HW2 Report R05725060 郭欣宜

執行環境 & 作業系統

- 執行環境
  - terminal
- 作業系統
  - Mac OS

程式語言,版本

- 程式語言
  - 版本 python3

執行方式

- python3 hw2.py

1.txt 和 2.txt 的 cosine similarity

- 0.18240843397014356

作業處理邏輯說明

import

```
#!/usr/local/bin/python3
"""This script converts a set of documents into tf-idf vectors"""

import os
import operator
import math
import time

import hw1
```

1. import 必要的library
2. import 上次的作業1(hw1.py)

查看是否此檔案為直接被執行

```
142  if __name__ == "__main__":
143      main()
```

1. 如果是直接執行此檔案，就執行main function

main function

```
130  def main():
131      """main function"""
132      print(time.strftime("%Y/%m/%d %H:%M:%S"), ' ---Start!')
133      construct_dictionary('IRTM/')
134      dictionary_list = read_dictionary('dictionary.txt')
135      dictionary_of_index = dictionary_index(dictionary_list)
136      dictionary_of_df = dictionary_df(dictionary_list)
137      transfer_tfidf_unit_vector('IRTM/', dictionary_of_index, dictionary_of_df)
138      cosine_similarity = cosine('unit_vector/1.txt', 'unit_vector/2.txt')
139      print(cosine_similarity)
140      print(time.strftime("%Y/%m/%d %H:%M:%S"), ' ---Done!')
```

1. 建立在IRTM資料夾底下所有文件的字典
2. 把建好的dictionary 讀進來
3. 把index的dictionary建好 : dictionary_of_index['term'] = index
4. 把df的dictionary建好 : dictionary_of_index['term'] = df
5. 把IRTM裡面的所有檔案算出它的tf-idf的unit vector
6. 算出1.txt和 2.txt的 cosine similarity

construct_dictionary function

```python
def construct_dictionary(documents_dir):
    """construct dictionary"""
    now_path = os.path.dirname(os.path.abspath(__file__))+'/'
    documents = os.listdir(now_path + documents_dir)
    dictionary = {}
    for document in documents:
        token_list = hw1.preprocessing(documents_dir+document)
        pre_token = ''
        for token in token_list:
            if token == pre_token:
                continue
            if token in dictionary:
                dictionary[token] = dictionary[token] + 1
            else:
                dictionary[token] = 1
            pre_token = token
    dictionary = sorted(dictionary.items(), key=operator.itemgetter(0))
    write_dictionary(dictionary, './dictionary.txt')
```

1. 取得現在執行file的資料夾位置
2. 把所有此資料夾的檔案存進documents的list
3. 建立 dictionary -> 用意為term的df
4. 針對每一個檔案先進行hw1的preprocessiong後算出把term存進dictionary，之後算出所有檔案term的df
5. 把所有term按照字母順序排列(sorted)
6. 最後把這個dictionary term 的list存進dictionary.txt

write_dictionary function

```python
def write_dictionary(dictionary, filename):
    """write dictionary"""
    now_path = os.path.dirname(os.path.abspath(__file__))+'/'
    f_value = open(now_path + filename, 'w', encoding='UTF-8')
    f_value.write("t_index\tterm\tdf\n")
    for i in range(0, len(dictionary)):
        f_value.write(str(i+1)+'\t' + dictionary[i][0] + '\t' + str(dictionary[i][1]))
        if i != len(dictionary)-1:
            f_value.write('\n')
    f_value.close()
```

1. 寫dictoinary的header t_index[tab]term[tab]df
2. 把dictionary寫進dictionary.txt

read_dictionary function

```
41  def read_dictionary(filename):
42      """read dictionary from ./filename"""
43      dictionary = hw1.read_file(filename)
44      dictionary_list = dictionary.split('\n')
45      # remove header
46      del dictionary_list[0]
47      return dictionary_list
```

1. 把原本寫好的dictionary讀進來存在list

dictionary_index function

```
49  def dictionary_index(dictionary_list):
50      """construct a dictionary of index"""
51      dictionary_of_index = {}
52      for i in dictionary_list:
53          content_list = i.split('\t')
54          dictionary_of_index[content_list[1]] = content_list[0]
55      return dictionary_of_index
```

1. 建立 index 的 dictionary 格式為 dictionary_of_index['term'] = index

dictionary_df function

```
57  def dictionary_df(dictionary_list):
58      """construct a dictionary of df"""
59      dictionary_of_df = {}
60      for i in dictionary_list:
61          content_list = i.split('\t')
62          dictionary_of_df[content_list[1]] = content_list[2]
63      return dictionary_of_df
```

1. 建立 df 的 dictionary 格式為 dictionary_of_df['term'] = df

transfer_tfidf_unit_vector function

```python
def transfer_tfidf_unit_vector(documents_dir, dictionary_of_index, dictionary_of_df):
    """transfer tf-idf unit vector"""
    now_path = os.path.dirname(os.path.abspath(__file__))+'/'
    documents = os.listdir(now_path + documents_dir)
    for document in documents:
        dictionary = {}
        token_list = hw1.preprocessing(documents_dir+document)
        for token in token_list:
            if token in dictionary:
                dictionary[token] = dictionary[token] + 1
            else:
                dictionary[token] = 1
        dictionary = sorted(dictionary.items(), key=operator.itemgetter(0))
        #write to unit_vector/filename
        now_path = os.path.dirname(os.path.abspath(__file__))+'/'
        if not os.path.isdir(now_path+'unit_vector'):
            os.mkdir(now_path+'unit_vector')
        f_value = open(now_path + 'unit_vector/' + document, 'w', encoding='UTF-8')
        f_value.write(str(len(dictionary))+'\n')
        f_value.write('t_index\ttf-idf')
        tfidf_squares = 0.0
        for term in dictionary:
            tfidf_pow = math.pow(count_tfidf(len(documents), float(term[1]), float(dictionary_of_df[term[0]
            tfidf_squares = tfidf_squares + tfidf_pow
        for term in dictionary:
            f_value.write('\n')
            tf_idf = count_tfidf(len(documents), float(term[1]), float(dictionary_of_df[term[0]]))
            f_value.write(dictionary_of_index[term[0]] + '\t' + str(tf_idf/math.sqrt(tfidf_squares)))
        f_value.close()
```

1. 把IRTM裡面的所有檔案算出它的tf-idf的unit vector
2. 算出每個檔案term的tf
3. 把term的dictionary以字母大小排列(sorted)
4. 如果沒有unit_vector的資料夾就建立一個
5. 把檔案中擁有的term數量寫在document的第一行
6. 寫document的標頭 t_index[tab]tf-idf
7. 接下來是把算出的tf-idf值變成unit vector
8. 先把所有tf-idf平方得出的值加總
9. 再把每一個tf-idf的值除以剛剛加總得值就是最後要寫進document的tf-idf

count_tfidf function

```python
def count_tfidf(documents_num, tf, df):
    """count term tf-idf"""
    idf = math.log10(documents_num/df)
    tf_idf = tf * idf
    return tf_idf

```

1. 算出給定的doument的tf-idf

cosine function

```
def cosine(doc_x, doc_y):
    """count the cosine similarity of two documents"""
    doc_x_content = hw1.read_file(doc_x)
    doc_y_content = hw1.read_file(doc_y)
    doc_x_content = doc_x_content.split('\n')
    doc_y_content = doc_y_content.split('\n')
    del doc_x_content[0]
    del doc_x_content[0]
    del doc_y_content[0]
    del doc_y_content[0]
    doc_x_content = [term.split('\t') for term in doc_x_content]
    doc_y_content = [term.split('\t') for term in doc_y_content]
    cosine_similarity = 0.0
    i = 0
    j = 0
    while i < len(doc_x_content) and j < len(doc_y_content):
        if int(doc_x_content[i][0]) < int(doc_y_content[j][0]):
            i = i + 1
            continue
        if int(doc_x_content[i][0]) > int(doc_y_content[j][0]):
            j = j + 1
            continue
        if int(doc_x_content[i][0]) == int(doc_y_content[j][0]):
            cosine_similarity = cosine_similarity + float(doc_x_content[i][1]) * float(doc_y_content[j][1])
            i = i + 1
            j = j + 1
            continue
    return cosine_similarity
```

1. 把要算cosine similarity的兩個檔案讀進來
2. 刪除term的數量和標頭
3. 以tab作為分割符號
4. 因為兩個檔案都有sorted過
5. 所以比對兩個檔案的index
6. 看index哪個比較小就加那個index的值
7. 兩個檔案中所有相等的index相乘各自的unit vector tf-idf就為最後要得到的cosine similarity

作業的心得

這次的作業讓我體會到如果資料量一大，算的速率跟程式寫的好壞有相當大的關聯性，像python的count我覺得最好不要用，而且要善用dictionary的特性，找term會快很多。