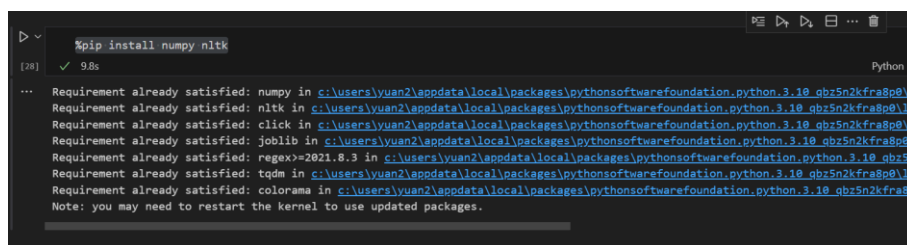
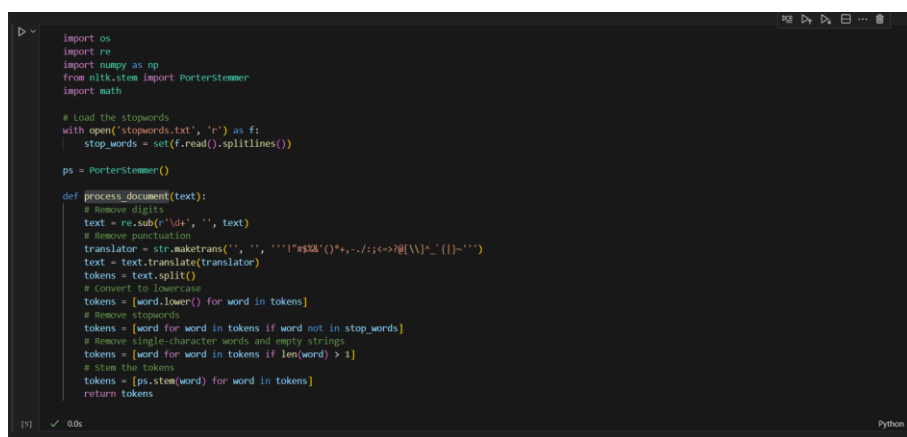


1. 執行環境：VS code
2. 程式語言：python 3.10
3. 執行方式：
 - I. 使用 VS code 或 Jupyter Notebook 打開 **pa2.ipynb**
 - II. 執行第一個 cell 安裝需要的 packages (**numpy**, **nlTK**)



```
[28] ✓ 9.8s Python
... Requirement already satisfied: numpy in c:\users\yuan2\appdata\local\packages\pythonsoftwarefoundation.python.3.10.qbz5n2kfra8p0\
Requirement already satisfied: nltk in c:\users\yuan2\appdata\local\packages\pythonsoftwarefoundation.python.3.10.qbz5n2kfra8p0\
Requirement already satisfied: click in c:\users\yuan2\appdata\local\packages\pythonsoftwarefoundation.python.3.10.qbz5n2kfra8p0\
Requirement already satisfied: joblib in c:\users\yuan2\appdata\local\packages\pythonsoftwarefoundation.python.3.10.qbz5n2kfra8p0\
Requirement already satisfied: regex>=2021.8.3 in c:\users\yuan2\appdata\local\packages\pythonsoftwarefoundation.python.3.10.qbz5
Requirement already satisfied: tqdm in c:\users\yuan2\appdata\local\packages\pythonsoftwarefoundation.python.3.10.qbz5n2kfra8p0\
Requirement already satisfied: colorama in c:\users\yuan2\appdata\local\packages\pythonsoftwarefoundation.python.3.10.qbz5n2kfra8
Note: you may need to restart the kernel to use updated packages.
```

- III. 確保 stopwords.txt 在相同的目錄中，執行第二個 cell，import 必要的 packages 以及 pa1 tokenize 的處理函數



```
[1] ✓ 0.0s Python
import os
import re
import numpy as np
from nltk.stem import PorterStemmer
import math

# load the stopwords
with open('stopwords.txt', 'r') as f:
    stop_words = set(f.read().splitlines())

ps = PorterStemmer()

def process_document(text):
    # Remove digits
    text = re.sub(r'\d+', '', text)
    # Remove punctuation
    translator = str.maketrans('', '', '!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~')
    text = text.translate(translator)
    tokens = text.split()
    # Convert to lowercase
    tokens = [word.lower() for word in tokens]
    # Remove stopwords
    tokens = [word for word in tokens if word not in stop_words]
    # Remove single-character words and empty strings
    tokens = [word for word in tokens if len(word) > 1]
    # Stem the tokens
    tokens = [ps.stem(word) for word in tokens]
    return tokens
```

IV. 執行第三個 cell，會在目錄下產生 dictionary.txt

```
# Initialize a dictionary for document frequency
document_frequency = {}

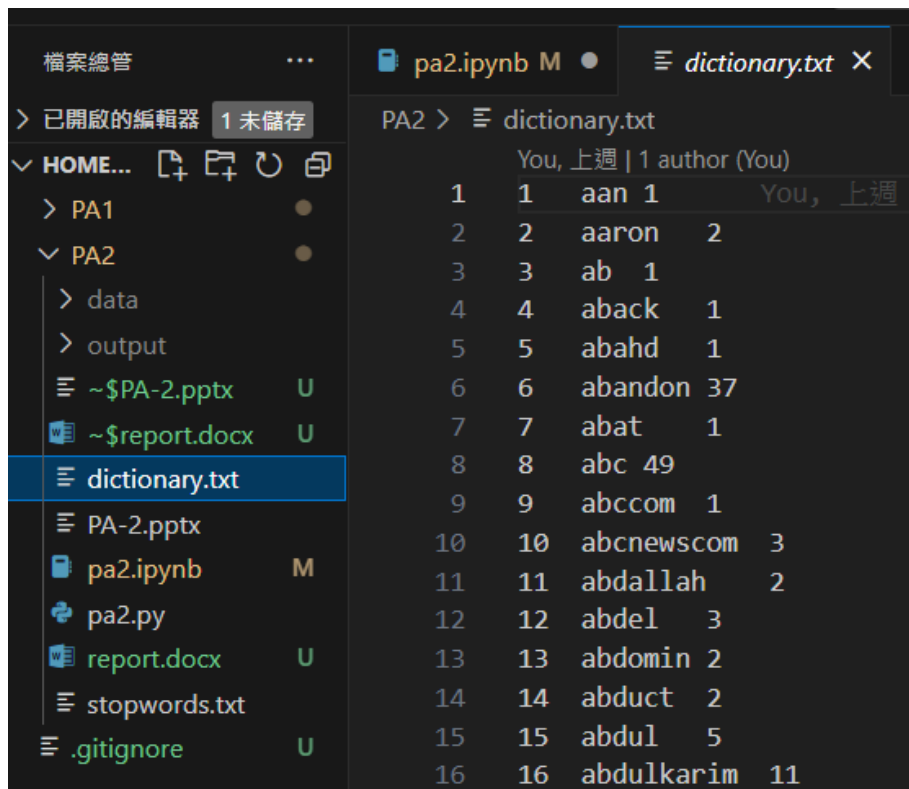
# Iterate through all the files in the dataset directory
for filename in os.listdir('./data'):
    if filename.endswith('.txt'):
        filepath = os.path.join('./data', filename)
        with open(filepath, 'r', encoding='utf-8') as f:
            text = f.read()
            tokens = process_document(text)

            # Update document frequency - only count each term once per document
            unique_tokens = set(tokens)
            for token in unique_tokens:
                if token in document_frequency:
                    document_frequency[token] += 1
                else:
                    document_frequency[token] = 1

# Sort the terms in ascending order
sorted_terms = sorted(document_frequency.items(), key=lambda x: x[0])

# Save the dictionary and document frequency to a file
with open('dictionary.txt', 'w') as f:
    for index, (term, df) in enumerate(sorted_terms, start=1):
        f.write(f'{index}\t{term}\t(df)\n')
```

dictionary.txt 截圖



Index	Term	Document Frequency (df)
1	aan	1
2	aaron	2
3	ab	1
4	aback	1
5	abahd	1
6	abandon	37
7	abat	1
8	abc	49
9	abccom	1
10	abcnewscom	3
11	abdallah	2
12	abdel	3
13	abdomin	2
14	abduct	2
15	abdul	5
16	abdulkarim	11

V. 執行第四個 cell，會在 `./output` 資料夾中，產生每一個文件的 `{DocID.txt}`，裡面包含所有的 unit tfidf vectors

```
# Build a term index dictionary the new index
term_index = {}
for index, (term, df) in enumerate(sorted_terms, start=1):
    term_index[index] = (term, df)

# Now compute the tf-idf vectors for each document
for filename in os.listdir('data'):
    if filename.endswith('.txt'):
        filepath = os.path.join('data', filename)
        with open(filepath, 'r', encoding='utf-8') as f:
            text = f.read()
            tokens = preprocess(text)

# Compute term frequency without Counter
tf = {}
for token in tokens:
    if token in tf:
        tf[token] += 1
    else:
        tf[token] = 1

# Create a term vector if length equal to the number of terms
tfidf_vector = {}
for term, freq in tf.items():
    if term in term_index:
        df = term_index[term][1]
        idf = 1 / math.log(df + 1) # Assuming all files in the dataset directory are text documents
        tfidf = freq * idf
        tfidf_vector[term_index[term][0]] = tfidf

# Normalize the tf-idf vector to unit length
norm = math.sqrt(sum(tfidf**2 for tfidf in tfidf_vector.values()))
if norm > 0:
    tfidf_vector_unit = {term: tfidf_vector[term] / norm for term in tfidf_vector}

# Get non-zero entries for the sparse representation
non_zero_entries = [(index + 1, tfidf) for index, tfidf in enumerate(tfidf_vector_unit) if tfidf > 0]

# Save the tfidf unit vector to a file
doc_id = os.path.splitext(filename)[0] # Assuming filename is 'DocID.txt'
with open(f'./output/{doc_id}.txt', 'w') as f:
    f.write(f'{len(non_zero_entries)}\n') # Write the number of non-zero entries
    for index, tfidf in non_zero_entries:
        f.write(f'{index}\t{tfidf:.3f}\n') # Write the term index and tf-idf value, formatted to 3 decimal places
```

1.txt 截圖

檔案總管		pa2.ipynb M	1.txt	
已開啟的編輯器 1 未儲存		PA2 > output > 1.txt		
HOME...	PA1	1	108	
	PA2	2	70 0.053	
	data	3	219 0.041	
	output	4	955 0.077	
	1.txt	5	1052 0.048	
	2.txt	6	1069 0.291	
	3.txt	7	1210 0.137	
	4.txt	8	1654 0.027	
	5.txt	9	1885 0.049	
	6.txt	10	2043 0.070	
	7.txt	11	2118 0.036	
		12	2148 0.087	
		13	2237 0.097	
		14	2350 0.081	

VI. 執行第五個 cell，會將指定 documents 的 vector 載入並且計算 cosine similarity (更改不同的 docx 以及 docy 編號，可以計算不同的文章相似性)

```
def cosine(docx, docy):
    # inline function to load vector
    def load(doc_id):
        with open(f'./output/{doc_id}.txt', 'r') as f:
            lines = f.readlines()
            vector = np.zeros(len(term_index)) # Ensure all vectors are of same length as term_index
            for line in lines[1:]:
                index, tfidf = line.strip().split()
                index = int(index) - 1 # Indices start from 1 in the file
                tfidf = float(tfidf)
                vector[index] = tfidf
            return vector

    vector_x = load(docx)
    vector_y = load(docy)

    # The vectors are already normalized (unit vectors), so just compute the dot product.
    cosine_similarity = np.dot(vector_x, vector_y)

    return cosine_similarity

# Example usage:
docx = '1'
docy = '8'
similarity = cosine(docx, docy)
print(f'Cosine similarity between {docx} and {docy}: {similarity:.3f}')

[30] ✓ 0.0s
... Cosine similarity between 1 and 8: 0.216
```

4. 作業處理邏輯說明：

Construct a dictionary based on the terms extracted from the given documents.

I. 先將 pa1 的 tokenization 寫成一個函數，處理依序為，移除數字、移除標點符號、Lowercasing、移除 stopwords、移除長度為一或零的字串、使用 NLTK library 中的 PorterStemmer 進行 stemming。

```
def process_document(text):
    # Remove digits
    text = re.sub(r'\d+', '', text)
    # Remove punctuation
    translator = str.maketrans('', '',
    '!"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~''')
    text = text.translate(translator)
    tokens = text.split()
    # Convert to lowercase
    tokens = [word.lower() for word in tokens]
    # Remove stopwords
    tokens = [word for word in tokens if word not in stop_words]
    # Remove single-character words and empty strings
    tokens = [word for word in tokens if len(word) > 1]
```

```
# Stem the tokens
tokens = [ps.stem(word) for word in tokens]
return tokens
```

II. 計算 document frequency 以及建立 `dictionary.txt`

將 `./data` 中 1095 個 txt 檔案依序讀入，對於單一個 document，先使用前述 `process_document` 的函數進行 tokenization，獲得單一 document 的 tokens 之後，用 `set` 轉換為 `unique_tokens`，並更新整個 collection 的 document frequency。

```
document_frequency = {}
# Iterate through all the files in the dataset directory
for filename in os.listdir('./data'):
    if filename.endswith('.txt'):
        filepath = os.path.join('./data', filename)
        with open(filepath, 'r', encoding='utf-8') as f:
            text = f.read()
            tokens = process_document(text)
            # Update document frequency - only count each term once per
document
            unique_tokens = set(tokens)
            for token in unique_tokens:
                if token in document_frequency:
                    document_frequency[token] += 1
                else:
                    document_frequency[token] = 1
```

最後以 ascending order 重新排序 document frequency，並存入 `dictionary.txt` 中

```
# Sort the terms in ascending order
sorted_terms = sorted(document_frequency.items(), key=lambda x:
x[0])
# Save the dictionary and document frequency to a file
with open('dictionary.txt', 'w') as f:
    for index, (term, df) in enumerate(sorted_terms, start=1):
        f.write(f"{index}\t{term}\t{df}\n")
```

Transfer each document into a tfidf unit vector.

- III. 計算 term frequency (tf)，從 `./data` 中 1095 個 txt 檔案中，再次經由 `process_document` 提取出單一 document 的 tokens，接著計算這些 document 的 term frequency。

```
# Build a term index dictionary for easy lookup
term_index = {term: index for index, (term, df) in
enumerate(sorted_terms, start=1)}

# Now compute the tf-idf vectors for each document
for filename in os.listdir('./data'):
    if filename.endswith('.txt'):
        filepath = os.path.join('./data', filename)
        with open(filepath, 'r', encoding='utf-8') as f:
            text = f.read()
            tokens = process_document(text)

        # Compute term frequency
        tf = {}
        for token in tokens:
            if token in tf:
                tf[token] += 1
            else:
                tf[token] = 1
```

接著，使用先前已經在計算 document frequency 時，建立好的 `term_index` 以及 document frequency 來計算 tfidf 的值。

```
# Create a zero vector of length equal to the number of terms
tfidf_vector = np.zeros(len(term_index))

for term, freq in tf.items():
    if term in term_index:
        tf_t = freq
        df_t = document_frequency[term]
        N = len(os.listdir('./data')) # Assuming all files
in the dataset directory are text documents
```

```

idf_t = math.log10(N / df_t)
tfidf_t = tf_t * idf_t
tfidf_vector[term_index[term] - 1] = tfidf_t # -1
because indices start from 1

```

將計算計算好的 tfidf 值轉換為 unit-tfidf，並且防止除以零的情況。

```

# Normalize the tf-idf vector to unit length
norm = np.linalg.norm(tfidf_vector)
if norm > 0:
    tfidf_vector_unit = tfidf_vector / norm
else:
    tfidf_vector_unit = tfidf_vector # avoid division by
zero

```

最後將單一 document 中有的 term，依照 term index 以及 tfidf

的值，儲存在 {DocID.txt} 中。

```

# Get non-zero entries for the sparse representation
non_zero_entries = [(index + 1, tfidf) for index, tfidf in
enumerate(tfidf_vector_unit) if tfidf > 0]

# Save the tf-idf unit vector to a file
doc_id = os.path.splitext(filename)[0] # Assuming filename
is 'DocID.txt'
with open(f'./output/{doc_id}.txt', 'w') as f:
    f.write(f"{len(non_zero_entries)}\n") # Write the
number of non-zero entries
    for index, tfidf in non_zero_entries:
        f.write(f"{index}\t{tfidf:.3f}\n") # Write the term
index and tf-idf value, formatted to 3 decimal places

```

Write a function **cosine(Doc_x, Doc_y)** which loads the tf-idf vectors of documents *x* and *y* and returns their cosine similarity.

IV. 最後，寫一個函數，先載入單一 document 的 unit tfidf vectors，然後再計算 cosine similarity。

```
def cosine(docx, docy):
    # Inline function to load vector
    def load(doc_id):
        with open(f'./output/{doc_id}.txt', 'r') as f:
            lines = f.readlines()
            vector = np.zeros(len(term_index)) # Ensure all vectors are
of same length as term_index
            for line in lines[1:]:
                index, tfidf = line.strip().split()
                index = int(index) - 1 # Indices start from 1 in the
file
                tfidf = float(tfidf)
                vector[index] = tfidf
            return vector

    vector_x = load(docx)
    vector_y = load(docy)

    # The vectors are already normalized (unit vectors), so just
compute the dot product.
    cosine_similarity = np.dot(vector_x, vector_y)

    return cosine_similarity
```

以下為範例用法，輸出為 Cosine similarity between 1 and 2: 0.195

```
# Example usage:
docx = '1'
docy = '2'
similarity = cosine(docx, docy)
print(f'Cosine similarity between {docx} and {docy}:
{similarity:.3f}')
```