Document for Programming Assignment 4

R06725016 吳亞璇
2018/01/08

程式語言

- Python 3.5.2

執行環境

- Python 3.5.2
- 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
```

- 切換至程式檔所在的目錄

```
~$ cd hw4
(IRTM_venv) ~/hw4$
```

- 此次作業檔案目錄如下

```
hw4
|___ documents          (text collection，共1095個檔案)
|___ data
     |___ stopwords.txt
     |--- dictionary.txt (在產生 doc vectors 時產生)
```

```
|___ cluster.py           (cluster as a class)
|___ priority_queue.py    (實作 heap)
|___ doc_processor.txt    (用以產生 doc vector)
|___ hw4.py               (作業主程式)
```

- 安裝 Python 套件 – nltk [註二]

```
pip3 install nltk
```

[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

Hierarchical Agglomerative Clustering 實作邏輯說明（similarity 以 centroid similarity 計算）

將 documents 轉換成 vector

- 和之前的作業一樣，先處理 1095 篇文章：
  - tokenization
  - stemming
  - remove stop words
  - 得到 terms
    - 產生 dictionary
- 利用 dictionary 和 documents 得到：
  - tf 值
  - df 值
    - idf 值
- 計算 tf-idf 值，以 normalized 的 tf-idf 值（vector）代表一份 document

Cluster

將 cluster 寫成一個 class，邏輯如下：

```
class Cluster:
  def __init__(self, parameters):
      初始化 cluster
  def merge(self, another cluster):
      更新此 cluster 包含的文件
      更新此 cluster 的大小
      更新此 cluster 的 centroid
      # centroid = (N1 * centroid1 + N2 * centroid2) / (N1 +N2)
  def get_min_similarity(self):
```

```
        回傳此 cluster 的 similarity heap 中的最小值
        和對應的 cluster index
```

Heap

利用 heap 資料結構來存取一個 cluster 對其他 cluster 的 similarity，效率較好

```
class Heap:
    def __init__(self):
        初始化 heapList
        初始化 heapSize
    def move_up(self, i):
        處於位置 i 的數值依序往上替換比它大的parents，
        使heap的架構維持正確，保持root為最小值
    def insert(self, k):
        將 k 置於 heap 末端
        move_up(heapSize)  # 從末端開始 move up
    def move_down(self, i):
        處於位置 i 的數值依序往下替換比它小的children，
        使heap的架構維持正確，保持root為最小值
    def delete(self, k):
        找到 k 所在的位置 p，刪除 k
        將 heap 末端值移到 p
        move_down(self, p) # 從 p 開始 move down
```

Hierarchical Agglomerative Clustering

- 初始化
  - 最初一篇文件即自成一個 cluster
  - 計算各個 cluster 與其他 cluster 的距離（相似度）
  - 建立每個 cluster 的 heap，以儲存距離（相似度）
  - 初始化 available list（用來紀錄某一 cluster 是否還能 merge）、merging history 等其他變數

- Clustering

```
for k from 1 to (1 - doc_num):
    k1 = 兩兩 cluster 中相距最短的一隊中 index 較小者
         (一律併入 index 較小的 cluster)
    k2 = 欲併入 k1 的 cluster index
    append merging history
    available list[k2] = 0
    清空 cluster k1 的 heap
    for 還能做合併的 cluster：
```

```
        heap delete 和 cluster k1, cluster k2 相關的距離值
        重新計算和 cluster k1 相關的距離值
        insert 和 cluster k1 相關的距離值
        insert 和該 cluster 相關的距離值到 cluster k1 的 heap
```