生科大四 C54046238 李承學

一、 執行環境

1. 作業系統：Ubuntu 18.04
2. Python 版本：Python 3.6.6

二、 程式檔案

1. bruteforce.py

   利用暴力法搜尋 frequent itemset

2. apriori.py

   利用 Apriori 搜尋 frequent itemset

3. apriori_hashtree.py

   利用 Apriori 搜尋 frequent itemset，並利用 hash tree 來計算 candidate 出現的次數

4. fp.py

   利用 FP growth 搜尋 frequent itemset

以上程式執行要加參數，參數的格式皆為

python3 bruteforce.py [-t threshold] file

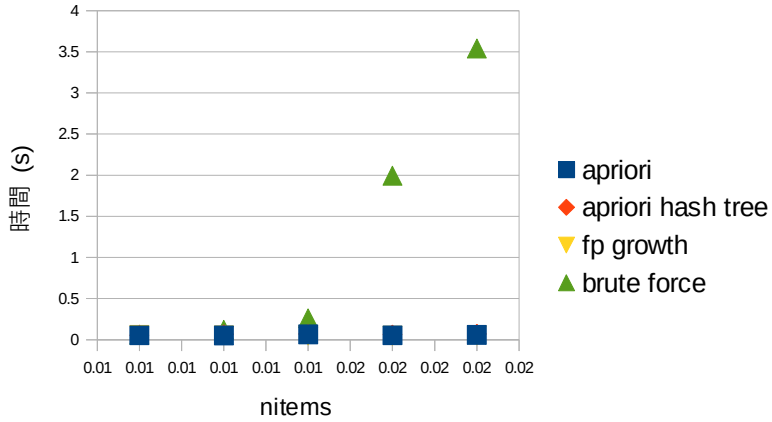
file 為輸入的檔案，格式為 csv 檔，以下有資料的說明，threshold 是 0 - 100 之間的一個浮點數，代表 minima support 設定的%數，例如有 1000 個 transactions，設定 5 代表 minimum support 為 1000 的 5%，即為 50。在未設定這項參數下的 threshold 預設為 10%。執行結果會直接印在螢幕上，若要存成檔案，要用>重導向至檔案

三、 Generator 資料

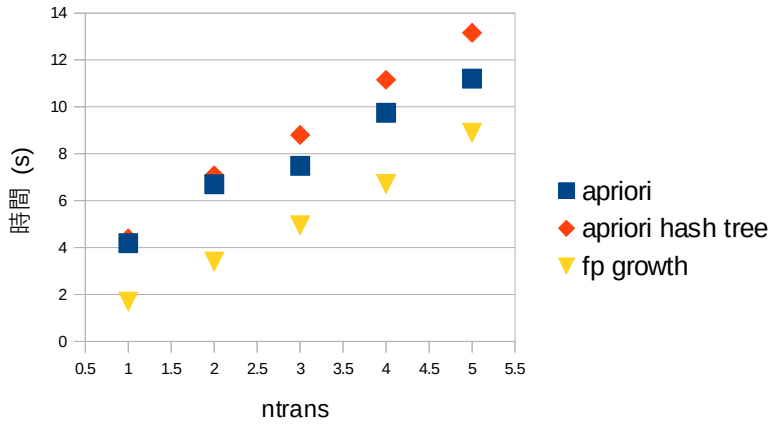
資料位於 data/generator/，所有資料都是由 IBM quest data generator 產生，並且整理為 csv 檔，檔名即為產生資料時所設定的參數。檔案中每個 row 就是一筆 transaction，這些檔案可以直接作為上述 python 程式的參數

四、 結果

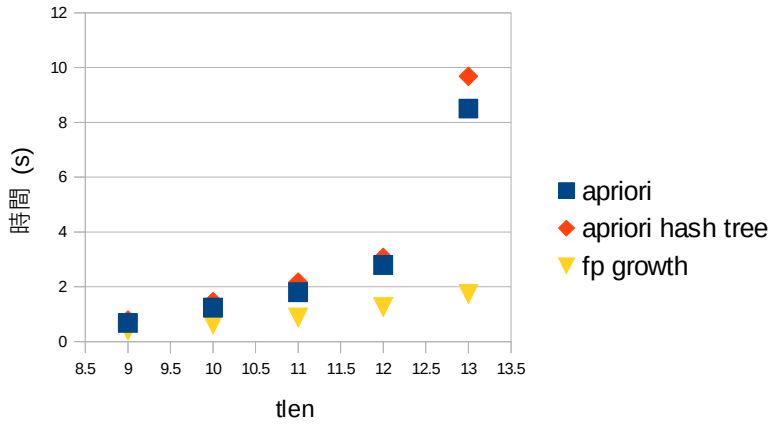
1. 測試小筆資料所花費的時間。測試資料位於 data/generator/small_data/，threshold 使用預設值 10%。從圖中可以觀察到 Apriori 和 FP growth 幾乎瞬間就完成，但是暴力法卻花了很多時間，而且隨著 item 種類增加，所花費的時間快速成長。資料位於 data/generator/nitems。因為暴力法花費太多時間，所以接下來的測試都不測暴力法



2. 測試不同 transaction 數量所花費的時間。測試資料位於 data/generator/ntrans/，threshold 使用預設值 10%。三種方法所花費的時間都隨著 transaction 增加而大致上呈線性上升

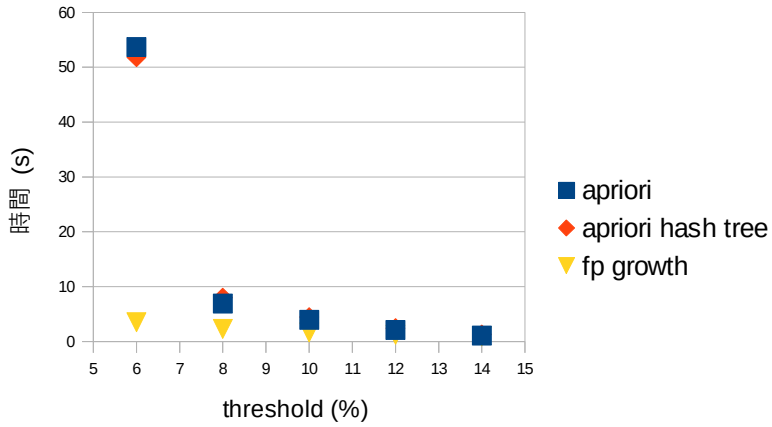


3. 測試每筆 transaction 的平均長度如何影響執行時間。測試資料位於 data/generator/tlen/，threshold 使用預設值 10%。從圖中可以看到 Apriori 的執行時間隨著平均長度的增加而快速成長，FP growth 則成長較為緩慢



4. 測試不同 threshold 的執行時間。測試資料為 data/generator/data.ntrans_1.tlen_15.nitems_0.1.csv，並將 threshold 設定為 6%～14%。從圖中可以

看到當 threshold 降低時，Apriori 執行所花費的時間會快速上升，而 FP growth 則上升較為緩慢



五、 結論

1. FP growth 相較於 Apriori 而言是更有效率的演算法，例如當 transaction 平均長度增加，或是 threshold 下降時，FP growth 所花費的時間上升都較緩慢
2. 雖然 Apriori 用 hash tree 可以增加比對 candidate 的效率，但是依照實際測試的結果，是否使用 hash tree 差異並不明顯，甚至 hash tree 有時還會增加執行時間

六、 Kaggle 資料

資料下載於 https://www.kaggle.com/xvivancos/transactions-from-a-bakery，並且已經整理為上述 python 程式可讀取的 csv 檔，位於 data/kaggle/bakery.csv。分別使用 Apriori、Apriori hash tree、FP growth 來做搜尋，設定 threshold 為 0.2%。執行結果為 kaggle_result/底下的三個檔案，檔案內容完全相同，並且依照 support count 進行排序