**Project FP growth && Apriori**

* **Implementation description**

1. **暴力法:**

找出所有可能的候選 patterns，將找出的 patterns 和全部 datasets 進行比比較並且統計數量，如果dataset的计数大於 min support，則這個 pattern 是合法的的frequent itemset。

1. **Apriori:**

首先先找出所有符合 min support 的 frequent sets (L1)後，用L1 items 兩兩組合出 長度為2的 candidate sets，與datasets對比得到 frequent sets （L2），同樣的方法得到L3，L4… ，直到候選set 超過 item 的數量後停止。

1. **Fp growth:**

FP-growth演算法只需要對資料庫進行兩次遍歷。

建立FP tree，從所有datasets中得到滿足大於min support的 frequent sets (L1)，按數量多少排序，把每個item中的資料項目按降冪依次插入到一棵以 NULL為根節點的樹中，同時在每個結點處記錄該結點出現的support。

Mining：首先從單元素項集合開始，然後在此基礎上逐步構建更大的集合。從FP樹中獲取prefix path，將首碼路徑轉化為conditional FP-tree。對於每一個frequent set，都要創建一棵conditional FP tree，conditional FP tree的結構與FP tree 相似。先對單個元素構建conditional FP tree（即刪除首碼路徑中支援度計數小於min support的tree），再對剩下的元素與單個元素兩兩組合構建新的conditional FP tree，遞迴直至conditional FP tree為空。

* **results comparison and discussion**

dataset：

IBM data 01：-ntrans 0.1 -tlen 10 1000筆數據 事物數為10

IBM data 02：-ntrans 1 -tlen 10 10000筆數據 事物數為10

IBM data 03：-ntrans 10 -tlen 10 100000筆數據 事物數為10

Kaggle data: 統計某大型公司各工作人員所用的編程語言，選取其中一部分轉換成IBM生成的格式約11000筆左右，統計了該公司10種目前階段最受歡迎的編程語言。

各方法所花費的時間(s)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Datasets（1000筆） | Datasets（10000筆） | Datasets（100000筆） | Kaggle data（11000筆） |
| 暴力法 | 0.254 | 1.144 | - | 1.150 |
| Apriori | 0.027 | 0.039 | 23.538 | 0.052 |
| FP GROWTH | 0.008 | 0.026 | 0.09 | 0.048 |

1. **資料規模大小**

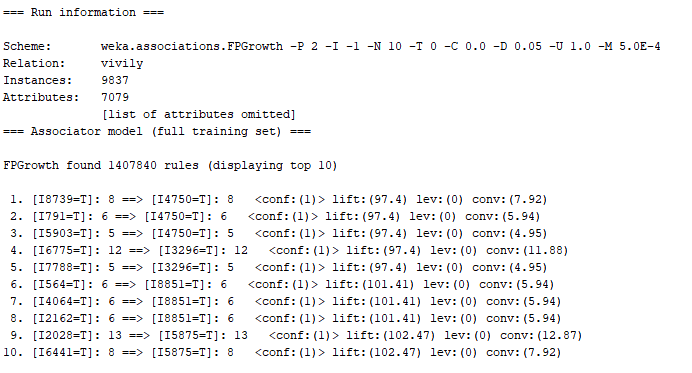
暴力法在運行中占記憶體大，且所耗的時間也最久。當資料規模越大時，使用暴力法的效率最低，在資料量到100000筆時，內存佔用接近95%但是跑不出結果；Apriori效率比FP growth低一點，因為Apriori演算法需要多次掃描資料庫，當資料量越大時，掃描資料庫帶來的消耗越多。

FP-Growth演算法只要兩次掃描資料庫，有效降低搜索時間，佔用記憶體較少。

### 最小支持度大小

當min support越小，各個演算法所花費的時間越多且暴力法演算法效率最差，佔用記憶體最多。當data>100000後開始記憶體爆炸。Min support越小，導致合法的frequent itemset越多，在mining的時候的反覆運算會更多。Apriori和fp growth在同樣數量級的資料運行時，佔用內存相差不多。

* **Weka**



以上是Weka 跑100000筆IBM datasets的 FP-Growth 的結果，耗時大概半分鐘多，同樣的資料用Apriori運行不能跑出結果，可能是筆電記憶體不夠。。。

* **conclusion and observation**

本次作業用了2種數據和3種不同的方法來mining data。

暴力法寫起來最簡單，但是運行速度最慢，效率最低。Apriori是逐层搜索的迭代方法，然而也可以看到Apriori算法的缺点： 在每一步产生侯选项目集时循环产生的组合过多，没有排除不应该参与组合的元素; 每次计算项集的支持度时，都对全部记录进行了一遍扫描比较，如果是一个大型的数据库的话，这种扫描比较会大大增加计算机系统的I/O开销。

相比起來，FPGrowth算法则只需扫描原始数据两遍，数据结构对原始数据进行压缩，效率较高。而且根據weka的結果，fp growth 比起apriori更能處理數量級較大的資料。

從data來看，使用fake data產生的結果看起來不是很make sense，還是看kaggle data的結果能反應出一些有趣的結論。如下圖所示，我們能得出在此外企公司，員工用C和C#的還是最多的，然後用了C#就很可能會用Ruby之類的結論。

****