資料探勘-關聯法則

簡要說明

- 使用python實作Apiori暴力法、Apiori使用hash tree搜尋support值以及使用FP-Growth方法直接找出frequent itemset，並且由frequent itemset生成關聯規則。
- 共實作於3種data，分別為上課投影片所舉例比較簡單的transaction、以IBM Quest Synthetic Data Generator產生之資料集以及Kaggle平台上公開提供之資料集。
- 暴力法為逐筆transaction循序搜尋下去，如果比對candidate成功便會將其support值加一，這是裡面最緩慢的方法。
- Hash tree，其實作的hash function為h(x) = x % 5，其中x為itemset當中所有數字相加，若item為英文字串，則將字串中所有字母轉為ascii code後加總。每個節點都可以有5棵子樹，因此搜尋candidate時便可以透過hash function的配對排除其他不必要的搜尋。
- FP-Growth為三種方法裡面最為迅速的方法，不論minimum support設定多低，都有不錯的表現，迅速地找出Frequent itemset。

檔案架構

- `dataset/Test.csv`: 本次實驗輸入資料，為驗證本程式與Weka輸出結果一致之測試使用
- `dataset/IBM-Quest-Data-Generator.exe/ttt.data.txt`: 本次實驗輸入資料，是由IBM Quest Synthetic Data Generator產生之資料
- `dataset/BreadBasket_DMS.csv`: 本次實驗輸入資料，是由Kaggle平台上公開提供之資料
- `fpTree.py`: FP-Tree物件實作、建立FP-Tree、以FP-Growth產出frequent itemset
- `apriori.py`: 產出L1與Cn，並以暴力法計算每個candidate set的support值來產出frequent itemset
- `apriori_byTree.py`: hash tree物件實作、建立hash tree，並計算每個candidate set的support值來產出frequent itemset
- `DataReader.py`: 讀取輸入檔案，轉換為dataframe形式
- `generateRule.py`:從frequent itemset產出rule
- `main.py`: 主程式，整合3種資料集*3種方法，共9種輸出結果

程式執行

以kaggle資料集，使用hashtree實作apiori方法為例，並設定minimum support為 300 instance、minimum confidence為0.5。

- 輸入：選擇資料集與實作法所對應的函式，輸入參數minimum support與minimum confidence

```
[11] starttime = datetime.datetime.now()
     kaggle_Apiori_hashtree(300,0.5)
     endtime = datetime.datetime.now()
     print (endtime - starttime)
```

- 輸出：(包含frequent itemset , rules, 執行時間)

```
freq itemsets:
871 ['Bread', 'Coffee']
308 ['Coffee', 'Hot chocolate']
327 ['Coffee', 'Cookies']
492 ['Coffee', 'Pastry']
336 ['Coffee', 'Medialuna']
476 ['Coffee', 'Tea']
610 ['Cake', 'Coffee']
466 ['Coffee', 'Sandwich']
rule: ['Hot chocolate'] --> ['Coffee']
support: 308 confidence: 0.5422535211267606

rule: ['Cookies'] --> ['Coffee']
support: 327 confidence: 0.6252390057361377

rule: ['Pastry'] --> ['Coffee']
support: 492 confidence: 0.5934861278648975

rule: ['Medialuna'] --> ['Coffee']
support: 336 confidence: 0.5637583892617449

rule: ['Cake'] --> ['Coffee']
support: 610 confidence: 0.6118355065195586

rule: ['Sandwich'] --> ['Coffee']
support: 466 confidence: 0.6472222222222223

0:00:15.679425
```

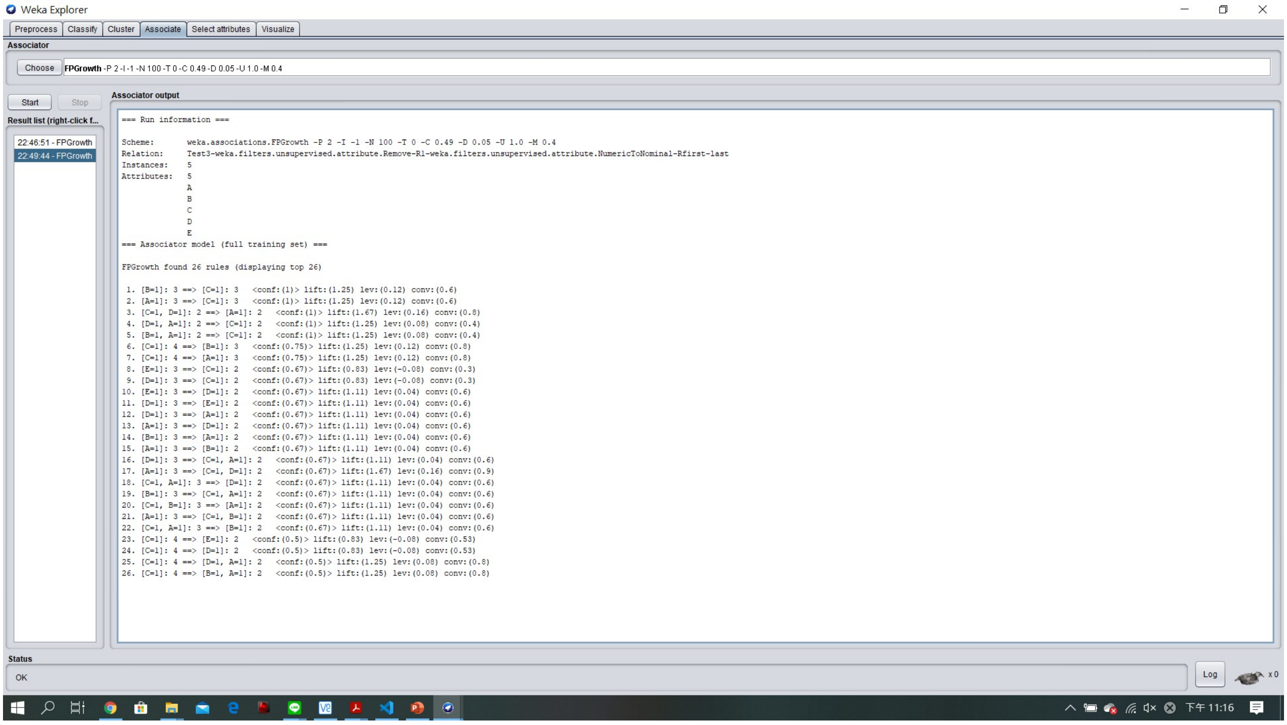
驗證

驗證1. 輸出規則正確

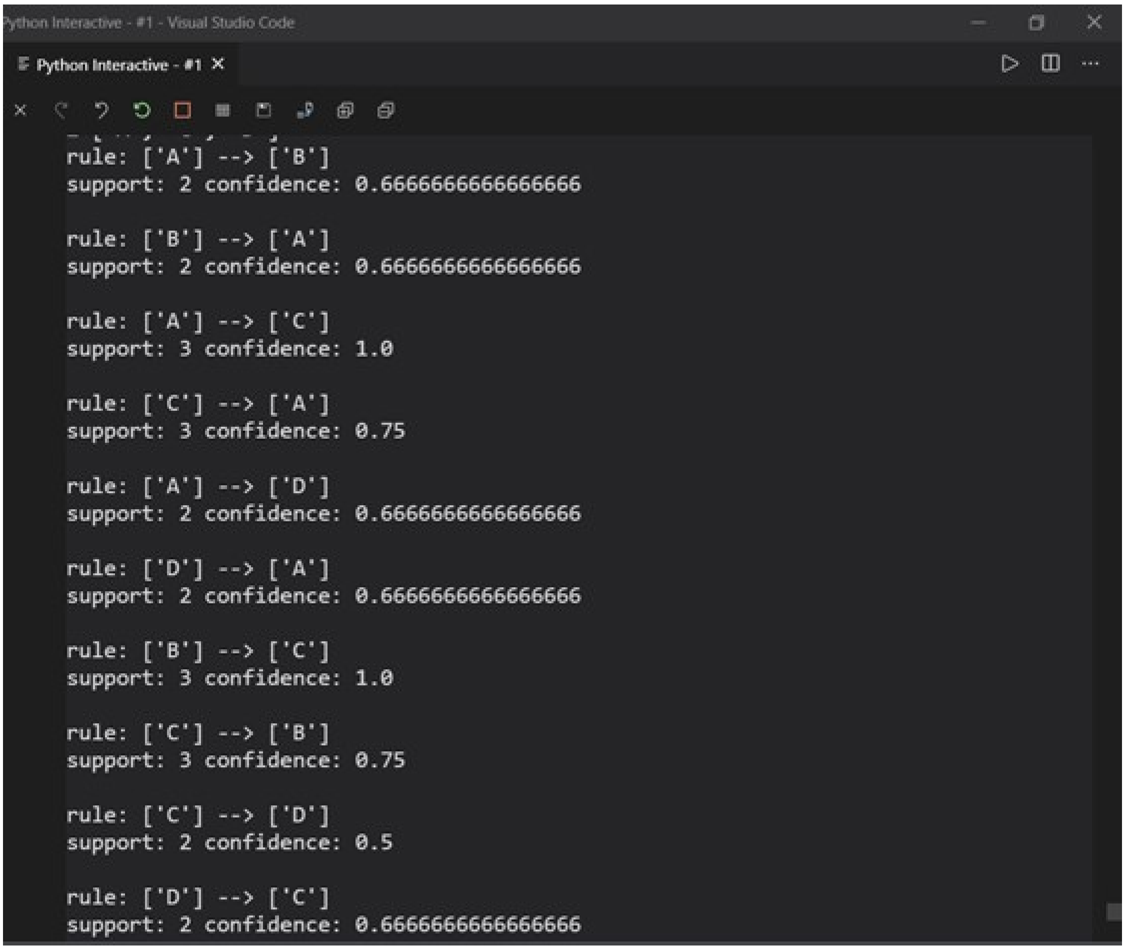
由於在大量數據的association rules並不會依照固定順序排列，再比對上有些難度，所以就採用上課投影片比較簡單的transaction，限制minimum support為0.4（2 instance）、minimum confidence為0.5，使用Weka產生association rules，再比對這次作業程式輸出的association rules，確認結果一致。

TID	Items
1	ABC
2	ABCD
3	BCE
4	ACDE
5	DE

- Weka輸出結果



- 本程式輸出結果



```
rule: ['C'] --> ['E']
support: 2 confidence: 0.5

rule: ['E'] --> ['C']
support: 2 confidence: 0.6666666666666666

rule: ['D'] --> ['E']
support: 2 confidence: 0.6666666666666666

rule: ['E'] --> ['D']
support: 2 confidence: 0.6666666666666666

rule: ['A'] --> ['B', 'C']
support: 2 confidence: 0.6666666666666666

rule: ['B'] --> ['A', 'C']
support: 2 confidence: 0.6666666666666666

rule: ['C'] --> ['A', 'B']
support: 2 confidence: 0.5

rule: ['A', 'B'] --> ['C']
support: 2 confidence: 1.0

rule: ['A', 'C'] --> ['B']
support: 2 confidence: 0.6666666666666666

rule: ['B', 'C'] --> ['A']
support: 2 confidence: 0.6666666666666666
```

```
rule: ['A'] --> ['C', 'D']
support: 2 confidence: 0.6666666666666666

rule: ['C'] --> ['A', 'D']
support: 2 confidence: 0.5

rule: ['D'] --> ['A', 'C']
support: 2 confidence: 0.6666666666666666

rule: ['A', 'C'] --> ['D']
support: 2 confidence: 0.6666666666666666

rule: ['A', 'D'] --> ['C']
support: 2 confidence: 1.0

rule: ['C', 'D'] --> ['A']
support: 2 confidence: 1.0

[35] Type code here and press shift-enter to run
```

驗證2. 確認3種方法產出frequent itemset之結果相同

由於從frequent itemset產出關連規則是使用相同函式(於驗證1已驗證過)，且規則數量較多在比對上有困難，因此只比對frequent itemset是否相同。
使用ibm資料集，限制minimum support的instance為5，確認3種方法所產出之frequent itemset相同。

- Apiori暴力法

```
3
6 ['63977', '83398']
6 ['23641', '63977']
6 ['37407', '63977']
6 ['23641', '83398']
6 ['37407', '83398']
7 ['36038', '52972']
6 ['23641', '37407']
6 ['23641', '63977', '83398']
6 ['37407', '63977', '83398']
6 ['23641', '37407', '63977']
6 ['23641', '37407', '83398']
6 ['23641', '37407', '63977', '83398']
rule: ['63977'] --> ['83398']
```

- Apiori使用hash tree

```
3
6 ['63977', '83398']
6 ['23641', '63977']
6 ['37407', '63977']
6 ['23641', '83398']
6 ['37407', '83398']
7 ['36038', '52972']
6 ['23641', '37407']
6 ['23641', '63977', '83398']
6 ['37407', '63977', '83398']
6 ['23641', '37407', '63977']
6 ['23641', '37407', '83398']
6 ['23641', '37407', '63977', '83398']
0:00:22.528119
```

- FP-Growth

	count	0	1	2	3
0	7	36038	52972	None	None
1	6	63977	83398	None	None
2	6	37407	63977	None	None
3	6	37407	83398	None	None
4	6	37407	63977	83398	None
5	6	23641	63977	None	None
6	6	23641	83398	None	None
7	6	23641	37407	None	None
8	6	23641	63977	83398	None
9	6	23641	37407	63977	None
10	6	23641	37407	83398	None
11	6	23641	37407	63977	83398

執行時間比較

- IBM data (./dataset/IBM-Quest-Data-Generator.exe/ttt.data.txt)
    - Basic data

Condition	
Number of items	10877
Number of transactions	1069
Max number of item	29

- Execution time

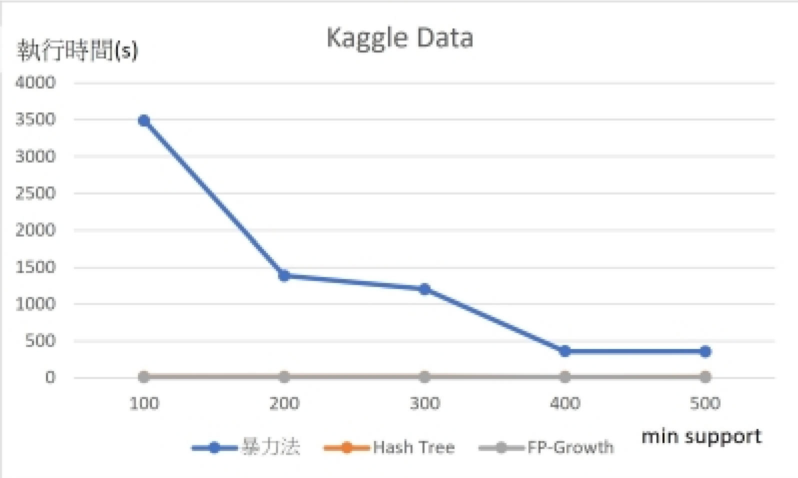
min support	4	5	6	7
暴力法	3:30:56.213	0:12.42.297	0:01:42.739	0:00:27.749
Hash Tree	0:15:31.734	0:00:17.721	0:00:13.553	0:00:10.75
FP-Growth	0:00:15.282	0:00:10.476	0:00:09.769	0:00:09.355

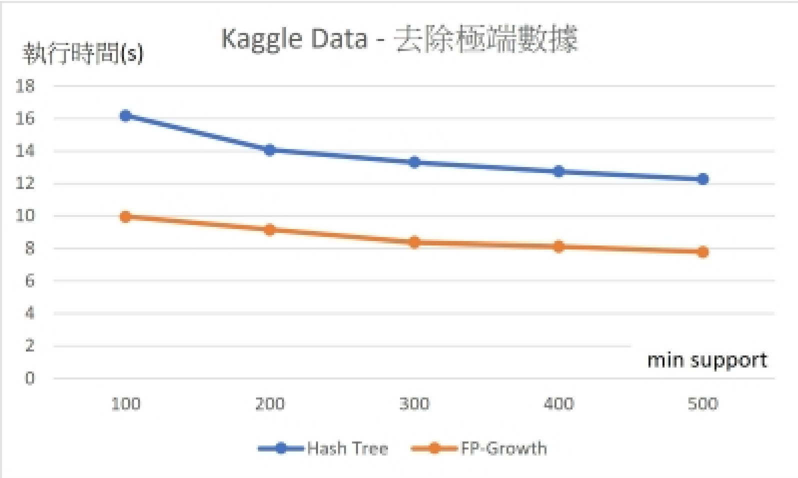
- Kaggle data (./dataset/BreadBasket_DMS.csv)
  - Basic data

Condition	
Number of items	21294
Number of transactions	9531
Max number of item	12

- Execution time

min support	100	200	300	400	500
暴力法	0:58:09.205	0:23:10.025	0:20:02.641	0:00:59.406	0:00:58.123
Hash Tree	0:00:16.186	0:00:14.071	0:00:13.3	0:00:12.744	0:00:12.256
FP-Growth	0:00:09.957	0:00:09.153	0:00:08.382	0:00:08.135	0:00:07.797





結論

- 本實驗是以IBM Quest Synthetic Data Generator所產生的資料及Kaggle上開放的dataset來作程式運行效能的評估。
- 隨著minimum support的下降，可以見到程式運行時間會越來越大幅度成長，尤其是暴力法最為明顯，必須多次查看所有data，雖然coding想法容易，但只要資料量大或是minimum support低的狀況下，並不是一個好的做法。反觀FP-Growth，因為是使用全部的transaction來建構FP tree，只需掃過一次data, 因此對於低minimum support的環境下影響不大，比較下來也是這之中最好的做法。

Author

成大工科 大四 李姵萱

完整程式可到github下載