其 tf-idf 值。
├── pa2.py
└── report
```

4.作業處理邏輯說明：

1. 載入套件：**scipy與計算cosine similarity相關的套件改為 numpy.dot**

```
import re
import os
import nltk
import math
import numpy as np
from numpy import dot
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords

nltk.download('stopwords')
```

2. Preprocessing (修改自 pa1.py)
   在 pa1 是先做 stemming 再移除 stop words ，當時沒有遇到太大問題；但這次要處理的文件較多，發現 stop words 會被 stem 掉，於是把兩者順序對調，並在最後移除單一字母如 a, x, w…

```
# 將pa1包成 preprocess function
def preprocess(text):

    # tokenization
    tokens = re.findall("[a-zA-Z]+", text)
    # lower-casing
    lowercase = [token.lower() for token in tokens]
    # stop-words removal
    stops = stopwords.words('english')
    filtered = [w for w in lowercase if w not in stops]
    # stemming
    ps = PorterStemmer()
    stemming = [ps.stem(i) for i in filtered]

    # remove single character like 'a', 'x', 'w'...
    word = [i for i in stemming if len(i) > 1]

    return word
```

3. Function defined
    1. 定義 count 用以計算 document frequency 並更新 dictionary 中的次數
    2.tf, idf 的計算

```
def count(term, dictionary):  # count document frequency
    if term in dictionary:
        dictionary[term] += 1
    else:
        dictionary.update({term: 1})
```

```
'''
Term Frequency: 單篇文章出現頻率(document-wise)
For example,
Doc1: I am so so happy
-> tf(so, Doc1) = 2/5 = 0.4
'''


def tf(term, tokens):
    freq_dict = dict([(token, tokens.count(token)) for token in tokens])
    freq_query = freq_dict.get(term, 0)  # return default value '0' when key missing
    len_doc = sum(freq_dict.values())
    result = freq_query / len_doc
    return term, result
```

```
'''
Document Frequency: 出現在幾篇文章(collection-wise)
idf(Inverse Document Frequency): The idf of a rare term is high, and low for a frequent term
'''


def idf(term, documents):
    doc_count = 0
    for document in documents:
        if term in document:
            doc_count += 1
        result = math.log10(len(documents) / (doc_count + 1))
    return result
```

4. 宣告需要用到的 dictionary 與 array（詳見註解）
    a. document 儲存前處理（preprocessing）完的 term
    b. dictionary 儲存 term 與 document frequency
    c. key2Index 是用來存反向的 dictionary，用 key 對照到該 term 在 dictionary 中的 index。
    d. read_sequence 紀錄 txt 讀取跟處理的順序。

```
path = "./data" # collection folder
files = os.listdir(path) # get all files under collection folder

# store document as
# [['apple', 'banana'], ['cat', 'dog']]
documents = []

# {term: doc_fre} EX: {'apple':2, 'banana':1}
dictionary = {}

# {term: index} EX: {'apple': 41...}
key2Index = {}

# record txt file read sequence
read_sequence = []
```

5. 前處理所有文件並將所有 tokens 加到 documents (list)

```
for file in files: # 遍歷資料夾
    if file.endswith(".txt"):
        read_sequence.append(file.split(".")[0])
        with open(path+"/"+file, 'rb') as txt:  # read txt files
            content = txt.read().decode('latin-1').replace('\n', '')  # decoding and ignore new line
            tokens = preprocess(content)
            documents.append(tokens)
```

6. 計算 tokens 的數量並存到 dictionary，此階段會產生 dictionary.txt

```
for document in documents:
    for token in document: # counting doc. frequency
        count(token, dictionary) # 計算 token 個數並記錄到 dictionary

# 寫入 dictionary.txt
print("Create dictionary")
dict_path = "./dictionary.txt"  # create dictionary.txt
f = open(dict_path,'w')
f.write("t_index\tterm\tdf\r")  # header
for idx, key in enumerate(sorted(dictionary)):  # using enumerate to get index and items
    toWrite = "%d\t%s\t%s\r" % (idx+1, key, dictionary[key])
    key2Index.update({key:idx+1})
    f.write(toWrite)
f.close()
```

7. 計算 tf-idf unit vector 並存為 ~~Doc{id}~~ -> doc{id}
    - 其中 output_dict 是用來記錄 document-wise 中每個 term 的 index 與它的 tf-idf，也就是題目要求的 ~~Doc{id}~~ -> doc{id} 主要內文

```
print("Calculate tf-idf")
output_path = "./output/"

for idx, document in enumerate(documents):
    print(str(idx) + ": " + read_sequence[idx])  # logging

    # output format
    output_dict = {}  # {idx: tf-idf}

    # unit vector
    unit_vec = 0

    f = open(output_path+'Doc'+read_sequence[idx]+'.txt','w')

    for token in document:
        # calculate tf
        word, term_tf = tf(token, document)
        # calculate idf
        term_idf = idf(token, documents)
        # calculate tf-idf
        tf_idf = term_tf*term_idf

        unit_vec += tf_idf**2

        print('%d\r%s\t%f\t%f\t%f' % (idx, word, term_tf, term_idf, tf_idf))

        if key2Index.get(word) not in output_dict:  # avoid output duplicate term in DocID.txt
            output_dict.update({key2Index.get(word) : tf_idf})

    unit_vec_root = math.sqrt(unit_vec)

    f.write(str(len(output_dict))+'\r')  # term amount
    f.write("t_index\ttf-idf\r")  # header

    for key in sorted(output_dict):  # sorted output dictionary
        f.write("%s\t%f\r" % (key, output_dict.get(key)/unit_vec_root))
    f.close()
```

- 這邊為了查看執行過程，會輸出 log 如下：

```
Calculate tf-idf
0: 1053
earthquak       0.007905        0.990196        0.007828
struck  0.007905        1.376656        0.010883
citi    0.007905        0.698970        0.005525
fill    0.003953        1.240074        0.004901
voic    0.011858        1.483112        0.017586
giggl   0.003953        2.562293        0.010128
children        0.015810        1.194316        0.018882
march   0.007905        1.261263        0.009970
narrow  0.003953        1.562293        0.006175
street  0.007905        0.788994        0.006237
teacher 0.007905        1.808965        0.014300
parad   0.003953        1.696991        0.006707
mark    0.003953        1.104916        0.004367
india   0.011858        1.200565        0.014236
republ  0.003953        1.213339        0.004796
day     0.011858        0.360896        0.004279
```

8. 寫出 function cosine()計算兩文件的 cosine similarity

```
'''
Input two document id to calculate their cosine similarity
through tf-idf unit vector
'''

def cosine(Docx, Docy):
    dict_x = {}  # dictionary of document x
    dict_y = {}  # dictionary of document y

    with open('./output/doc' + str(Docx) + '.txt',
              'r') as txt:  # read file write to dictionary
        content_x = [
            i.replace("\n", "").split("\t") for i in txt.readlines()[2:]
        ]

    for item in content_x:
        dict_x.update({item[0]: item[1]})

    with open('./output/doc' + str(Docy) + '.txt',
              'r') as txt:  # read file write to dictionary
        content_y = [
            i.replace("\n", "").split("\t") for i in txt.readlines()[2:]
        ]

    for item in content_y:
        dict_y.update({item[0]: item[1]})

    for item in dict_x:  # check items in another dict(y), if non-exist->give 0
        if item not in dict_y:
            dict_y.update({item: 0})

    for item in dict_y:  # check item in another dict(x), if non-exist->give 0
        if item not in dict_x:
            dict_x.update({item: 0})

    x = []  # create list x[] to store tf-idf score of document x
    y = []  # create list y[] to store tf-idf score of document y

    for key in sorted(dict_x):  # call sorted dictionary x
        x.append(float(dict_x.get(key)))  # get value: string to float

    for key in sorted(dict_y):  # call sorted dictionary y
        y.append(float(dict_y.get(key)))  # get value of y

    x = np.array(x)  # convert list x to array for calculating cosine similarity
    y = np.array(y)  # convert list y to array for calculating cosine similarity

    similarity_scores = np.dot(x,y) / np.sqrt(np.dot(x, x) * np.dot(y, y))

    return similarity_scores
```

- 讀取先前步驟產生的 doc{id}.txt，用 dictionary 存 Docx, Docy 中各自的 tf-idf。
- 檢查 Doc{x,y} 的 term 是否也存在對方 dictionary 中，若沒有則在 dictionary {x,y} 加上該 term 並將數量設為 0。
- ~~透過 scipy 套件 spatial 計算兩文件 tf-idf unit vector 的 cosine similarity。~~

-> 透過 numpy 的dot, sqrt 計算兩文件 tf-idf unit vector 的 cosine similarity。

- (Verified) 用 cosine 計算兩文件相似度相同的文件值應為 1。
- The cosine similarity between document 1 and 2 is 0.21。

```
cosine(1, 1)
```

executed in 11ms, finished 01:41:51 2021-12-04

```
1.0
```

```
cosine(1, 2)
```

executed in 8ms, finished 01:41:52 2021-12-04

```
0.20455809049576973
```