**Project Ⅰ**

**Association Rule**

**Name : 黃柏喻**

**Student ID: N16064674**

**Department: Mechanical Engineering**

1. **使用方法與概念解析**
2. **暴力法**

暴力法採用的是將所有可能的candidate patterns 全部盡數列出，接著將所有列出的patterns 全部丟進dataset 進行比對並統計數量，當數量大於minimum support (通常是佔該資料庫大小的一定比例) 時，該pattern 便是我們要找的高頻特徵組合。

1. **Apriori Method**

Apriori 的方法是先找出所有符合minimum support 的one-item frequent set (L1)後，利用這些item 組合出two-item candidate set，經檢查數量後得出 L2，之後的步驟便以此類推，直到超出candidate set 超過item的數量後才停止。

1. **Fp growth**

此方法利用的是建構”樹”(Fp tree)來快速搜索的方法，第一次從所有案例中抓出one-item frequent set (L1)的時候，除了要滿足minimum support 之外，還要照著數量由大到小排列，接著將此順序實行在資料庫的所有案例上，除了刪除不滿足support 的items外，順序化的資料庫對後續建構Fp tree(由上往下)也相當重要。像是找到conditional pattern bases 後要建構conditional FP tree就是要依照此原則進而找出frequent patterns。

演算法實做上，是從最後mining 的演算法開始思考，用for 迴圈包著if 的遞迴函數，採用的想法是由下往上找父節點，因為子結點可能有很多個不過父節點只會有一個。因此在node的類別中，需要有子結點的集合還要有父節點的地址，接著就是利用整個tree中，包含相同item的node中可以不斷往右邊鏈結，也就是singular link list 的概念。

1. **結果比較與討論**

測試的dataset包括

* IBM trans = 100 & item = 10…(a)
* IBM trans = 1000 & item = 10…(b)
* IBM trans = 10000 & item = 100…(c)
* UCI Absenteeism at work…(d)

所有的運行結果都在Github的Report資料夾內(HTML形式)。

關於UCI dataset 的處理是採用以下的方式:   
(Github上的Readme.md 也有解說)

由於這份資料的項目有點煩雜且參差不齊，所以有些特徵並不考慮，然後主要抓出了以下幾點特徵並轉成數值化的項目模式。  
1-- Absenteeism time in hours > 平均值  
2-- Monday  
3-- Tuesday  
4-- Wednesday  
5-- Thursday  
6-- Friday  
7-- Distance from residence to working > 中位數  
8-- Disciplinary failure == 1 (Yes)  
9-- Education == 1 (high school)  
10-- Social drinker == 1 (Yes)

其中2,3,4,5,6一個案例中只會有一個數字出現，也就是該員工的請假日期。最後的資料形態變為 740 個工作缺席按例，然後有 10 項items。

Table 2.1 各資料庫與方法所花費時間(sec)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | a | b | c | d |
| 暴力法 | 0.13 | 0.295 | X | 1.141 |
| Apriori | 0.009 | 0.029 | 0.198 | 0.003 |
| Fp growth | 0.007 | 0.01 | 0.07 | ~=0.0 |

註: X 表示暴力法造成記憶體無法負荷

在使用暴力法、Aprioir、Fp growth 三種方法後，發現了以下幾點特徵。

* 暴力法在運行中最佔記憶體，且所耗的時間也最久。
* 在同樣的dataset、minimum support條件下，Fp growth可以說是最快的方法。

後來在 (c) 資料庫中，我把minimum support 往下調到0.1後，分別再使用兩個方法跑一次，圖2.1中可以發現Apriori 方法花了0.72秒左右，但是圖2.2中，Fp growth 反而花了將近2.6秒，與前面的結論相反。可能的原因是minimum support = 0.1時，符合條件的set數量會激增，而也因此需要花更多時間進行tree的建構，導致time complexity比Apriori直接進行數量上的比較來的高。



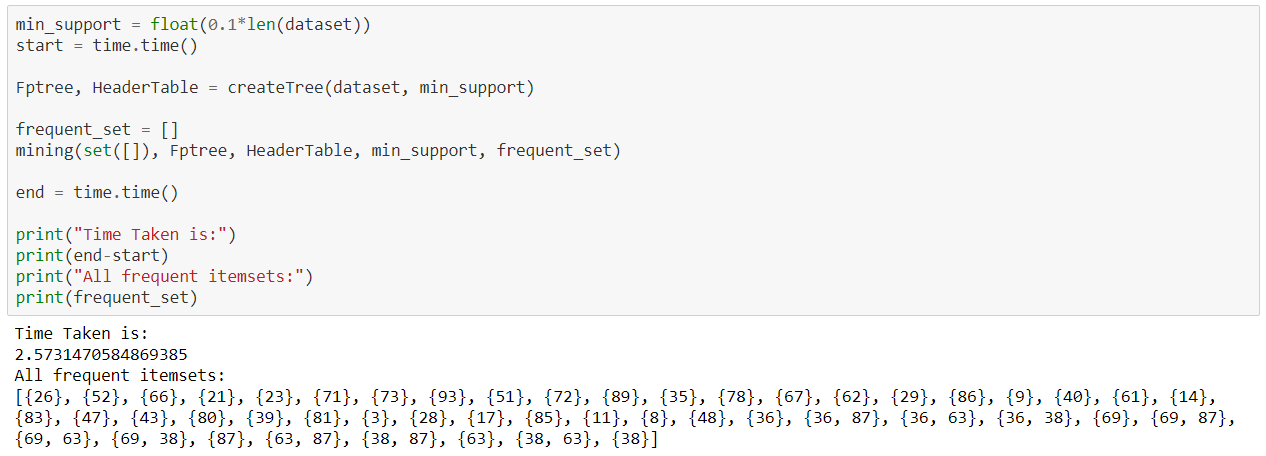
Figure 2.1 Apriori method with minimum support = 0.1

Figure 2.2 Fp growth method with minimum support = 0.1

1. **心得**

三種方法在coding難度上各有不同，暴力解好理解但是非常吃資源，Apriori的優點是讓前面產出的結果去篩選出後面的frequent itemset，整體的space complexity相對低很多，Fp growth則是tree的特性由高到低、由上而下進行排列、建構，優點是有同樣特徵的分枝可以用計數的方式省掉很多空間，在找尋鏈結上也相當迅速。三個方法大致上呈現的趨勢是行數愈多、效率愈高。

再來就是資料的前處理，IBM data generator產生的資料庫有點紊亂，python是非常適合用來進行資料處理的語言，所以我採用的是pandas套件裡的Dataframe 資料型態，另外就是UCI的dataset，這個資料檔裡面有一些我認為不會與預測目標有直接關係的參數，我把這些刪除後並把剩餘許多中文資料數值化以便進行與”工作缺席”相關的參數分析，因此，我認為把資料整理成可用性高的形態也是資料科學家的一大工作。

由於本身是機械系，有一些資工的課程miss掉，像是data structure、algorithm等，看到tree時並沒有立即與recursive 連結上，導致花了很多時間在使用iterative上，行數曾一度衝上250行但是仍失敗，後來看了一些線上課程和弄懂了網路上的解法後才慢慢增加了一些概念，逐一的trace 後才自己實做出來。

此次學到最多的東西就是寫一個好的演算法該有的步驟。首先，除了要了解該方法的核心概念並手動解過一個較小的資料庫，第二步，開始計畫需要用甚麼方式寫，包括資料型態、類別變數、函式功能，先用pseudo code 寫一遍在紙上後，第三步才有可能順利的打完整個程式，不然中間會有很高的機會要砍掉重練，當然在嘗試過後失敗了，就有參考了很多網路上的程式碼，再進行逐一的trace 還有修改。