資訊檢索與文字探勘導論

作業二

會計四
B06702064
林聖硯

1. 執行環境

Anaconda Spyder

2. 程式語言

Python 3.6.8

3. 執行方式

(1) 需要 import nltk(preprocessor.py 檔需要)及 numpy 套件
使用 pip install nltk、pip install numpy

(2) preprocessor.py 放入 hw1 裡面文字前處理的 function

(3) 請先在建一個名為「result」的資料夾(存放 Task2 tfidf 的結果)
(放在 zip 檔內的 Doc1 是我手動移動出來的)

(4) 打開 pa2.py 並執行 217-241 行即可得到 cosine similarity 的結果

*備註

(1) 請依序執行(後面的 Task 會需要用到前面生成的東西)

(2) 整個執行過程大約需要 6 分鐘(我一直用 for + list 抱歉 QQ)

(3) txt 檔及 stopword 檔已經寫死在程式裡

(4) 若要計算不同文章的 cosine similarity 請更改 num_doc1 及 num_doc2 傳入的參數，直接輸入想要的兩篇文章號碼即可

```
#####Task 1: get document frequency#####
number_txt = generate_txt_list()
raw_text = get_raw_text(number_txt)
bag_of_words = get_BOW(raw_text)
df = get_df(bag_of_words, raw_text)
words_index = get_word_index(bag_of_words)
save_df(words_index, bag_of_words, df)
#195秒完成

#####Task 2: Transfer each document into a tf-idf unit vector#####
num_doc = get_doc_length(raw_text)
len_BOW = get_BOW_length(bag_of_words)
idf_array = get_idf_array(df)
tf_matrix = get_tf_matrix(raw_text, bag_of_words)
tfidf_matrix = get_tfidf_matrix(tf_matrix, idf_array, len_BOW)
unit_tfidf_matrix = get_unit_tfidf_matrix(num_doc, tfidf_matrix)
save_unit_tfidf_matrix(num_doc, unit_tfidf_matrix, number_txt)
#140秒完成

#####Task 3: compute cosine similarity#####
num_doc1 = 1
num_doc2 = 2
len_BOW = get_BOW_length(bag_of_words)
index_1, index_2, unit_tfidf_1, unit_tfidf_2 = retrieve_index_and_tfidf(num_doc1, num_doc2)
cosine_similarity = compute_cosine_similarity(index_1, index_2, unit_tfidf_1, unit_tfidf_2, len_BOW)
print(f"Cosine similarity of document{num_doc1} and document{num_doc2} is", round(cosine_similarity, 5))
```

結果呈現

```
In [32]: print(f"Cosine similarity of document{num_doc1} and document{num_doc2} is",
round(cosine_similarity, 5))
Cosine similarity of document1 and document2 is 0.18093
```

4. 作業處理邏輯說明

(0)import 套件

導入 numpy 及 Preprocessor 檔案

```python
#import necessary modules from hw1
from Preprocessor import text_preprocessing, get_text

#import necessary module
import numpy as np

```

(Task 1-1)讀檔

generate_txt_list:製造出['1.txt', '2.txt',...]的 list

get_raw_text:讀取每個文本並且經過 preprocessing 之後存到 list

```python
#####Task 1: caculate document frequency#####
def generate_txt_list():
    '''
    generate a list containing ['1.txt', '2.txt', ...]
    '''
    number_txt = []
    for i in range(1095):
        number_txt.append(str(i+1) + ".txt")
    return number_txt

def get_raw_text(number_txt):
    '''
    retrieve 1095 documents from txt files
    output : list of list ([news[preprocessed words, prerpocessed words, ...]])
    '''
    raw_text = []
    for txt in number_txt:
        text = get_text(txt)
        text_final = text_preprocessing(text)
        raw_text.append(text_final)
    return raw_text
```

(Task 1-2)

get_BOW:創造出 bag of words 的 list (裡面的 token 順序就固定下來)

get_df:計算 BOW 裡面每個 token 在幾個文本出現過

```python
def get_BOW(raw_text):
    '''
    generate bag of words model
    '''
    bag_of_words = []
    for text in raw_text:
        bag_of_words.extend(text)
    bag_of_words = dict.fromkeys(bag_of_words)
    bag_of_words = sorted(bag_of_words)
    bag_of_words = bag_of_words[1:]
    return bag_of_words
```

```python
def get_df(bag_of_words, raw_text):
    '''
    generate document frequency in a list
    the order of the words is unchanged
    '''
    df = []
    for word_find in bag_of_words:
        count = 0
        for news in raw_text:
            for word_news in news:
                if (word_find == word_news):
                    count += 1
                    break
        df.append(count)
    return df
```

(Task 1-3)

get_word_index:替 BOW 裡面的每個 token 創造出對應的 term_index(從 1 開始)

save_df:將 term_index, term, document frequency 存到 txt 裡

```python
def get_word_index(bag_of_words):
    '''
    generate index for each word in BOW
    '''
    words_index = [i+1 for i in range(len(bag_of_words))]
    return words_index

def save_df(words_index, bag_of_words, df):
    '''
    save document frequency of each word into dictionary.txt
    '''
    with open('dictionary.txt', 'w') as f:
        f.write('{:<15} {:<15} {}'.format('t_index', 'term', 'df'))
        f.write('\n')
        for i in range(len(words_index)):
            f.write('{:<15} {:<15} {}'.format(words_index[i], bag_of_words[i], df[i]))
            f.write('\n')
```

(Task 2-1)

get_doc_length:計算有幾篇 document，先存著備用

get_BOW_length:計算 BOW 裡面有幾個 token，存著備用

get_idf_array:使用 df 計算 idf，並且存到 numpy array 方便後面計算

```python
#####Task 2: Transfer each document into a tf-idf unit vector#####
def get_doc_length(raw_text):
    '''
    return number of total documents
    '''
    num_doc = len(raw_text)
    return num_doc

def get_BOW_length(bag_of_words):
    '''
    return length of BOW
    '''
    len_BOW = len(bag_of_words)
    return len_BOW

def get_idf_array(df):
    '''
    turn df into idf numpy array
    '''
    df_array = np.array(df)
    num_doc = len(raw_text)
    idf_array = np.log10(num_doc/df_array)
    return idf_array
```

(Task 2-2)

get_tf_matrx:生成 tf 矩陣

get_tfidf_martrix:使用 idf vector 和 tf matrix 生成 tfidf 矩陣

```python
def get_tf_matrix(raw_text, bag_of_words):
    '''
    building tf matrix (14269, 1095)
    row : term, column : document
    '''
    tf = []
    #1095, 14299
    for news in raw_text:
        temp = []
        for word_find in bag_of_words:
            count = 0
            for word_news in news:
                if (word_find == word_news):
                    count += 1
            temp.append(count)
        tf.append(temp)
    #tf-idf
    tf_matrix = np.array(tf)
    tf_matrix = tf_matrix.transpose()
    return tf_matrix

def get_tfidf_matrix(tf_matrix, idf_array, len_BOW):
    '''
    combine tf matrix and idf vector into tfidf matrix
    return (14269, 1095) (term, document)
    '''
    tfidf_list = []
    for i in range(len_BOW):
        tfidf = tf_matrix[i] * idf_array[i]
        tfidf_list.append(tfidf)
    tfidf_matrix = np.array(tfidf_list)
    return tfidf_matrix
```

(Task 2-3)

get_unit_tfidf_matrix:normalize tfidf

save_unit_tfidf_matrix:將 normalized 過後的結果存入 txt 檔

```python
def get_unit_tfidf_matrix(num_doc, tfidf_matrix):
    '''
    normalize tfidf matrix with respect to each document
    return (14269, 1095) (term, document)
    '''
    unit_tfidf_list = []
    for i in range(num_doc):
        tfidf_sum = np.sqrt(np.sum(np.square(tfidf_matrix.transpose()[i])))
        unit_tfidf_list.append(tfidf_matrix.transpose()[i]/tfidf_sum)
    unit_tfidf_matrix = np.array(unit_tfidf_list)
    unit_tfidf_matrix = unit_tfidf_matrix.transpose()
    return unit_tfidf_matrix
```

```python
def save_unit_tfidf_matrix(num_doc, unit_tfidf_matrix, number_txt):
    '''
    save tf-idf file
    '''
    for i in range(num_doc):
        doc_tfidf = unit_tfidf_matrix.transpose()[i]
        #nonzero_index start from zero to 14298
        nonzero_index = np.nonzero(doc_tfidf)[0]
        location = './result/doc'+ number_txt[i]
        with open(location, 'w') as f:
            f.write('{}'.format(len(nonzero_index)))
            f.write('\n')
            f.write('{:<10} {}'.format('t_index', 'tf-idf'))
            f.write('\n')
            for index in nonzero_index:
                f.write('{:<10} {}'.format(words_index[index], doc_tfidf[index]))
                f.write('\n')
```

(Task3-1)

retrieve_index_and_tfidf:將 unit_tfidf 及其 index 從 txt 檔取出

```
#####Task 3: compute cosine similarity#####
def retrieve_index_and_tfidf(num_doc1, num_doc2):
    '''
    get term indexes and their unit tfidfs from specified documents
    '''
    index_1 = []
    index_2 = []
    unit_tfidf_1 = []
    unit_tfidf_2 = []
    pos1 = './result/doc' + str(num_doc1) + '.txt'
    pos2 = './result/doc' + str(num_doc2) + '.txt'
    with open(pos1) as f:
        count = 0
        for line in f.readlines():
            count+= 1
            if (count == 1 or count == 2):
                continue
            else:
                index_1.append(int(line.split()[0]))
                unit_tfidf_1.append(float(line.split()[1]))
    with open(pos2) as f:
        count = 0
        for line in f.readlines():
            count+= 1
            if (count == 1 or count == 2):
                continue
            else:
                index_2.append(int(line.split()[0]))
                unit_tfidf_2.append(float(line.split()[1]))
    index_1 = np.array(index_1)
    index_2 = np.array(index_2)
    unit_tfidf_1 = np.array(unit_tfidf_1)
    unit_tfidf_2 = np.array(unit_tfidf_2)
    return index_1, index_2, unit_tfidf_1, unit_tfidf_2
```

(Task3-2)

compute_cosine_similarity:計算兩篇文章的餘弦相似度，並回傳結果

```
def compute_cosine_similarity(index_1, index_2, unit_tfidf_1, unit_tfidf_2, len_BOW):
    unit_tfidf_array_1 = np.zeros(len_BOW)
    unit_tfidf_array_2 = np.zeros(len_BOW)
    count_1 = 0
    for index in index_1:
        unit_tfidf_array_1[index-1] = unit_tfidf_1[count_1]
        count_1 += 1

    count_2 = 0
    for index in index_2:
        unit_tfidf_array_2[index-1] = unit_tfidf_2[count_2]
        count_2 += 1
    cosine_similarity = np.dot(unit_tfidf_array_1, unit_tfidf_array_2)
    return cosine_similarity
```