# 國立成功大學

Data Mining 資料探勘

Project 1

Association Analysis

姓名：金雅倫

學號：P96074105

# 簡介< Introduction >

## (一)概述

### ✧ 使用之資料集：

找尋： Kaggle 網站上與關聯法則分析之相關資料

使用： IBM 資料生成器產生不同的資料集(資料參數個人自行設定)

實作：1. Apriori Algorithm

2. FP-growth

工具：WEKA 關聯分析工具 (將專案之資料集利用 WEKA 工具進行關聯法則之分析)、IBM-Quest-Data-Generator、Excel

## (二)Dataset

### 1. Kaggle 資料：

✧ 資料來源：https://www.kaggle.com/avenn98/world-of-warcraft-demographics

World of Warcraft 魔獸世界，縮寫作 WoW。

使用魔獸世界在 2017 年統計之玩家資料，尋找玩家之性別、居住地、使用之 Server、參與陣營、擔任腳色、職業及腳色種族之間之關係。

### 2. IBM 資料集：

將 IBM 所產生之資料應用在購物上，以-ntrans 設定為交易筆數；

-tlen 為每筆交易平均物品數量；-nitems 為交易之全部物品種類進行參數的設定，以供找到不同參數中資料數量間彼此之關聯規則。

# 實作< Implementation >

## ✧ 實作環境：

作業系統：Window 10

處理器：intel i5-4200H CPU @2.80GHz

記憶體：16GB

使用的程式語言：Python

搭配的額外分析工具：WEKA

## (一)Apriori

### ✧ 步驟：

1. 每個選項都是 1-itemset，對每個選項計數生成 C1，再把不滿足 min support 之值刪除，獲得 frequent 1-itemset：L1。
2. 對 L1 連接所生成的集合為：C2，對每個 C2 進行計數，刪除不滿足 min support 之值，獲得 frequent 2-itemset：L1。
3. 以此類推，對 Lk-1 連結所產生的 K 項集合：Ck，對每個 Ck 進行計數，根據 min support 刪除不滿足之項，獲得 frequent k-itemset。

### ✧ 程式部分內容：

```
def load_data_set():
    with open(r'C:\Users\ellen\Desktop\WoW Demographics.csv', newline='') as csv

      rows = csv.reader(csvfile)
      data_set = []
      for row in rows:
          data_set.append (row)
    return data_set


def create_C1(data_set):
    C1 = set()
    for t in data_set:
        for item in t:
            item_set = frozenset([item])
            C1.add(item_set)
    return C1


def is_apriori(Ck_item, Lksub1):
    for item in Ck_item:
        sub_Ck = Ck_item - frozenset([item])
        if sub_Ck not in Lksub1:
            return False
    return True
```

讀取 CSV 之檔案內容
以迴圈方式輸出每一列

找出 frequent candidate 建立 1-itemset：C1

判斷 itemset 是否滿足 Apriori property

```
def create_Ck(Lksub1, k):
    Ck = set()
    len_Lksub1 = len(Lksub1)
    list_Lksub1 = list(Lksub1)
    for i in range(len_Lksub1):
        for j in range(1, len_Lksub1):
            l1 = list(list_Lksub1[i])
            l2 = list(list_Lksub1[j])
            l1.sort()
            l2.sort()
            if l1[0:k-2] == l2[0:k-2]:
                Ck_item = list_Lksub1[i] | list_Lksub1[j]
                if is_apriori(Ck_item, Lksub1):
                    Ck.add(Ck_item)
    return Ck
```

判定是否保留：若本來就小於 min support 的，則下面的子集 itemset 也一定小於 min support，可直接刪除，不用再重跑一次。把觀念寫進程式裡：

is_apriori：若 Ck 的 k-1 項子集不在 Lk-1 中(即不大於等於 min support)，則此項集合可以直接從 Ck 中刪除，由此可獲得 Ck。

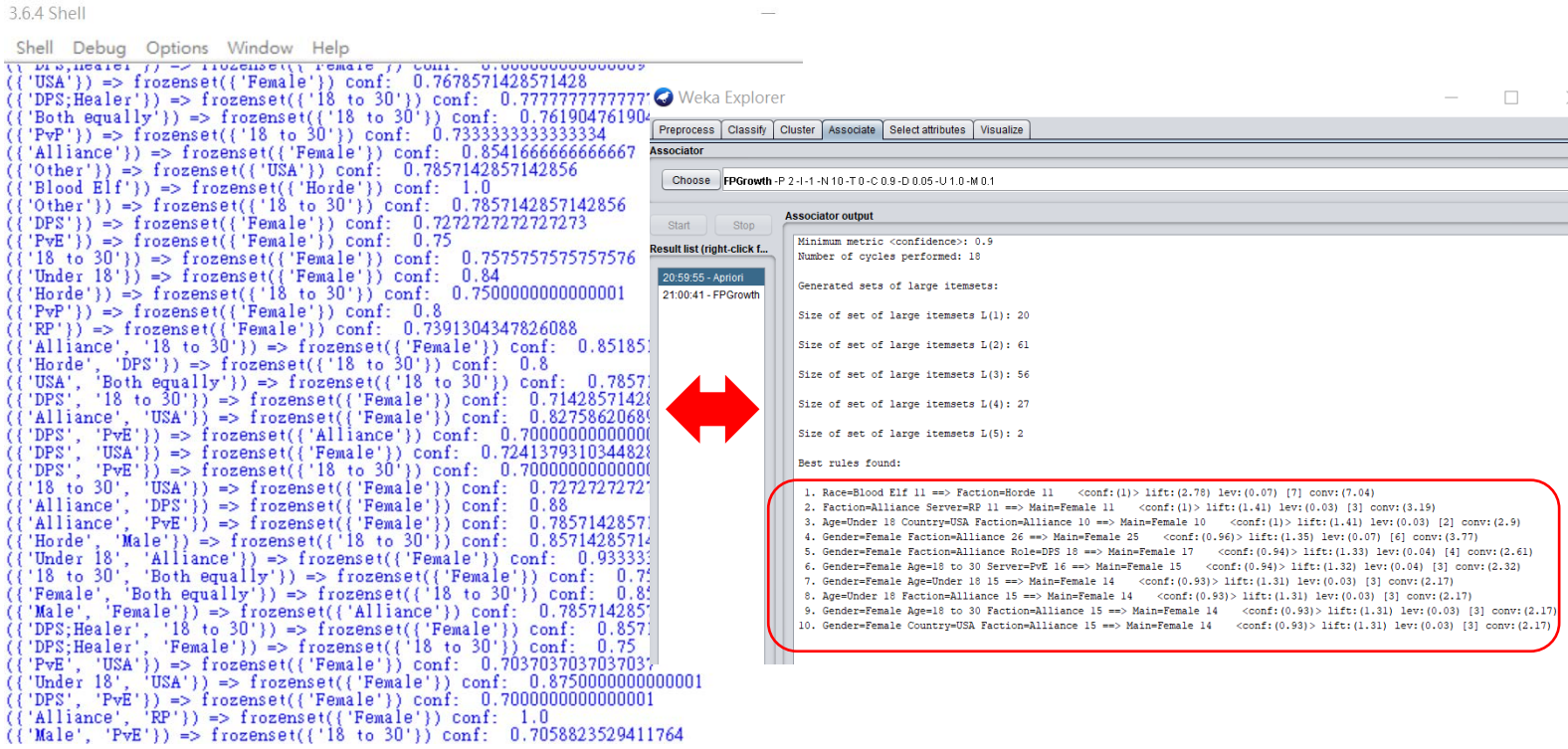
```
def generate_Lk_by_Ck(data_set, Ck, min_support, support_data):
    Lk = set()
    item_count = {}
    for t in data_set:
        for item in Ck:
            if item.issubset(t):
                if item not in item_count:
                    item_count[item] = 1
                else:
                    item_count[item] += 1
    t_num = float(len(data_set))
    for item in item_count:
        if (item_count[item] / t_num) >= min_support:
            Lk.add(item)
            support_data[item] = item_count[item] / t_num
    return Lk


def generate_L(data_set, k, min_support):
    support_data = {}
    C1 = create_C1(data_set)
    L1 = generate_Lk_by_Ck(data_set, C1, min_support, support_data)
    Lksub1 = L1.copy()
    L = []
    L.append(Lksub1)
    for i in range(2, k+1):
        Ci = create_Ck(Lksub1, i)
        Li = generate_Lk_by_Ck(data_set, Ci, min_support, support_data)
        Lksub1 = Li.copy()
        L.append(Lksub1)
    return L, support_data
```

找出 Lk，透過 Lk-1 與 k 連接所產生集合記做：Ck

## ✧ Dataset 執行：

以魔獸世界在 2017 年統計之玩家資料作為範例執行 Apriori。




上圖：將程式輸出結果與 WEKA 輸出結果做答案比對

(程式列出所有關聯法則，WEKA 設定列出 10 項關聯法則，因此後續將針對 WEKA 之關聯法則進行詳細分析)

## ✧ 分析：

藉由程式與 WEKA 結果可分析出女性玩家選女生角色有相當高之關聯，而在女性玩家中：

(1) 屬於聯盟到陣營(Faction = Alliance)製造傷害之人(DPS)

(2) 年齡介於 18 到 30 歲之人，Server 是 PVE 或屬於連盟陣營

(3) 年齡小於 18 歲

(4) 玩家國家在美國，屬於連盟陣營(Faction = Alliance)之人

玩家所選擇角色為女生角色有相當高之關聯。

另外，年齡小於 18 歲中：

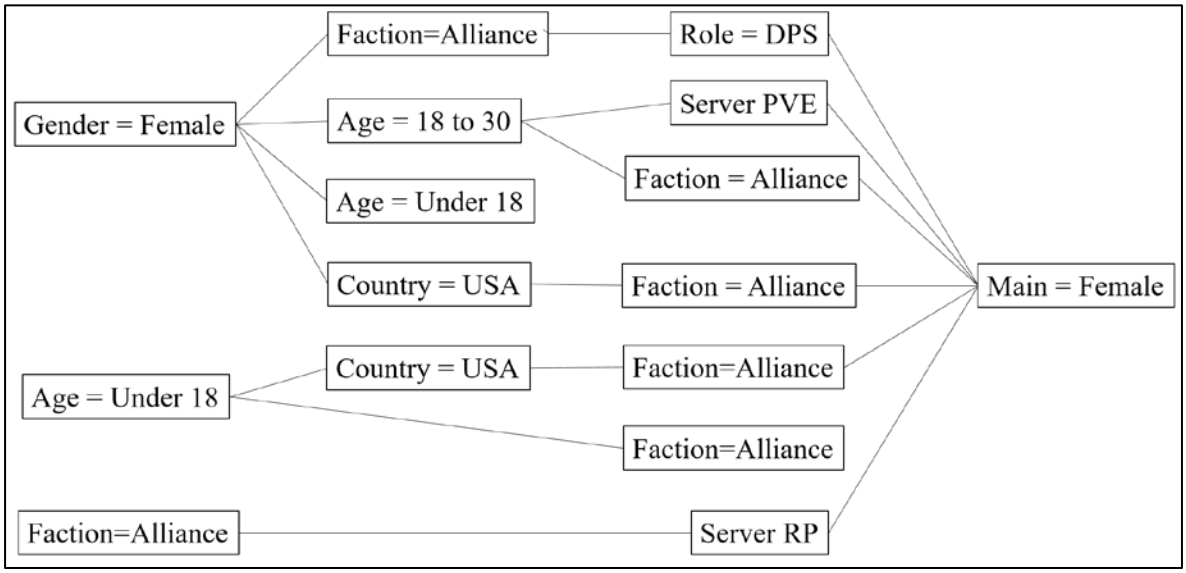
(1) 玩家國家在美國，屬於連盟陣營(Faction = Alliance)

(2) 屬於連盟陣營(Faction = Alliance)之人

玩家所選擇角色為女生角色有相當高之關聯。

而屬於連盟陣營(Faction = Alliance)且伺服器屬於 RP 之玩家所選擇角色為女生角色也有相當高之關聯。

綜觀來說，玩家選擇的角色為女生角色和非常多 item 皆有相當高之關聯。而連盟陣營(Faction = Alliance)和年齡小於 18 歲(Age = Under 18)和玩家會選擇女生角色 (Female) 呈現高度正相關。




圖：將關聯結果畫出並分析之示意圖

## (二)FP-Growth

### ✧ 步驟：

分為兩部分：根據原始資料構成 FP-tree；在 FP-tree 上挖掘頻繁模式，產生 FP-tree，建構 FP-tree 的過程中：

(1) 掃描所有 item，找出 frequent 1-item，按照 support 值排序調整

(2) 在掃瞄所有 item，利用每個事物中的 frequent itemset 創造 FP-tree

(3) FP-tree 根結點為 null，按排序順序將事物出現的項目添加到相關之分支

## (三)程式部分內容：

```
    def root(self):
        return self._root

    def add(self, transaction):
        point = self._root

        for item in transaction:
            next_point = point.search(item)
            if next_point:
                next_point.increment()
            else:
                next_point = FPNode(self, item)
                point.add(next_point)
                self._update_route(next_point)

            point = next_point

    def _update_route(self, point):
        assert self is point.tree

        try:
            route = self._routes[point.item]
            route[1].neighbor = point
            self._routes[point.item] = self.Route(route[0], point)
        except KeyError:
            self._routes[point.item] = self.Route(point, point)
```

此樹已有當前節點，創建新節點添加到子節點中，再更新此節點的新節點。

```
def conditional_tree_from_paths(paths):
    tree = FPTree()
    condition_item = None
    items = set()
    for path in paths:
        if condition_item is None:
            condition_item = path[-1].item

        point = tree.root
        for node in path:
            next_point = point.search(node.item)
            if not next_point:
                items.add(node.item)
                count = node.count if node.item == condition_item else 0
                next_point = FPNode(tree, node.item, count)
                point.add(next_point)
                tree._update_route(next_point)
            point = next_point
```

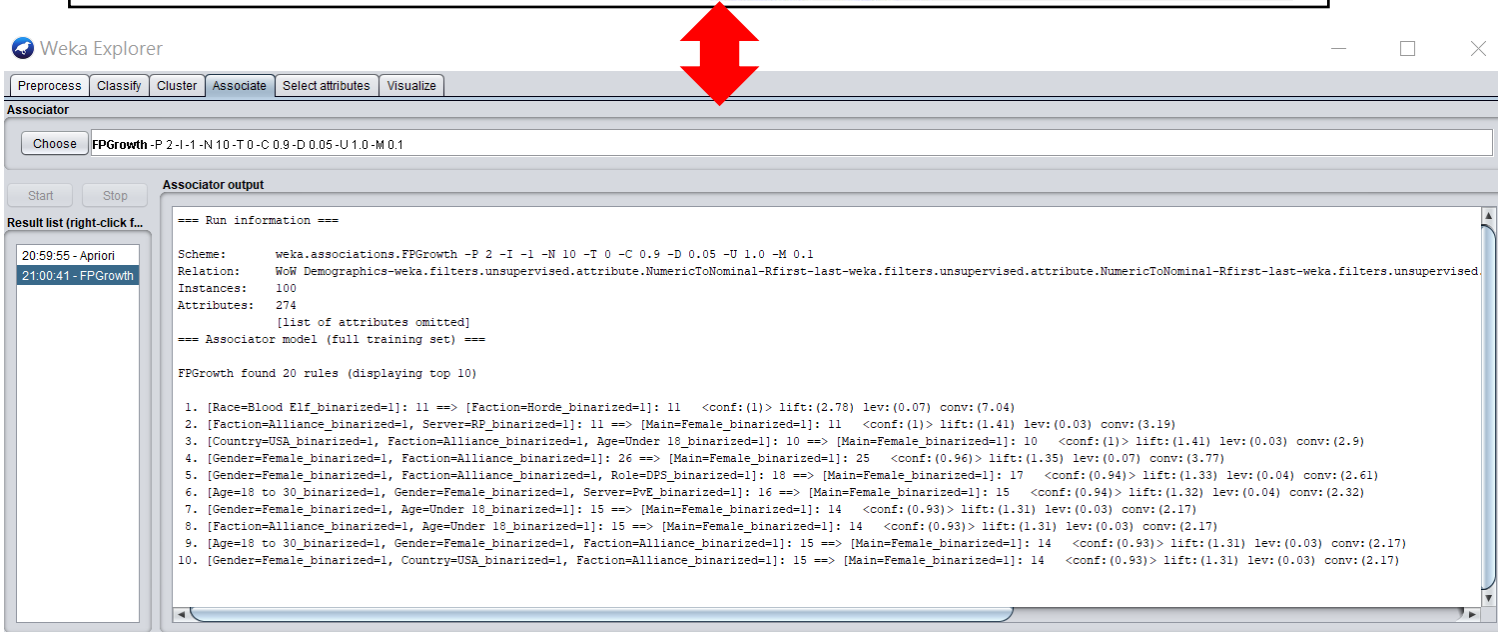
從 prefix path 建立 cnditional FP-tree

將 path 中的節點導入新樹，只計算 leaf nodes，其他節點計算 leaf count)

## ✧ Dataset 執行：

以魔獸世界在 2017 年統計之玩家資料作為範例執行 FP-Growth。

```
['Female'] 129
['Female', '18 to 30'] 81
['Female', '18 to 30', 'Alliance'] 37
['Female', '18 to 30', 'Alliance', 'DPS'] 21
['Female', '18 to 30', 'Alliance', 'PvE'] 23
['Female', '18 to 30', 'Both equally'] 21
['Female', '18 to 30', 'DPS'] 34
['Female', '18 to 30', 'DPS', 'Horde'] 12
['Female', '18 to 30', 'DPS;Healer'] 20
['Female', '18 to 30', 'DPS;Tank'] 11
['Female', '18 to 30', 'Horde'] 28
['Female', '18 to 30', 'PvE'] 35
['Female', '18 to 30', 'PvE', 'DPS'] 17
['Female', '18 to 30', 'PvP'] 16
['Female', '18 to 30', 'RP'] 12
['Female', '18 to 30', 'USA'] 38
['Female', '18 to 30', 'USA', 'Alliance'] 18
['Female', '18 to 30', 'USA', 'Alliance', 'PvE'] 13
['Female', '18 to 30', 'USA', 'Both equally'] 15
['Female', '18 to 30', 'USA', 'DPS'] 18
['Female', '18 to 30', 'USA', 'PvE'] 19
['Female', 'Alliance'] 66
['Female', 'Alliance', 'DPS'] 39
['Female', 'Alliance', 'Male'] 11
['Female', 'Alliance', 'PvE'] 36
['Female', 'Alliance', 'PvE', 'DPS'] 20
['Female', 'Alliance', 'RP'] 18
['Female', 'Alliance', 'Under 18'] 22
['Female', 'Blood Elf'] 11
['Female', 'Both equally'] 25
['Female', 'DPS'] 56
['Female', 'DPS', 'Horde'] 16
['Female', 'DPS', 'RP'] 17
['Female', 'DPS', 'Under 18'] 18
['Female', 'DPS;Healer'] 27
['Female', 'DPS;Tank'] 14
['Female', 'Druid'] 14
['Female', 'Horde'] 43
['Female', 'Horde', 'Blood Elf'] 11
['Female', 'Male'] 14
['Female', 'PvE'] 57
['Female', 'PvE', 'DPS'] 26
['Female', 'PvE', 'Horde'] 15
```




上圖：將程式輸出結果與 WEKA 輸出結果做答案比對

## ✧ 分析：

藉由程式與 WEKA 結果可分析出女性玩家選女生角色有相當高之關聯，因在所有規則量中，女性玩家找到的規則數最多，因此以女性玩家作為以下詳細的分析。經由結果可分析出，女性玩家(Gender = Female)的規則數為 129，而當玩家為性別為女性 (Female) 時，可觀察出：

1. 年紀為 18 到 30 的玩家，選擇：

   (1) 聯盟陣營(Alliance)

(2) 在遊戲中製造傷害的人(DPS)；

(3) 為普通伺服器(PVE)；

(4) 玩家所屬國家在美國(USA)，都有高頻率之關係。

2. 屬於聯盟陣營 (Alliance) 的玩家，選擇：

(1) 在遊戲中製造傷害的人(DPS)；

(2) 為普通伺服器(PVE)，都有高頻率之關係。

3. 屬於在遊戲中製造傷害的人(DPS)。

4. 屬於普通伺服器(PVE)的玩家，選擇：

(1) 在遊戲中製造傷害的人(DPS)；

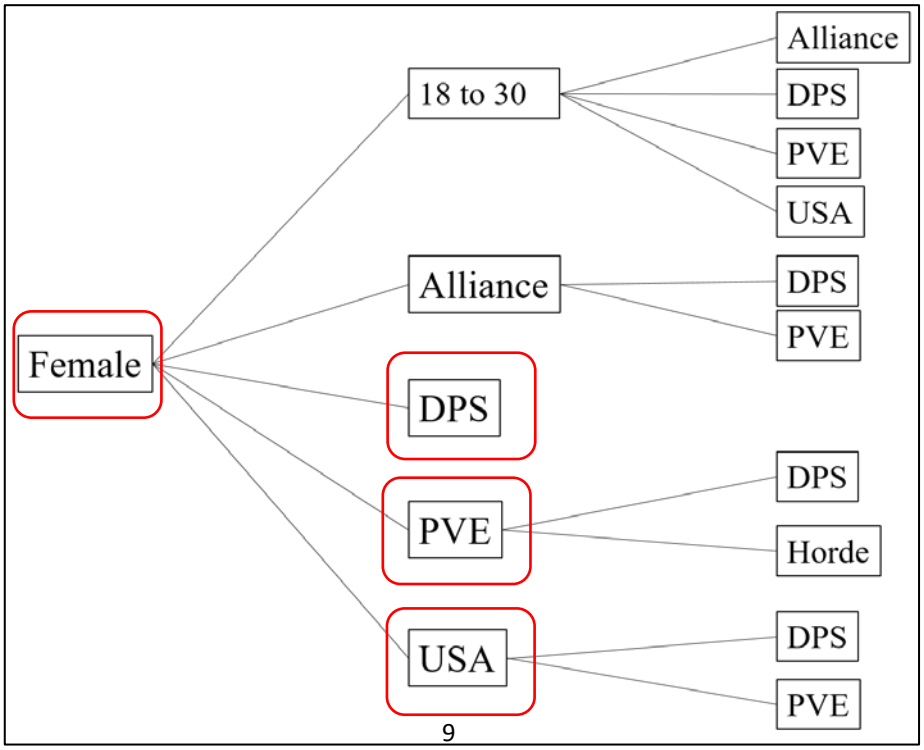
(2) 部落陣營(Horde)，都有高頻率之關係。

5. 所屬國家在美國(USA)之玩家，選擇：

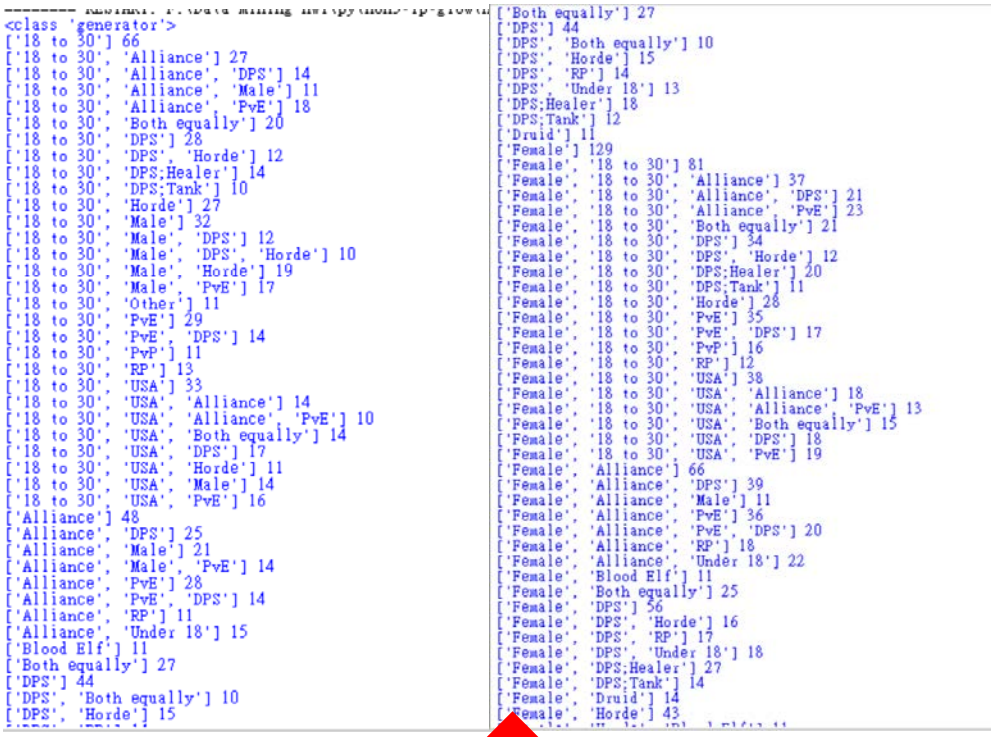
(1) 在遊戲中製造傷害的人(DPS)；

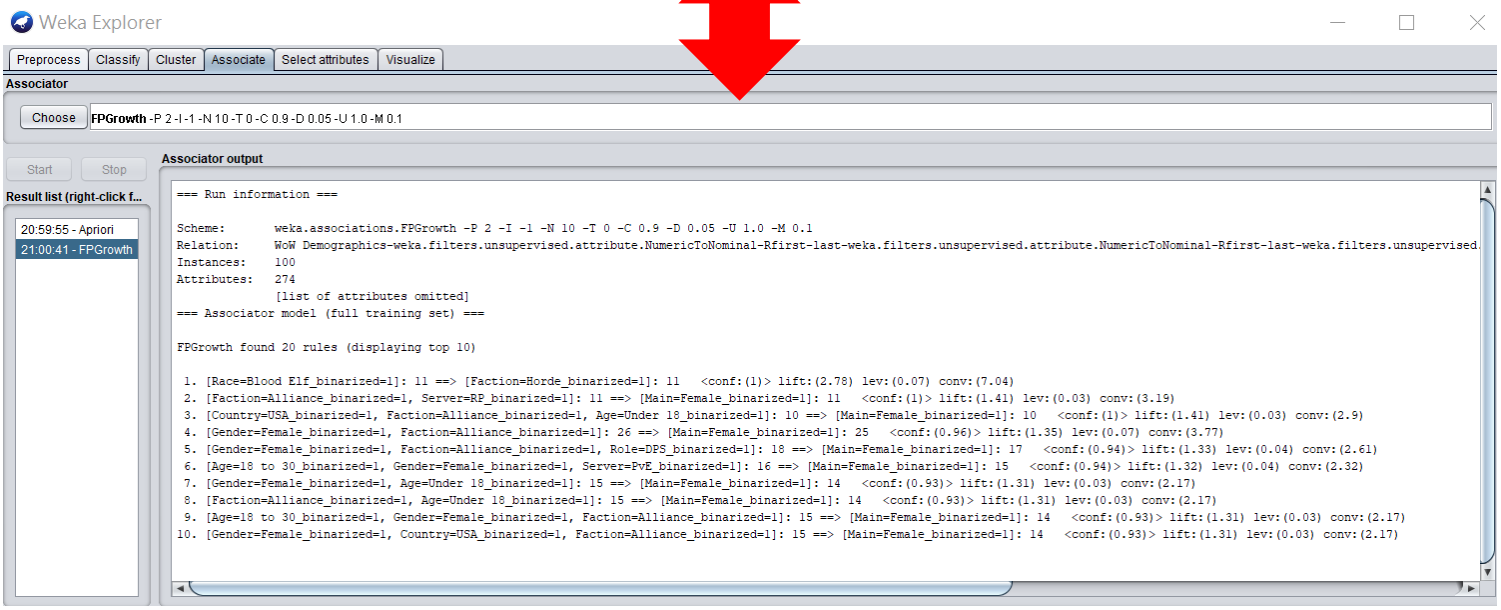
(2) 為普通伺服器(PVE)，都有高頻率之關係。

可由以上分析看出，當玩家性別為女性 (Female) 時，和製造傷害的玩家 (DPS) 及普通伺服器(PVE)的玩家、所屬國家在美國(USA)之玩家呈現高度正相關。如下圖為將關聯結果畫出並分析之示意圖。



1. 比較兩者輸出的規則，發現實作的演算法雖然要跑 30 S 但是可以找到完整的 Pattern，幾乎所有能找到的規則都出現了，但是 WEKA 只有找到與此相關的相似規則。



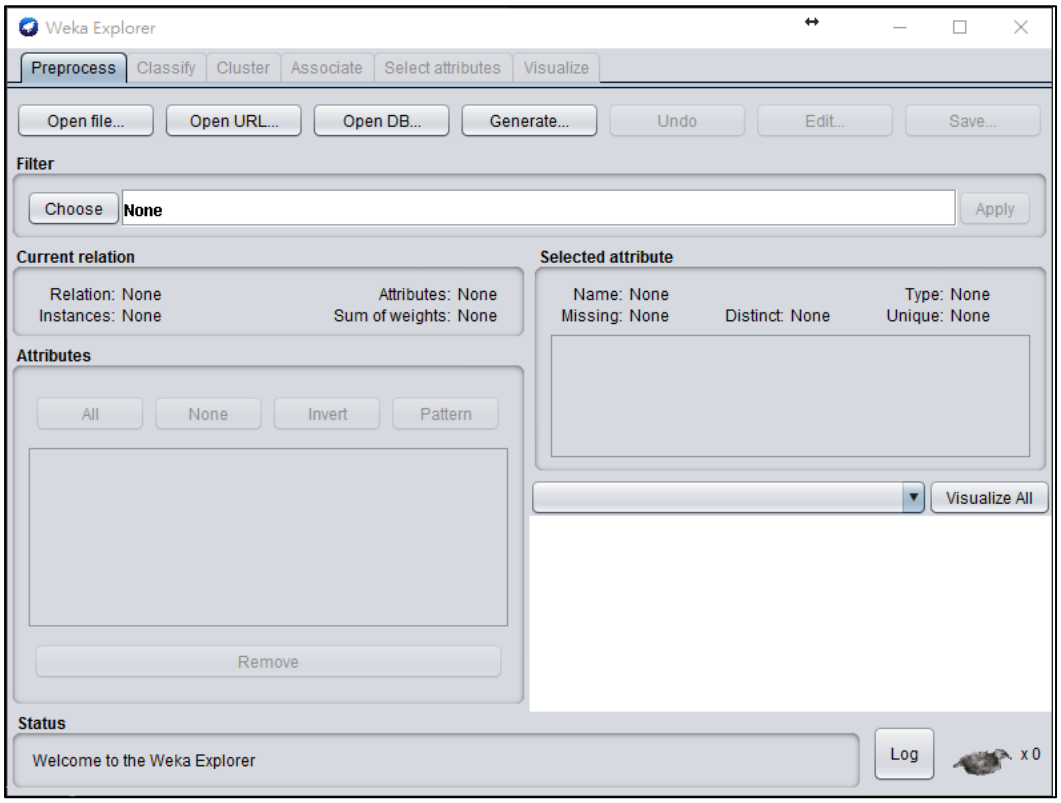
## (三)IBM

### ✧ WEKA 執行：演算法：Apriori

使用–IBM-Quest-Data-Generator.exe參數產生資料。

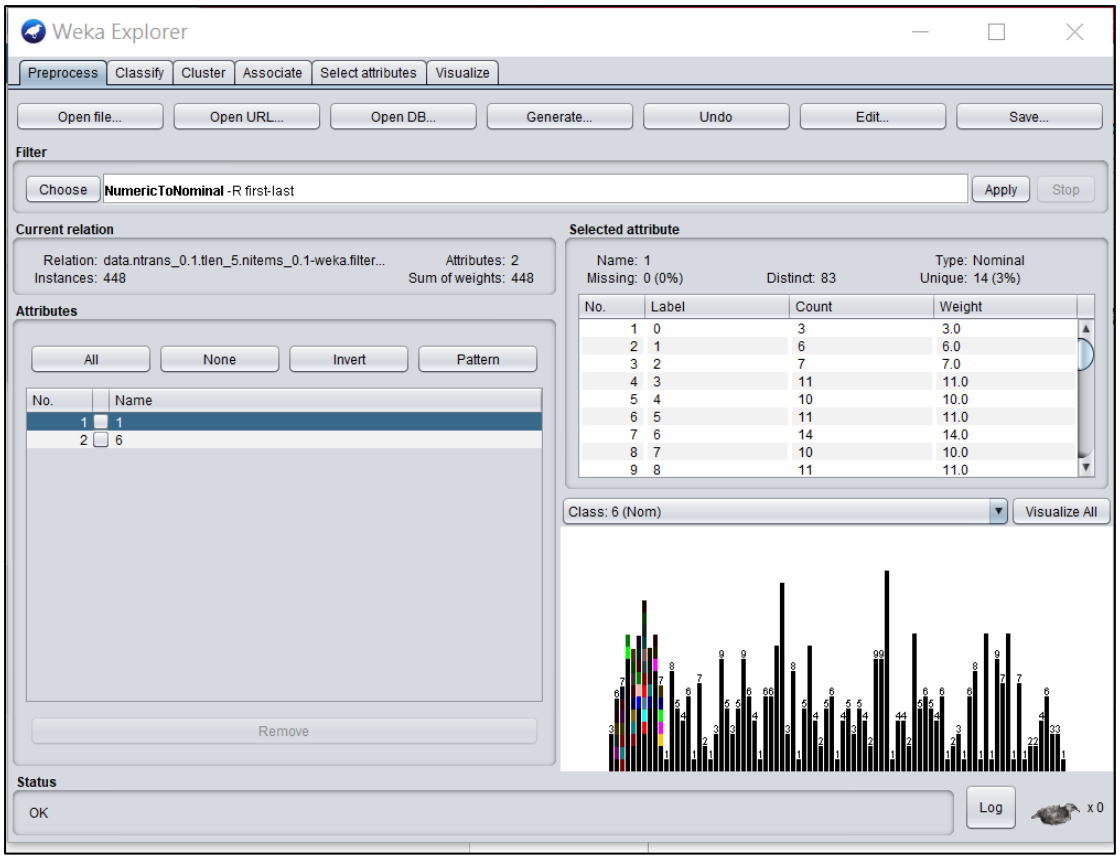
**步驟：**

1. 開啟WEKA：



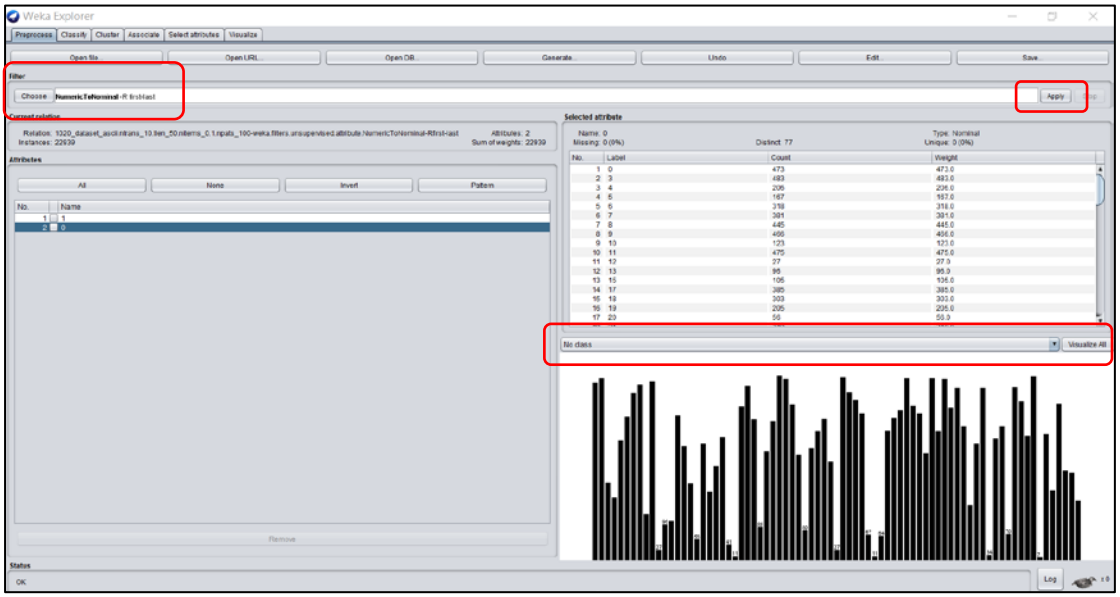

2. 讀取檔案：

先放入較小的資料進行測試如下圖(data.ntrans_0.1.tlen_5.nitems_0.1.csv)，可以看到左上有資料的詳細資訊，以及該資料中各個 Item 占所有的比例，後續測試發現，差距越大，越容易找到關聯規則，因此後面的平均每筆交易出現的商品數目會特別往上調整。



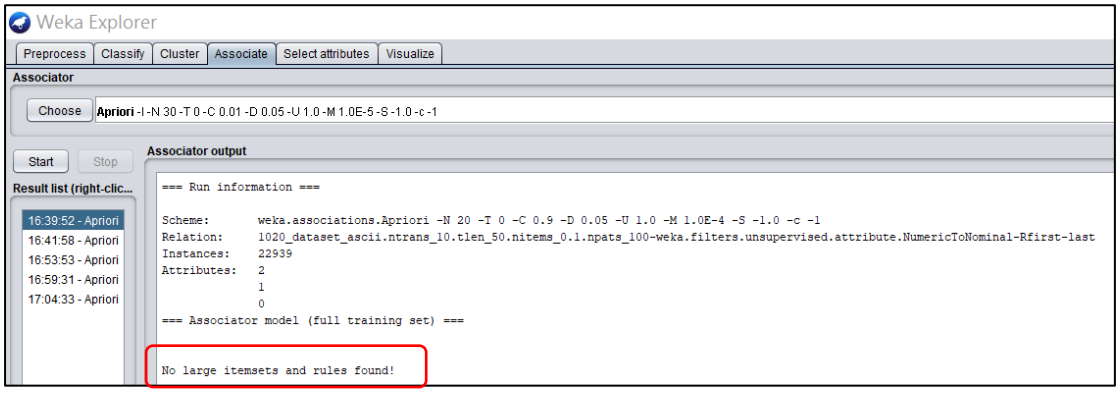

3. 設定資料型態選擇演算法：

這步驟會發現如果所使用的資料不符合規定，會無法選取演算法(無法點擊 Apriori、FP-Growth)。因為 Apriori 不能處理連續的數值型 (Numeric) 資料，只能處理名目資料 (Nominal)。因此需要在 preprocess 書籤的 filter 選擇 NumericToNomial 功能，Visualize 選擇 No class 後點擊 Apply。如下圖所示。

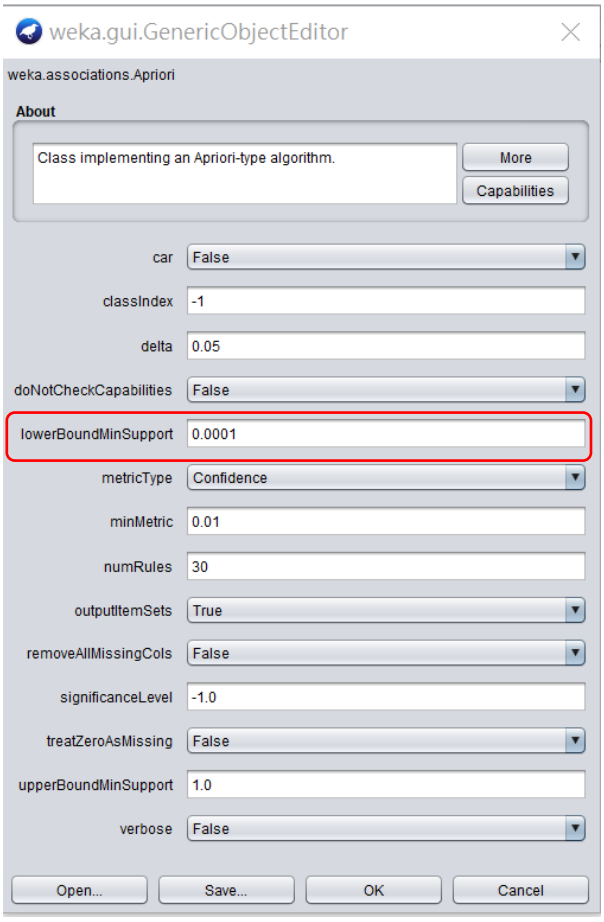


4. 建立 Model：

按下 Start 之後，會將數據建立成 model，並且進行演算法，找尋 Rules，在這其中碰到的問題是找不到 rules，出現 No large itemsets and rules found，之後發現可能是因為參數的門檻太高，因此在其他資料的測試中，將 lowerBoundMinSupport 降低，甚至有調低到 0.0001 才出現規則的情況。




調整後之參數：



5. 問題與解決方法

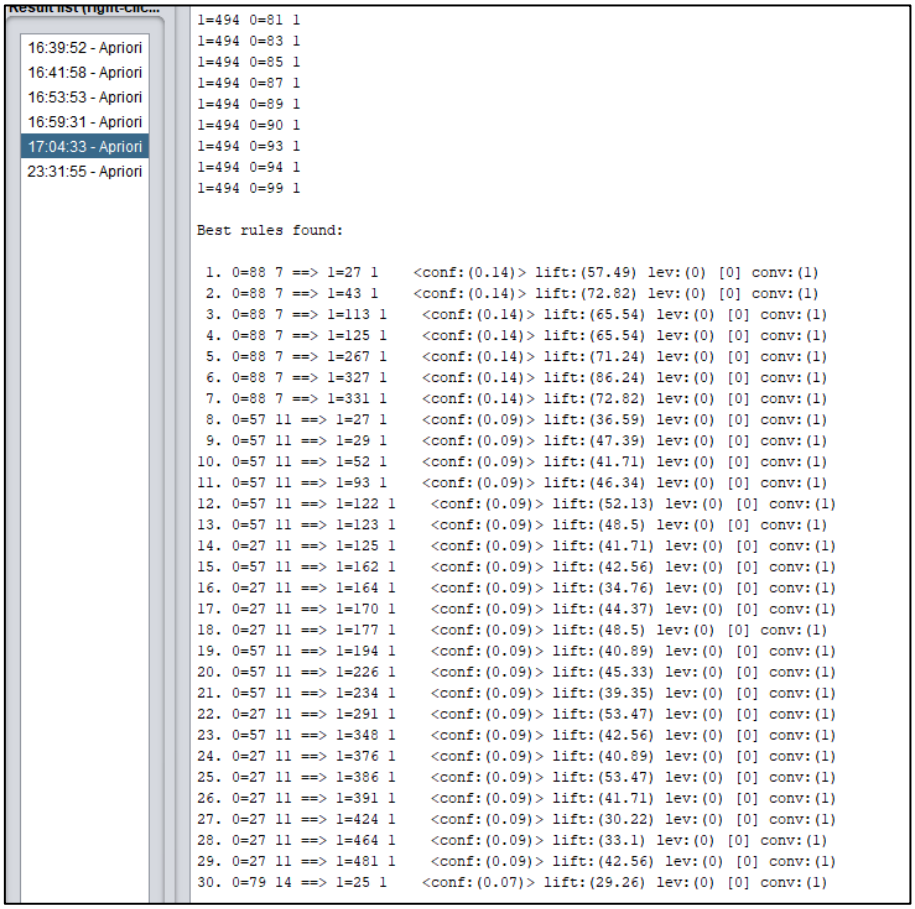
執行 IBM 資料檔：

1020_dataset_ascii.ntrans_10.tlen_50.nitems_0.1.npats_100.arff 過程中因為下列參數設定因此沒有找到規則：

| | |
|---|---|
| **lowerBoundMinSupport** | **0.1** |
| **minMetric** | 0.9 |
| **upperBoundMinSupport** | 1.0 |

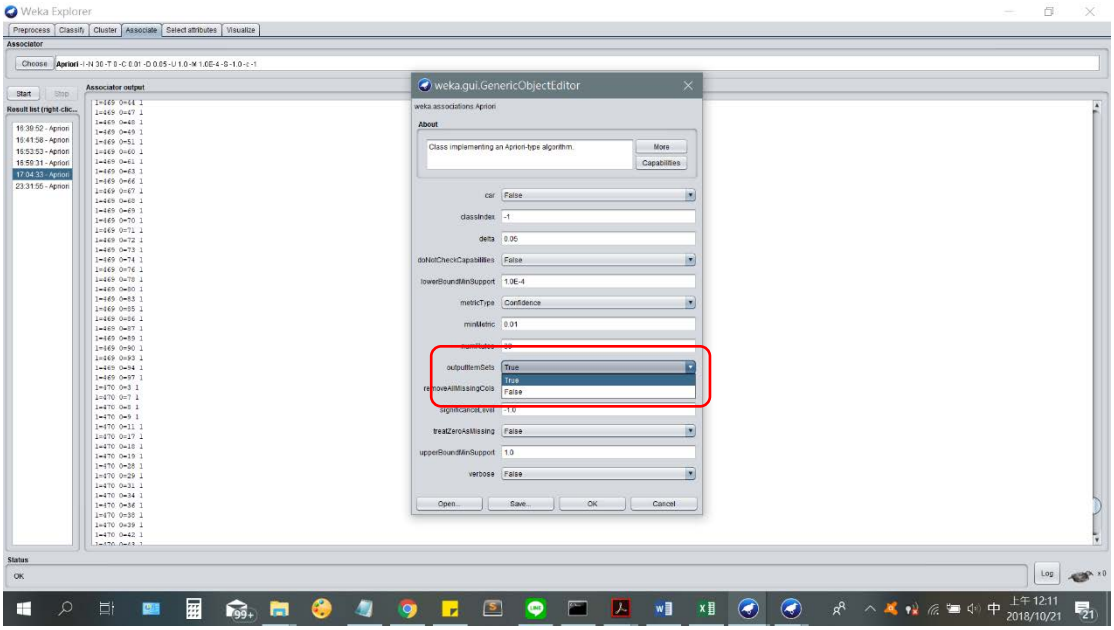
降低數值

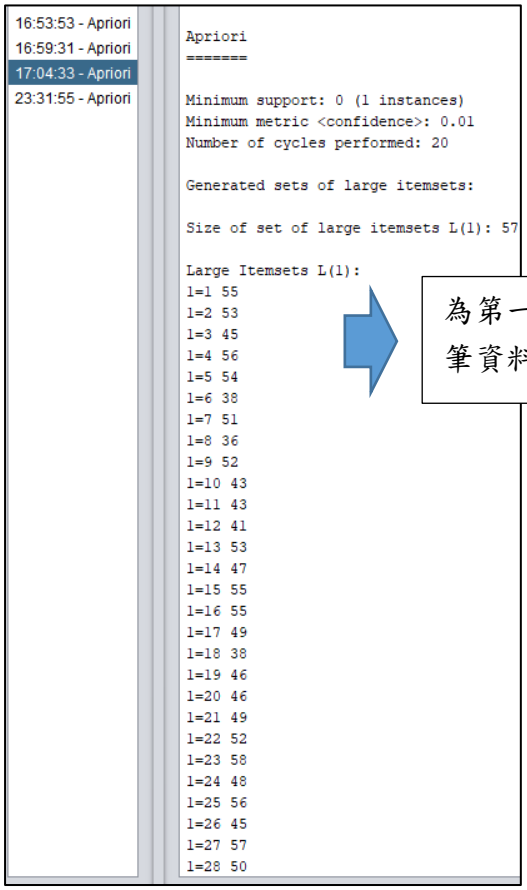
後來將 lowerBoundMinSupport 降低至 0.0001，minMetric 調為 0.01，upperBoundMinSupport 調為 0.1，找到了 30 條規則。



6. 分析所有 itemset：

使用 WEKA 分析過程中，為了觀察出資料間產生的關聯有何規則，因此將 WEKA 列出所有的 itemset 以利後續分析觀察，將 outputitemset 選項選擇 True(本來 WEKA 設定為 False)，會列出所有 itemset，從資料中觀察出：




為第一筆交易資料有 55 筆資料，第二筆交易數量有 53 筆資料，第三筆交易數量有 45 筆資料…以此類推。

7. 找到之規則及想法：

在 1020_dataset_ascii.ntrans_10.tlen_50.nitems_0.1.npats_100.arff 的資料中，將 lowerBoundMinSupport 從預設 0.1 調整到 0.0001 才找到規則，試著把數值提高至 0.0008 還是可以找到規則，但規則數會開始慢慢減少，到 0.001 再往上後就都找不到任何規則，後來發現 delta 比起 lowerBoundMinSupport 調整的幅度高太多，因此調整 delta 的大小，終於找到 rule。但個人覺得還是因為調整

lowerBoundMinSupport 的可能性大於調整其他參數，也或許是資料在分析時太集中在 items 值上。下圖為 WEKA 找到之規則。

```
Best rules found:

 1. 0=88 7 ==> 1=27 1    <conf:(0.14)> lift:(57.49) lev:(0) [0] conv:(1)
 2. 0=88 7 ==> 1=43 1    <conf:(0.14)> lift:(72.82) lev:(0) [0] conv:(1)
 3. 0=88 7 ==> 1=113 1    <conf:(0.14)> lift:(65.54) lev:(0) [0] conv:(1)
 4. 0=88 7 ==> 1=125 1    <conf:(0.14)> lift:(65.54) lev:(0) [0] conv:(1)
 5. 0=88 7 ==> 1=267 1    <conf:(0.14)> lift:(71.24) lev:(0) [0] conv:(1)
 6. 0=88 7 ==> 1=327 1    <conf:(0.14)> lift:(86.24) lev:(0) [0] conv:(1)
 7. 0=88 7 ==> 1=331 1    <conf:(0.14)> lift:(72.82) lev:(0) [0] conv:(1)
 8. 0=57 11 ==> 1=27 1    <conf:(0.09)> lift:(36.59) lev:(0) [0] conv:(1)
 9. 0=57 11 ==> 1=29 1    <conf:(0.09)> lift:(47.39) lev:(0) [0] conv:(1)
10. 0=57 11 ==> 1=52 1    <conf:(0.09)> lift:(41.71) lev:(0) [0] conv:(1)
11. 0=57 11 ==> 1=93 1    <conf:(0.09)> lift:(46.34) lev:(0) [0] conv:(1)
12. 0=57 11 ==> 1=122 1    <conf:(0.09)> lift:(52.13) lev:(0) [0] conv:(1)
13. 0=57 11 ==> 1=123 1    <conf:(0.09)> lift:(48.5) lev:(0) [0] conv:(1)
14. 0=27 11 ==> 1=125 1    <conf:(0.09)> lift:(41.71) lev:(0) [0] conv:(1)
15. 0=57 11 ==> 1=162 1    <conf:(0.09)> lift:(42.56) lev:(0) [0] conv:(1)
16. 0=27 11 ==> 1=164 1    <conf:(0.09)> lift:(34.76) lev:(0) [0] conv:(1)
17. 0=27 11 ==> 1=170 1    <conf:(0.09)> lift:(44.37) lev:(0) [0] conv:(1)
18. 0=27 11 ==> 1=177 1    <conf:(0.09)> lift:(48.5) lev:(0) [0] conv:(1)
19. 0=57 11 ==> 1=194 1    <conf:(0.09)> lift:(40.89) lev:(0) [0] conv:(1)
20. 0=57 11 ==> 1=226 1    <conf:(0.09)> lift:(45.33) lev:(0) [0] conv:(1)
21. 0=57 11 ==> 1=234 1    <conf:(0.09)> lift:(39.35) lev:(0) [0] conv:(1)
22. 0=27 11 ==> 1=291 1    <conf:(0.09)> lift:(53.47) lev:(0) [0] conv:(1)
23. 0=57 11 ==> 1=348 1    <conf:(0.09)> lift:(42.56) lev:(0) [0] conv:(1)
24. 0=27 11 ==> 1=376 1    <conf:(0.09)> lift:(40.89) lev:(0) [0] conv:(1)
25. 0=27 11 ==> 1=386 1    <conf:(0.09)> lift:(53.47) lev:(0) [0] conv:(1)
26. 0=27 11 ==> 1=391 1    <conf:(0.09)> lift:(41.71) lev:(0) [0] conv:(1)
27. 0=27 11 ==> 1=424 1    <conf:(0.09)> lift:(30.22) lev:(0) [0] conv:(1)
28. 0=27 11 ==> 1=464 1    <conf:(0.09)> lift:(33.1) lev:(0) [0] conv:(1)
29. 0=27 11 ==> 1=481 1    <conf:(0.09)> lift:(42.56) lev:(0) [0] conv:(1)
30. 0=79 14 ==> 1=25 1    <conf:(0.07)> lift:(29.26) lev:(0) [0] conv:(1)
```

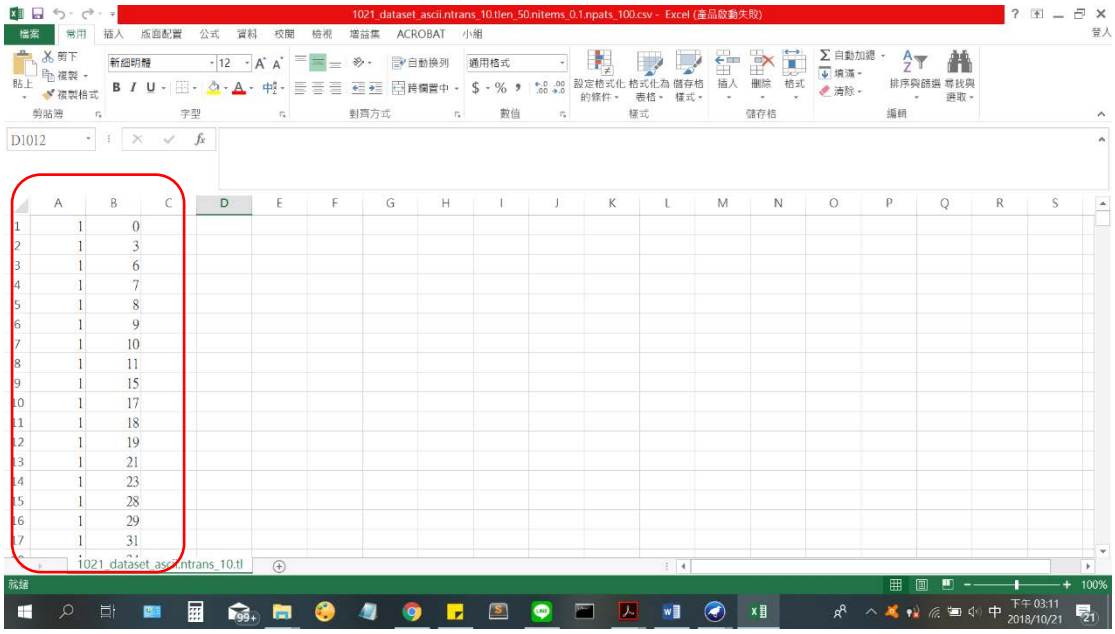
有 7 個 88，分別出現 27、43、113、125、267、327、331 之 7 個數，因此 confidence 值為 1/7=0.14，往下以此類推。

## ✧ WEKA 執行：演算法：FP-Growth

## ✧ 步驟：

1. 基本上操作步驟和前面 Apriori 所述差不多，但值得注意的是，在選擇 FP-Growth 時，點選的資料述值型態不同，所以操作過程需特別注意。

因為須找到 A 和 B 資料出現的關係(如下圖所示)，因此資料數值需轉換成 Binary。



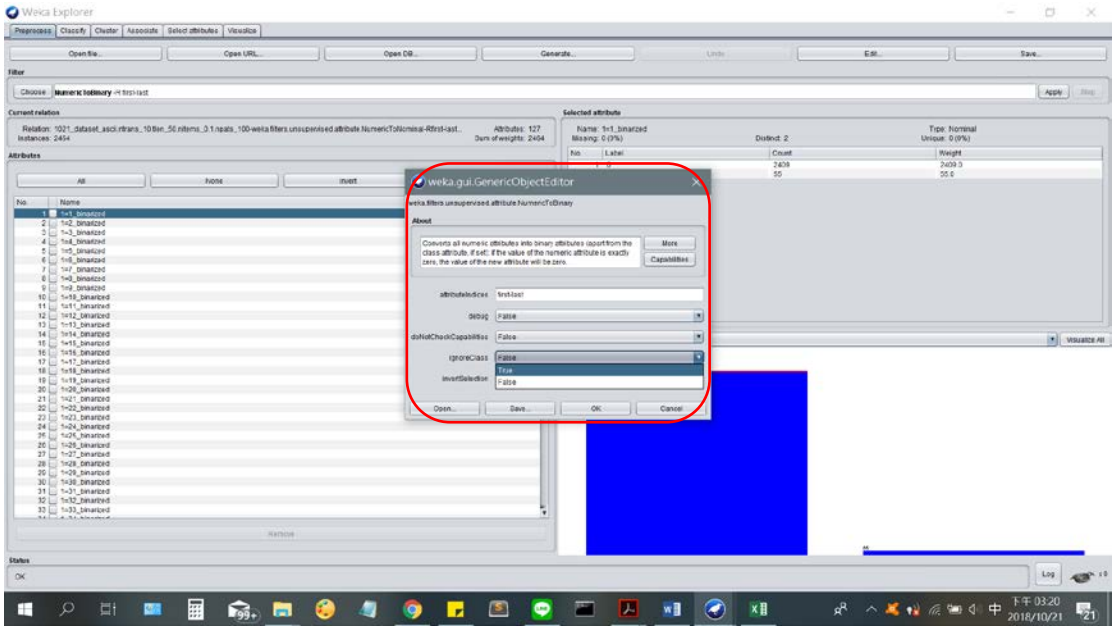

2. FP-Growth 處理資料步驟如下：

(1) Filter 選擇 NumericTONominal → NominalToBinary →NumericTOBinary

(2) 每一個型態之 Class 部分須選擇 NoClass

(3) NumericTOBinaryvm 須將 ignoreclaa 選擇 True (WEKA 預設為 False)

(4) 選擇 Apply




3. 問題與解決方法：

一開始因為產生的資料太大導致錯誤訊息顯示 WEKA 內部記憶體容量不足，因此將資料篩選縮減過後，成功放入 WEKA 執行。

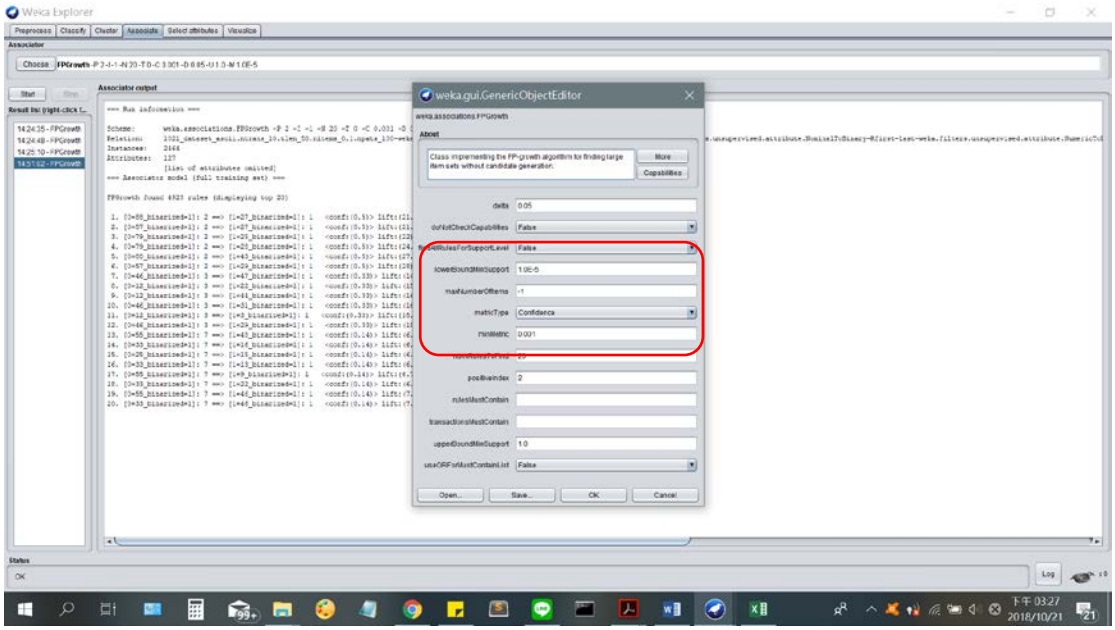
過程中一樣因為下列參數設定因此沒有找到規則：

| | |
|---|---|
| **lowerBoundMinSupport** | **0.1** |
| **minMetric** | 0.9 |
| **upperBoundMinSupport** | 1.0 |

為 WEKA 預設之參數數

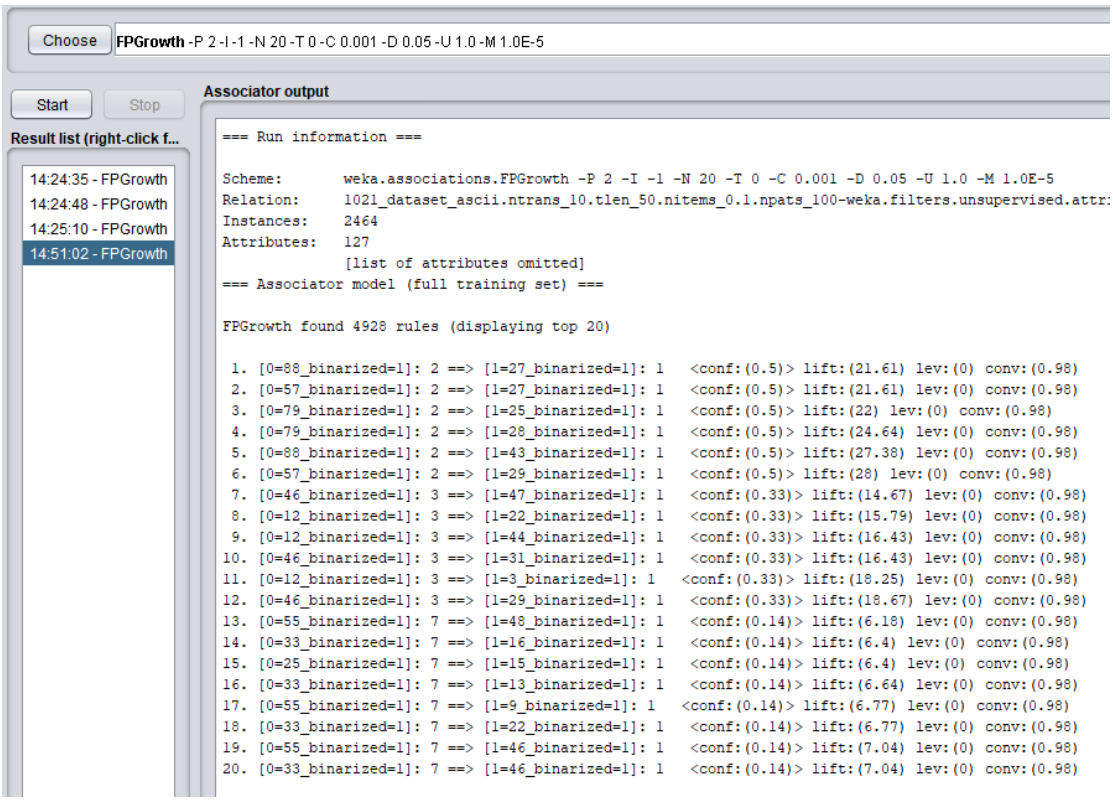
降低數值

後來將 lowerBoundMinSupport 降低至 0.00001，minMetric 調為 0.001，upperBoundMinSupport 調為 0.1，找到了 20 條規則。




4. 找到之規則及想法：

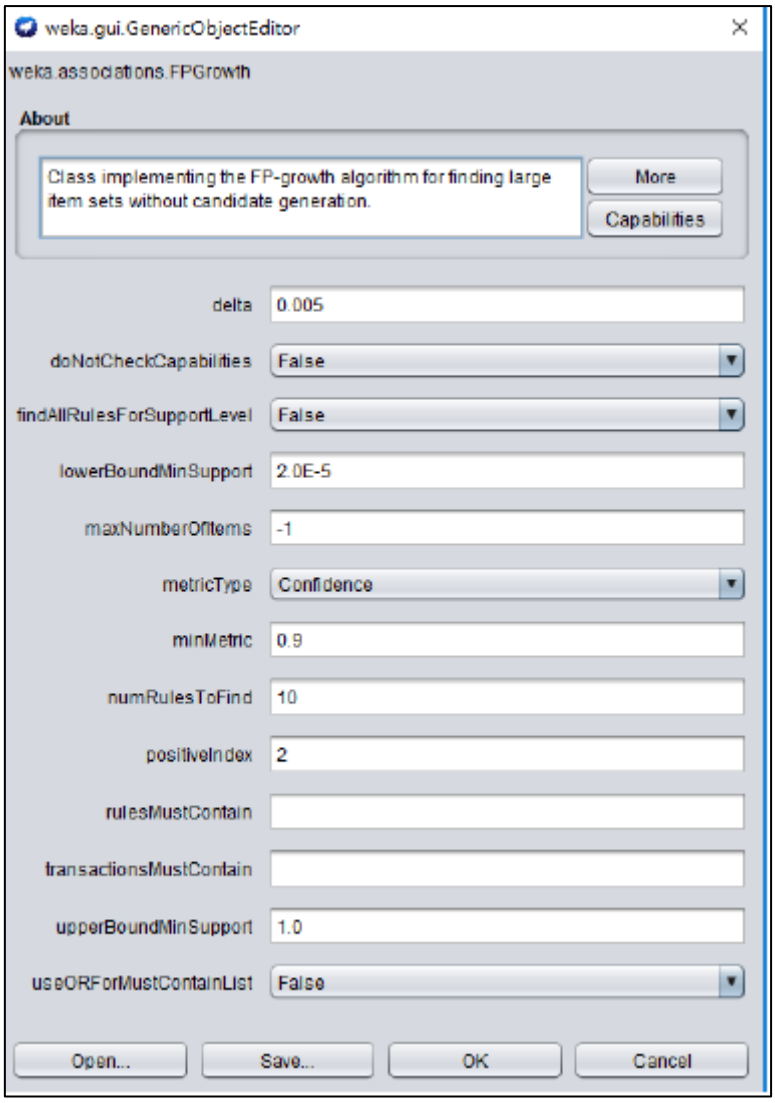
在 1021FP_dataset_ascii.ntrans_10.tlen_50.nitems_0.1.npats_100.arff 的資料中，將 lowerBoundMinSupport 從預設 0.1 調整到 0.00001 才找到規則，調整 lowerBoundMinSupport 的可能性大於調整其他參數，也或許是資料在分析時太集中在 items 值上。下圖為 WEKA 找到之規則。




發現第 88 個商品在第 27 筆資料和第 43 筆資料重複出現，confidence = 1/2 =0.5，第 12 個商品在第 22、44、3 筆資料重複出現，因此 confidence = 1/3 =0.33…等等。

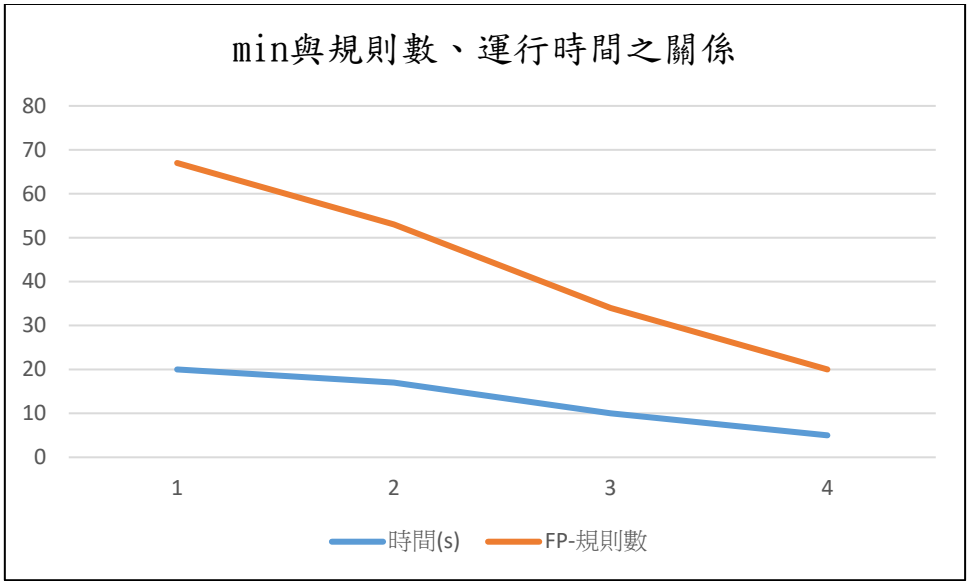
5. 比較結果分析：

使用 WEKA 的 FPGrowth 以及實作的 FPGrowth 演算法跑同一個資料，可以觀察到，WEKA 僅僅需要 9.30 秒，而實作的演算法需要 58.93，觀察出兩者演算法時間差距在於，WEKA 有設定兩個終止條件，加快了演算法的速度，其一為每次迭代改變的 min support 值，另一個則是預計找到的規則上限，達到其一則會停止，因此可以加速時間上之運行。




一開始執行 WEKA 時，因為門檻值的設定，所以有許多次都無法成功跑出相關規則，後來上網查資料，發現和參數設定有關，因此做了許多次不同參數的分析，發現 min 門檻值會影響搜尋規則數及搜尋時間，一開始 min 值設定過高幾乎都找不到規則，後來設定成 0.0001 及 0.00001 到 0.00005 區間都可以跑出關聯規則，發現 min 值愈高與時間及規則數成反比關係，原因可能是由於門檻值調動，使相關交易物品的規則數找到更多。

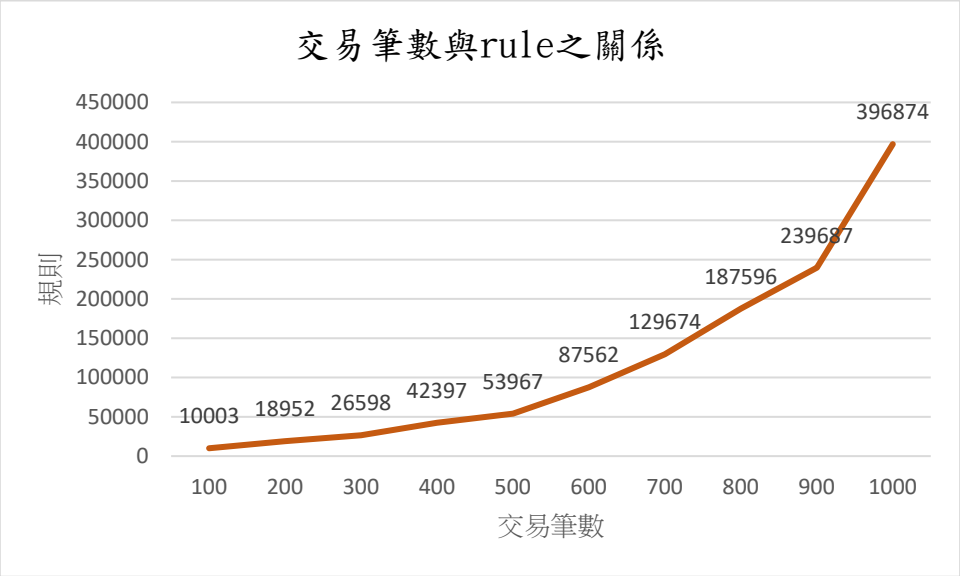
| ntrans交易量 *(default=1000) | Tlen 交易平均數量 | Nitems 交易數量 | Npats 交易項目數量 | Minsup | 時間(s) | FP-規則數 |
|---|---|---|---|---|---|---|
| **10** | 50 | 0.1 | 100 | 1 | 20 | 67 |
| **10** | 50 | 0.1 | 100 | 2 | 17 | 53 |
| **10** | 50 | 0.1 | 100 | 3 | 10 | 34 |
| **10** | 50 | 0.1 | 100 | 4 | 5 | 20 |




圖：Min 與 rule、time 關係

✧ 變量：總交易數量與執行時間、規則之關係

| 交易筆數 | 平均每筆交易物品量 | 物品種類數量 | Minsup | 執行時間 | FP Growth 關聯規則量 |
|---|---|---|---|---|---|
| 100 | 10 | 100 | 1 | <1s | 10003 |
| 200 | 10 | 100 | 1 | <1s | 18952 |
| 300 | 10 | 100 | 1 | <1s | 26598 |
| 400 | 10 | 100 | 1 | <1s | 42397 |
| 500 | 10 | 100 | 1 | 1~2s | 53967 |
| 600 | 10 | 100 | 1 | 2~3s | 87562 |
| 700 | 10 | 100 | 1 | 2~3s | 129674 |
| 800 | 10 | 100 | 1 | 3~4s | 187596 |
| 900 | 10 | 100 | 1 | 3~4s | 239687 |
| 1000 | 10 | 100 | 1 | 4~5s | 396874 |




發現交易筆數和運算規則呈現線性成長關係，雖沒有很直接往上增加，但可以發現整體趨勢向上，因此若固定平均每筆交易物品量、物品種類數量，交易數量增加時，整體資料量也會增加，從中能找到相互關係的規則數也會變多，但相對運行時間也會增加許多。

## (四)心得

此項作業初期因程式的關係花費較多時間，之前沒有寫過資料探勘相關的演算法，因此初期上網找資料學習後，也去學校圖書館翻閱相關書籍，雖然過程中因為一些 bug 和語法的關係花了不少精力，但真實學到東西的感覺非常好! WEKA 的操作也讓我學到原來資料分析不是像表面所想的那麼簡單，一個小小的參數變化或門檻值設定的不同，都會影響整體分析的效率和產出。雖然第一次使用相關軟件操作，資料使用的不夠完善，產生關係也不夠準確，但還是從中學習到許多知識，每次實際動手操作改變資料集的型態，觀察不一樣結果之間的關係非常有趣，期望自己的技術能更精進，在未來也能應用在更多不同的資料型態中，分析出更有價值的資訊以供使用。