

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



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 re
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
filenames = os.listdir(folder_path)
|
# 按照數字的順序排序文件名
def custom_sort(filename):

    match = re.match(r'(\d+)', filename)
    if match:
        return int(match.group())
    return filename

# 排序
sorted_filenames = sorted(filenames, key=custom_sort)

for filename in sorted_filenames:
    # 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 [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'], term_info['frequency'])
        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 因 term_document_frequency 在前面做過了因此此處可以直接讀取

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

        # 獲取 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': 0.35188515796150277}, {'index': 10994, 'tf-idf': 0.0380854911796817
31, 'tf': 0.0000000000000000, 'idf': 0.0000000000000000}, {'index': 3126

```

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. 讀取文件並將其儲存為 df 的形式以便日後處理資料的 Function

```

In [13]: import pandas as pd

def getDoc(filepath):
    t_index_list = []
    tfidf_list = []

    # 讀取檔案並將其轉為Dataframe形式以便之後處理資料
    with open(filepath, 'r', encoding='utf-8') as file:
        # 跳過Header
        next(file)
        for line in file:
            parts = line.strip().split()
            if len(parts) == 2:
                t_index, tfidf = int(parts[0]), float(parts[1])
                t_index_list.append(t_index)
                tfidf_list.append(tfidf)

    data = {'t_index': t_index_list, 'tf-idf': tfidf_list}
    df = pd.DataFrame(data)
    return df

```

```

## 以文件一做測試
dataframe = getDoc('./output/1.txt')
print(dataframe)

```

```

      t_index    tf-idf
0      11970  0.005025
1       4970  0.003534
2        338  0.002228
3       5710  0.005692
4       1955  0.003865
..      ...      ...
120     7207  0.002693
121     6102  0.005632
122    10426  0.007297
123     7561  0.008118
124     7217  0.003042

```

```

[125 rows x 2 columns]

```

7. 分別計算其 unit vector 的 Function

```

In [15]: # 計算其unit vector
def norm(tfidf_List):
    norm_vector = sum(tfidf ** 2 for tfidf in tfidf_List) ** 0.5
    return norm_vector

```

8. 讀取檔案並計算兩者的 cosine similarity

```
: def cosine(doc1, doc2):  
  
    #讀取兩個檔案  
    doc1_df = getDoc(doc1)  
    doc2_df = getDoc(doc2)  
  
    #將tf-idf欄位分別重新命名，以便merge後做內積  
    doc1_df.rename(columns={'tf-idf': 'docX'}, inplace=True)  
    doc2_df.rename(columns={'tf-idf': 'docY'}, inplace=True)  
  
    # 以t_index做outer merge  
    merged_df = pd.merge(doc1_df, doc2_df, on='t_index', how='outer').fillna(0)  
  
    # 轉換為unit vector  
    docX_norm = norm(merged_df['docX'].to_list())  
    docY_norm = norm(merged_df['docY'].to_list())  
  
    similarity = 0  
  
    # 計算cosine similarity  
    for index, row in merged_df.iterrows():  
        similarity += (row['docX']/docX_norm) * (row['docY']/docY_norm)  
  
    return similarity  
  
: similarity = cosine('./output/1.txt', './output/2.txt')  
print(f'Cosine Similarity: {similarity}')  
  
Cosine Similarity: 0.20222730203538028
```