R12725049 徐尚淵 作業二

1. 執行環境：Jupyter Notebook
2. 程式語言：Python (版本 3.11.4)

```
In [1]: from platform import python_version

        print(python_version())

        3.11.4
```

3. 執行方式：

   直接使用 Jupyter Notebook GUI Run code

   ▶ Run

4. 作業處理邏輯說明

   1. 先做與作業一相同的前處理 (簡化截圖)

```
In [2]: import os
        import math

        # 資料夾路徑
        folder_path = r'./data'

        # 存儲文檔內容的列表
        document_contents = []

        # 列出資料夾中的所有文件
        for filename in os.listdir(folder_path):
            # 文件路徑
            file_path = os.path.join(folder_path, filename)

            if os.path.isfile(file_path):
                with open(file_path, "r", encoding="utf-8") as file:
                    document_contents.append(file.read())
```

```
In [3]: print(document_contents[0])

        the white house is also keeping a close watch on yugoslavia, where
        opposition forces are about to step up the pressure on president slobodan
        milosevic. but will it work? nbc's jim maceda is in belgrade tonight.
```

```
In [7]: # Empty List for storing processed document
        processed_texts = []

        for document_content in document_contents:
            result = []
            # lower casting
            document_content = document_content.lower()
            # Tokenized
            tokenized_content = tokenize_text(document_content)
            # Stopwords removal
            filtered_tokens = [token for token in tokenized_content if token not in stop
            # Stemming
            for t in filtered_tokens:
                result.append(ps.stem(t))
            # 存入List
            processed_texts.append(result)

        # 以文件一做測試
        print(processed_texts[0])

        ['white', 'hous', 'also', 'keep', 'close', 'watch', 'yugoslavia', 'opposit', 'f
        orc', 'step', 'pressur', 'presid', 'slobodan', 'milosev', 'work', 'nbc', 'jim',
        'maceda', 'belgrad', 'tonight', 'serbia', 'eve', 'gener', 'strike', 'two-hour',
        'roadblock', 'tast', 'come', 'tomorrow', 'say', 'opposit', 'nationwid', 'work',
```

2. 計算每個文字在 Document 當中出現的頻率，利用 term_info_List 儲存之後可能會用到的資訊。

In [8]:
```
# Empty list 用以儲存每個單字的index,term,frequency
term_info_list = []

# Empty dict for storing term_document_frequency
term_document_frequency = {}

# Loop start
for words in processed_texts:
    # 避免重複計算
    unique_words = set(words)

    # 更新Frequency
    for word in unique_words:
        if word in term_document_frequency:
            term_document_frequency[word] += 1
        else:
            term_document_frequency[word] = 1

# 按照字母順序排列字典
sorted_terms = sorted(term_document_frequency.items(), key=lambda x: x[0])

# 紀錄 index並存入 term_info_list 中
for index, (term, frequency) in enumerate(sorted_terms, start=1):
    term_info = {'index': index, 'term': term, 'frequency': frequency}
    term_info_list.append(term_info)
```

3. 將 Dictionary 輸出並命名為 dictionary.txt

:
```
# 儲存字典為txt
output_file = "dictionary.txt"

with open(output_file, "w", encoding="utf-8") as file:
    # Header
    file.write("t_index   term                    df\n")

    for term_info in term_info_list:
        line = "{:<8}  {:<25}  {}\n".format(term_info['index'], term_info['term']
        file.write(line)

print(f"字典已保存到 {output_file}")
```

字典已保存到 dictionary.txt

4. 計算每個文件的 tf-idf vector

Transfer each document into a tf-idf unit vector.

In [10]:
```
# Empty List for storing each document
document_term_frequencies = []

# Loop start
for words in processed_texts:
    # Empty dict for storing document_term_frequency
    document_term_frequency = {}

    # 更新frequency
    for word in words:
        if word in document_term_frequency:
            document_term_frequency[word] += 1
        else:
            document_term_frequency[word] = 1

    # 存入個文件的 List
    document_term_frequencies.append(document_term_frequency)

# 計算文件總數
total_documents = len(processed_texts)

tfidf_vectors = []

for document_term_frequency in document_term_frequencies:
    tfidf_vector = []

    for word, term_frequency in document_term_frequency.items():
        # 計算 TF
        tf = term_frequency / sum(document_term_frequency.values())

        # 計算 IDF
        idf = math.log10(total_documents / term_document_frequency[word])

        # 計算 TF-IDF
        tfidf = tf * idf

        # 擷取 term_index
        term_index = next((item['index'] for item in term_info_list if item['term'] == word), None)

        # Add term_index and TF-IDF 值到向量
        tfidf_info = {'index': term_index, 'tf-idf': tfidf, 'tf' : tf, 'idf' : idf}
        tfidf_vector.append(tfidf_info)

    tfidf_vectors.append(tfidf_vector)
```

:
```
# 以隨機文件做測試
print(tfidf_vectors[597])
```

```
[{'index': 14237, 'tf-idf': 0.0009284568811649149, 'tf': 0.00263852242744063
3, 'idf': 0.3518851579615027 7}, {'index': 10994, 'tf-idf': 0.0380854911796817
```

5. 將每個文件的 tf-idf 儲存

```
output_folder = "output"
os.makedirs(output_folder, exist_ok=True)

# 將每個文件的 tf-idf 儲存
for document_index, words in enumerate(processed_texts, start=1):
    # 文件名為 Docid.txt
    document_filename = f"{document_index}.txt"

    with open(os.path.join(output_folder, document_filename), "w", encoding="utf
        # Header
        file.write("t_index   tf-idf\n")

        for tfidf_info in tfidf_vectors[document_index - 1]:
            line = "{:<8}  {}\n".format(tfidf_info['index'], tfidf_info['tf-idf'
            file.write(line)
```

6. Cosine Similarity Function

```
In [13]: def cosine(vector_x, vector_y):
             # 計算向量內積
             dot_product = sum(x['tf-idf'] * y['tf-idf'] for x, y in zip(vector_x, vector

             # 計算向量大小
             magnitude_x = math.sqrt(sum(x['tf-idf']**2 for x in vector_x))
             magnitude_y = math.sqrt(sum(y['tf-idf']**2 for y in vector_y))

             # Cosine similarity
             similarity = dot_product / (magnitude_x * magnitude_y)

             return similarity

         def unit_vector(vector):
             # 計算向量大小
             magnitude = math.sqrt(sum(x['tf-idf']**2 for x in vector))

             # 計算單位向量
             unit_vector = [{'tf-idf': x['tf-idf'] / magnitude} for x in vector]

             return unit_vector
```

7. 先計算單位向量再計算 Cosine Similarity

```
In [14]: # 計算兩文件各自的單位向量
         unit_vector_x = unit_vector(tfidf_vectors[154])
         unit_vector_y = unit_vector(tfidf_vectors[958])

         # 計算cosine similarity
         similarity = cosine(unit_vector_x, unit_vector_y)

         print(f"文件 x 和文件 y 的cosine similarity: {similarity}")
```

```
文件 x 和文件 y 的cosine similarity: 0.6642675123400708
```