

106-1 資訊檢索與文字探勘導論 Programming Assignment 2 Report

R06725032 資管碩一 吳定寰

1. 執行環境

(.ipynb) Jupyter notebook

2. 程式語言與版本

Python 3.6

3. 執行方式及安裝套件

(1). `pip install nltk, scikit-learn`(需先安裝 `numpy` 及 `scipy`)

(2). 執行方式：

直接在 Jupyter 內 run cell 即可，會在同資料夾內輸出 `dictionary.txt`

檔，在 `Vector Files` 資料夾內會輸出每個 `txt file` 的 `vector file`，並會

在 cell 的下方 print 出 `1.txt` 與 `2.txt` 的 `cosine similarity`，約為

`0.2415868251107653`，執行方式如下圖所示。

```

In [11]: import re, nltk, os, collections

from nltk.stem.porter import *
from sklearn import feature_extraction
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# R0672532
DOCUMENTS_DIR = "RTM"
OUTPUT_DIR = "."
VECTORFILE_DIR = ".\\Vector Files"
dictionary = {}
seq_dict = {}
all_terms = []
for filename in os.listdir(DOCUMENTS_DIR):
    txt_dict = {}
    txt_term = []
    terms = ""
    # 開啟news.txt檔並作初步的分割，去執行英文小寫化
    raw = open(os.path.join(DOCUMENTS_DIR, filename), 'r', encoding='UTF-8').read().lower().strip().split()

    # token化分割的單詞
    tokens = []
    for text in raw:
        tokens.append(re.compile("[^a-zA-Z]").sub(" ", text))

    # 使用nltk的PorterStemmer
    stemmer = PorterStemmer()
    singles = [stemmer.stem(token) for token in tokens]

    # 刪除stopwords.txt去除stopwords並輸出結果
    stopwords = open(os.path.join(OUTPUT_DIR, 'stopwords.txt'), 'r', encoding='UTF-8').read()
    filtered_words = [word for word in singles if word not in stopwords]

    # 將txt file每個term整理成dictionary並計算df
    for word in filtered_words:
        if word not in dictionary:
            dictionary.update({word: 1})
        else:
            dictionary[word] += 1
            terms += (word + ' ')
    od = collections.OrderedDict(sorted(dictionary.items(), key=lambda t: t[0]))
    output_file2 = open(os.path.join(OUTPUT_DIR, 'dictionary.txt'), 'w', encoding='UTF-8')
    output_file2.write("L_index\tterm\tidf\n")
    seq = 1
    # 輸出所有dictionary內的term及df
    for k, v in od.items():
        seq_dict.update({k: seq})
        output_file2.write(str(seq) + '\t' + k + '\t' + str(v) + '\n')
        seq += 1
    output_file2.close()
    all_terms.update({str(filename): terms})

# 使用scikit-learn計算tf-idf
for filename in os.listdir(DOCUMENTS_DIR):
    # 將詞語換為矩陣
    vectorizer = CountVectorizer()
    # 計算每個詞語的tf-idf
    transformer = TfidfTransformer()
    # 前個fit_transform用於計算tf-idf，後個fit_transform將詞語換為矩陣
    tfidf = transformer.fit_transform(vectorizer.fit_transform(all_terms[str(filename)]))
    # 獲取所有詞語
    word = vectorizer.get_feature_names()
    # 將tf-idf換為矩陣
    weight = tfidf.toarray()
    output_file3 = open(os.path.join(VECTORFILE_DIR, filename), 'w', encoding='UTF-8')
    output_file3.write(str(len(word)) + '\nt_index\ttf-idf\n')
    # 輸出結果
    for i in range(len(weight)):
        for j in range(len(word)):
            if (str(word[j]) in seq_dict[str(word[j]])) + '\t' + str(weight[i][j]) + '\n'
            else:
                output_file3.write("ERROR\n")
    output_file3.close()

# 讀取docnames的vector file並轉為矩陣
def read_file_to_list(docname):
    doc_content = open(os.path.join(VECTORFILE_DIR, docname), 'r', encoding='UTF-8').readlines()
    doc_content = doc_content[2:]
    result = []
    for line in doc_content:
        result.append(line.replace("\n", "").split("\t"))
    return result

# 計算cosine similarity並print出結果
def cosine(docx, docy):
    docx_list = read_file_to_list(docx)
    docy_list = read_file_to_list(docy)
    cosine_sim_score = 0.0

    i, j = 0, 0

    while (i < len(docx_list) and j < len(docy_list)):
        x_t_index = int(docx_list[i][0])
        y_t_index = int(docy_list[j][0])
        x_tfidf = float(docx_list[i][1])
        y_tfidf = float(docy_list[j][1])

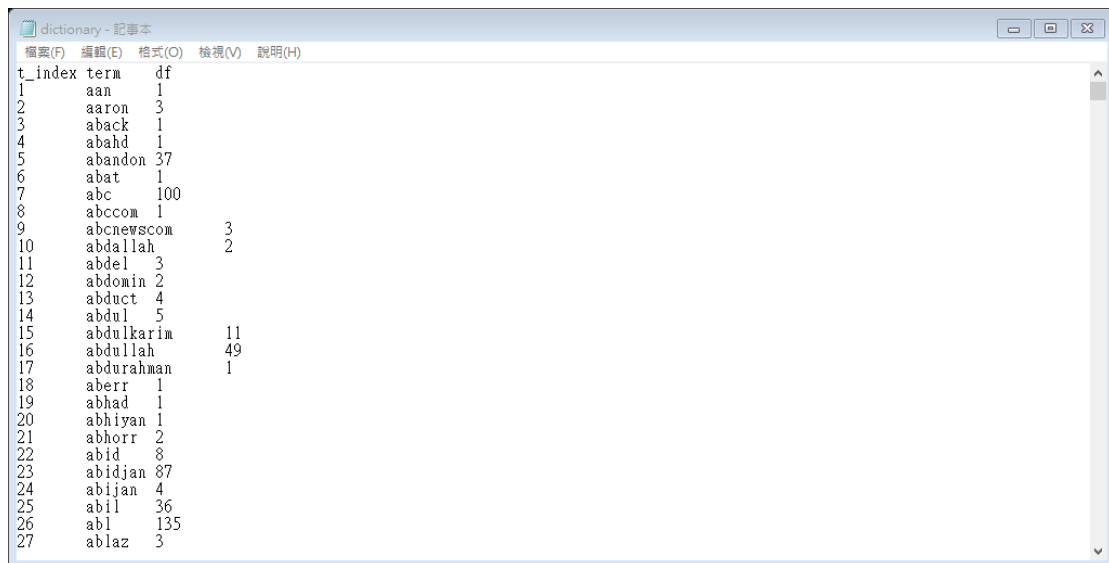
        if x_t_index == y_t_index:
            cosine_sim_score += x_tfidf * y_tfidf
            i += 1
            j += 1
        else:
            if x_t_index > y_t_index:
                j += 1
            else:
                i += 1
    print(cosine_sim_score)

cosine('1.txt', '2.txt')
0.2415868251107653

```

在 Jupyter notebook 開啟 textProcessing.ipynb 檔並執行

同資料夾內會輸出 dictionary.txt 檔

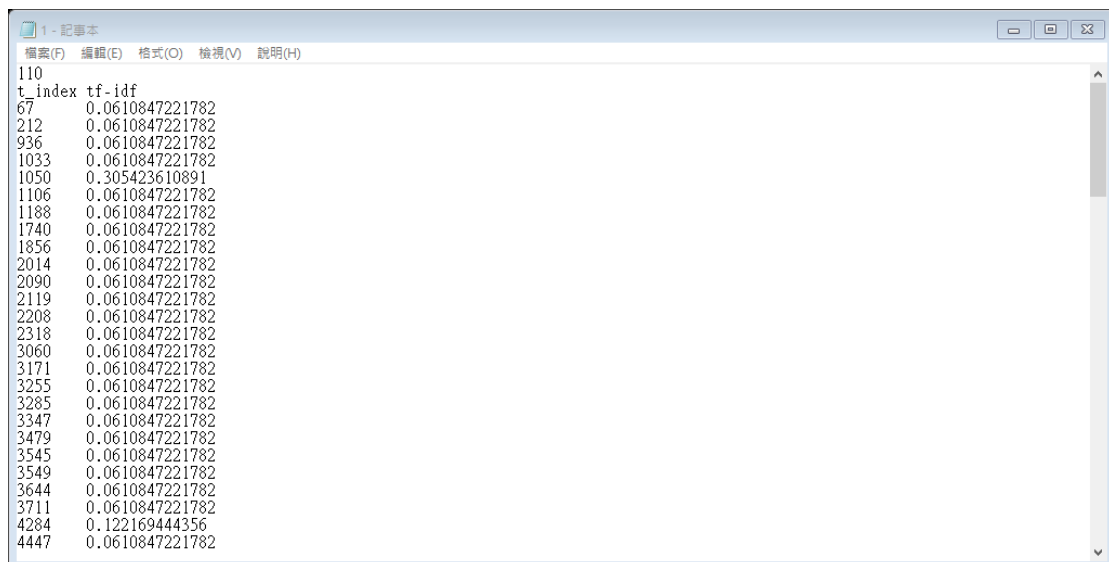


t_index	term	df
1	aan	1
2	aaron	3
3	aback	1
4	ababd	1
5	abandon	37
6	abat	1
7	abc	100
8	abccom	1
9	abcnewscom	3
10	abdallah	2
11	abdel	3
12	abdomin	2
13	abduct	4
14	abdul	5
15	abdulkarim	11
16	abdullah	49
17	abdurahman	1
18	aberr	1
19	abhad	1
20	abhyan	1
21	abhor	2
22	abid	8
23	abidjan	87
24	abijan	4
25	abil	36
26	abl	135
27	ablaz	3

輸出之 dictionary.txt

在 Vector Files 資料夾內會輸出各 txt 檔的 vector file，以 1.txt 檔為

例：



t_index	tf-idf
110	
67	0.0610847221782
212	0.0610847221782
936	0.0610847221782
1033	0.0610847221782
1050	0.305423610891
1106	0.0610847221782
1188	0.0610847221782
1740	0.0610847221782
1856	0.0610847221782
2014	0.0610847221782
2090	0.0610847221782
2119	0.0610847221782
2208	0.0610847221782
2318	0.0610847221782
3060	0.0610847221782
3171	0.0610847221782
3255	0.0610847221782
3285	0.0610847221782
3347	0.0610847221782
3479	0.0610847221782
3545	0.0610847221782
3549	0.0610847221782
3644	0.0610847221782
3711	0.0610847221782
4284	0.122169444356
4447	0.0610847221782

4. 作業處理邏輯說明

依序開啟 IRTM 資料夾內的*.txt 檔進行全小寫化、去跳行符號並依空

白字元進行初步分割後，使用正則表示式留下符合規則的 token，接

著使用 nltk 套件的 PorterStemmer，最後再藉由讀入 stopwords.txt 比對來去除 stopwords，將每個 term 整理為 dictionary 並計算 df，待全部處理完成後，依 term 排序輸出成 dictionary.txt 檔。接著，使用 scikit-learn 套件計算 tf-idf，並將其依照*.txt 命名規則在 Vector files 資料夾內輸出結果。最後，呼叫 function cosine 來計算 1.txt 與 2.txt 間的 cosine similarity。Function cosine 則是依照上課講義的邏輯，先使用 function read_file_to_list 分別將 1.txt 及 2.txt 轉為矩陣後進行計算，當 $x_t_index=y_t_index$ 時進行計算，不相等時則依情況將 $i+1$ 或 $j+1$ ，最後再 print 出。

5. 心得

此次作業延續 PA-1，整理成 dictionary.txt 還算容易，但後面計算 tf-idf 的部分則不是很簡單，要找到快速的方法還要理解過程是需要一些時間的，function cosine 則是依照講義和上課提到的邏輯進行即可。