Document for Programming Assignment 2

R06725016 吳亞璇
2017/10/23

程式語言

- Python 3.5.2

執行環境

- Ubuntu 16.04.3 LTS
  - LTS 為 Long Term Support

執行方式

- 安裝 Python

```
sudo apt-get install python3.5
```

- 檢查 Python 版本

```
$ python3 --version
Python 3.5.2  # 或任何 3.5.2 以上的版本
```

- 安裝虛擬環境（virtual environment）[註一]

```
sudo apt-get install python3-venv
```

- 建立虛擬環境（virtual environment）

```
python3 -m venv IRTM_venv
```

- 執行虛擬環境

```
$ source IRTM_venv/bin/activate
(IRTM_venv) ~$  # 無論所在路徑為何，(IRTM_venv)必會出現，表示你已經在虛擬環境中
```

- 假設檔案目錄如下

```
hw2
|___ __init__.py
|___ process_content.py (執行 terms 的產生)
|___ hw2.py             (此次作業的主程式)
|___ consine.py         (cosine(Doc_x, Doc_y)的實作)
|___ documents          (text collection，共1095個檔案)
```

- 切換至程式檔所在的目錄

```
~$ cd hw2
(IRTM_venv) ~/hw2$
```

- 安裝 Python 套件 — nltk [註二]

```
pip3 install nltk
```

- 下載 nltk 的 stopwords

```
$ python3
>>> import nltk
>>> nltk.download('stopwords')  # 等待下載完成
>>> exit()
```

- 安裝 Python 套件 — numpy [註三]

```
pip3 install numpy
```

- 執行程式

```
python3 hw2.py  # 輸出為 dictionary.txt 和 result/Doc1.txt
```

[註一] 使用虛擬環境有許多優點，包括：

- 讓專案有獨立的執行環境。當多人在不同機器上跑同一專案時，也能確保環境的一致性。
- 便於控管套件，避免升級套件時影響到其他專案的執行。

[註二] NLTK（Natural Language Toolkit）

- NLTK 是 Python 的自然語言處理套件，附帶不同程度的預先處理功能，例如：Tokenization
- 此次作業中我只用 NTLK 來實作 Porter's Stemming Algorithm

[註三] Numpy

- 是 Python 支援高階的維度陣列與矩陣運算的擴充程式庫
- 此次作業用它來實作 `cosine function`，運算速度會比用 `list` 實作快

文件處理邏輯說明（分為三個部份）

產生 Terms

- function 寫在 `process_content.py` 中，在 `hw2.py` 中 `import` 進來
- 拿掉 `http`, `www` 開頭的 string ，也拿掉標點符號、非英文字母的字元
- 以 Porter's Algorithm 做 stemming
- 拿掉 stop words
- 基本上和作業一相同，但是因為要標記文件，所以須以一樣的方式處理文件，不然文件會找不到它該有的 term

產生 Dictionary，計算 `df, tf, idf, tf-idf` 值

- Dictionary 的資料結構和 pseudo code 如下，產生 dictionary 的過程中一併計算出 `df` 和 `tf` 值

```
dictionary = [
  # term
  {
    "df": df_value,
    "tf": {
      doc_id: tf_value, ... # postings
    }
  }, ...
```

```
]
```

```
for term in term_list:
  for doc in doc_list:
      if term in doc:
        dictionary[term_index]["df"] += 1   # df值加一
        tf = doc.count(term)                # 計算該term在該doc的tf值
        dictionary[term_index]["tf"][str(doc_index)] = tf
```

- 計算 `idf` 值

```
import math
# doc_count為文件數量
idf_list = [math.log10(doc_count / term["df"]) for term in dictionary]
```

- 計算 `normalized tf-idf` 值

```
tf_idf_list = []
for tf in tf_list:
  get term_index
  tf_idf_list.append(tf * idf_list[term_index])

doc_length = 開根號(sum(每個tf_idf值的平方))
normalized_tf_idf_list = []
for tf_idf in tf_idf_list:
  normalized_tf_idf_list.append(tf_idf/doc_length)
```

- 寫檔
  - 目前的程式檔中只產生 `Doc1.txt`，可調整 `hw2.py` 中 `get_tf_related_values_and_write_file` 的參數產出所有檔案

實作 `cosine(Doc_x, Doc_y)`

- function 寫在 `cosine.py` 中，之後要用到時再 `import` 即可
- 讀檔，將文件轉成 `numpy.array`

```
import numpy as np
data_list = []
with open(文件路徑) as file:
  跳過第一行
  for line in file:
    data_list.append(normalized tf-idf值)
doc_vector = np.asarray(data_list)  # 將list轉成array
```

- 計算 cosine 值

```
def cosine(Doc_x, Doc_y):
```

```
get doc_x_vector, doc_y_vector
normalize doc_x_vector to normalized_doc_x  # 將vector的magnitude設為1
normalize doc_y_vector to normalized_doc_y  # 將vector的magnitude設為1
cosine_value = normalized_doc_x dot normalized_doc_y
```