科目：1071_資料探勘 DATA MINING

學生：楊沛霖

學號：Q36071156

日期：2018-10-23

前言：

1. 本次作業利用不同的工具(自己方法、WEKA)以及資料（老師課堂資料、IBM、Kaggle）使用 FP-growth 來尋找 freqItems 並比較不同工具的結果。
2. 本次作業的 inputData 都放在 use_data 資料夾內，outputData 則放置在 save_data 資料夾。

Topic 1:

先以老師上課資料確認程式運行是否正確。

Min_sup : 2

執行時間 : 1.1854257

使用資料(IBM Quest Data Generator):

FP-growth:

老師講義上的圖(圖 一)與這次程式實作出來的圖(圖 二)

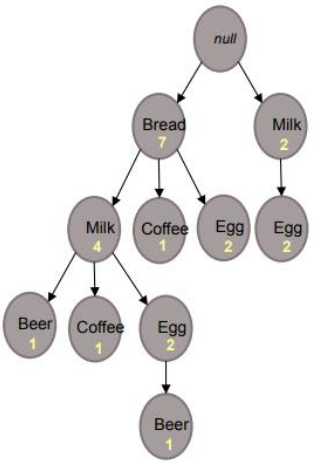


圖 一

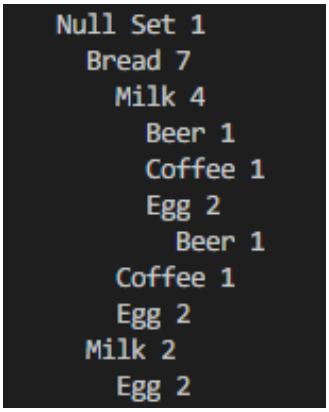
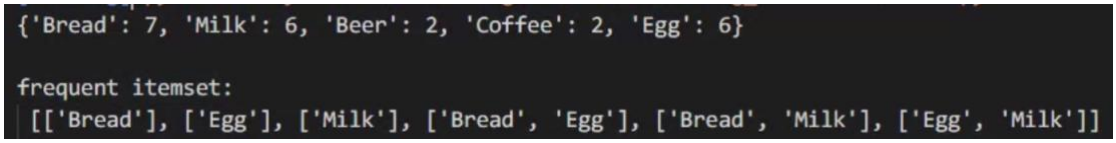


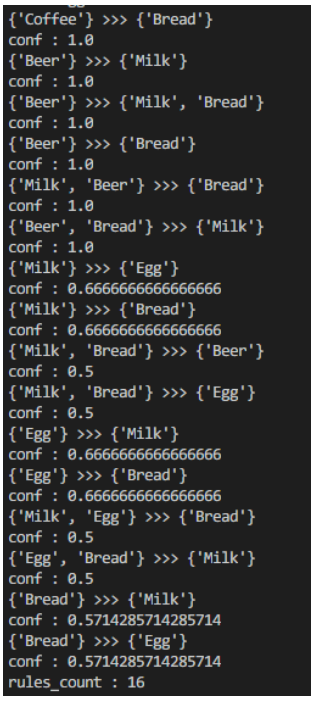
圖 二

Apriori:



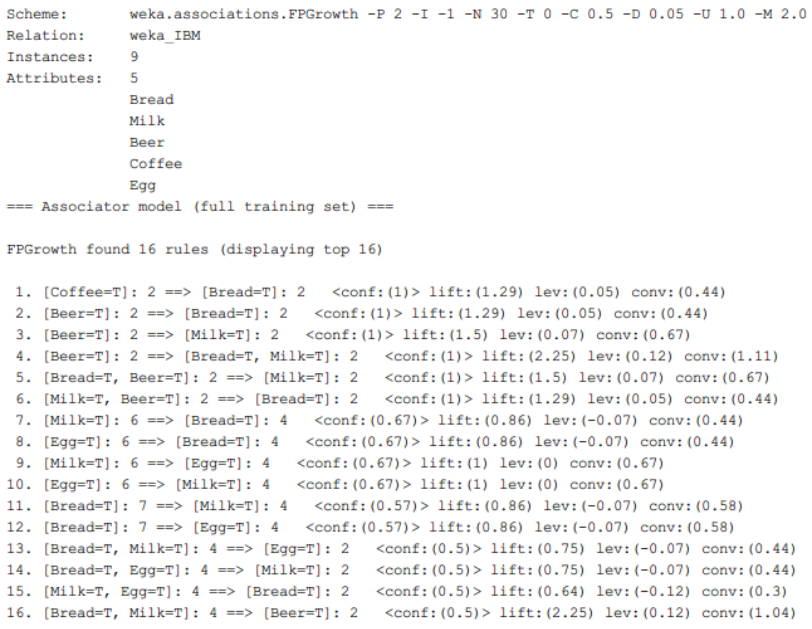
Association analysis:

分別為自己寫的程式(圖 三)與 WEKA 的 FP-tree(圖 四)的關聯顯示,可以看到兩邊答案相同. 對應 function 為 start.py 中的 associate(freqItems, simpDat)



```
{'Coffee'} >>> {'Bread'}
conf : 1.0
{'Beer'} >>> {'Milk'}
conf : 1.0
{'Beer'} >>> {'Milk', 'Bread'}
conf : 1.0
{'Beer'} >>> {'Bread'}
conf : 1.0
{'Milk', 'Beer'} >>> {'Bread'}
conf : 1.0
{'Beer', 'Bread'} >>> {'Milk'}
conf : 1.0
{'Milk'} >>> {'Egg'}
conf : 0.6666666666666666
{'Milk'} >>> {'Bread'}
conf : 0.6666666666666666
{'Milk', 'Bread'} >>> {'Beer'}
conf : 0.5
{'Milk', 'Bread'} >>> {'Egg'}
conf : 0.5
{'Egg'} >>> {'Milk'}
conf : 0.6666666666666666
{'Egg'} >>> {'Bread'}
conf : 0.6666666666666666
{'Milk', 'Egg'} >>> {'Bread'}
conf : 0.5
{'Egg', 'Bread'} >>> {'Milk'}
conf : 0.5
{'Bread'} >>> {'Milk'}
conf : 0.5714285714285714
{'Bread'} >>> {'Egg'}
conf : 0.5714285714285714
rules_count : 16
```

圖 三



```
Scheme:       weka.associations.FPGrowth -P 2 -I -1 -N 30 -T 0 -C 0.5 -D 0.05 -U 1.0 -M 2.0
Relation:     weka_IBM
Instances:    9
Attributes:   5
              Bread
              Milk
              Beer
              Coffee
              Egg
=== Associator model (full training set) ===

FPGrowth found 16 rules (displaying top 16)

 1. [Coffee=T]: 2 ==> [Bread=T]: 2   <conf:(1)> lift:(1.29) lev:(0.05) conv:(0.44)
 2. [Beer=T]: 2 ==> [Bread=T]: 2   <conf:(1)> lift:(1.29) lev:(0.05) conv:(0.44)
 3. [Beer=T]: 2 ==> [Milk=T]: 2   <conf:(1)> lift:(1.5) lev:(0.07) conv:(0.67)
 4. [Beer=T]: 2 ==> [Bread=T, Milk=T]: 2   <conf:(1)> lift:(2.25) lev:(0.12) conv:(1.11)
 5. [Bread=T, Beer=T]: 2 ==> [Milk=T]: 2   <conf:(1)> lift:(1.5) lev:(0.07) conv:(0.67)
 6. [Milk=T, Beer=T]: 2 ==> [Bread=T]: 2   <conf:(1)> lift:(1.29) lev:(0.05) conv:(0.44)
 7. [Milk=T]: 6 ==> [Bread=T]: 4   <conf:(0.67)> lift:(0.86) lev:(-0.07) conv:(0.44)
 8. [Egg=T]: 6 ==> [Bread=T]: 4   <conf:(0.67)> lift:(0.86) lev:(-0.07) conv:(0.44)
 9. [Milk=T]: 6 ==> [Egg=T]: 4   <conf:(0.67)> lift:(1) lev:(0) conv:(0.67)
10. [Egg=T]: 6 ==> [Milk=T]: 4   <conf:(0.67)> lift:(1) lev:(0) conv:(0.67)
11. [Bread=T]: 7 ==> [Milk=T]: 4   <conf:(0.57)> lift:(0.86) lev:(-0.07) conv:(0.58)
12. [Bread=T]: 7 ==> [Egg=T]: 4   <conf:(0.57)> lift:(0.86) lev:(-0.07) conv:(0.58)
13. [Bread=T, Milk=T]: 4 ==> [Egg=T]: 2   <conf:(0.5)> lift:(0.75) lev:(-0.07) conv:(0.44)
14. [Bread=T, Egg=T]: 4 ==> [Milk=T]: 2   <conf:(0.5)> lift:(0.75) lev:(-0.07) conv:(0.44)
15. [Milk=T, Egg=T]: 4 ==> [Bread=T]: 2   <conf:(0.5)> lift:(0.64) lev:(-0.12) conv:(0.3)
16. [Bread=T, Milk=T]: 4 ==> [Beer=T]: 2   <conf:(0.5)> lift:(2.25) lev:(0.12) conv:(1.04)
```

圖 四

比較結果:

程式經過老師範例的洗禮過後,能正常運行並且和分析工具 WEKA 所顯示的資料相同,因此可以加入 IBM 資料去處理了。可以看到程式運作出來的顯示和 WEKA 所出現的 Rule 是一致的，因此程式應該是成功。

Topic 2:

實際運行 IBM Quest Data Generator 所生成的資料

Min_sup : 2

執行時間 : 2.124941

使用資料(IBM Quest Data Generator):

使用參數 :

lit -ntrans 1 -ascii -tlen 5 -nitems 10 -npats 3

圖 五為 IBM Quest Data Generator， 表一 為切換成老師投影片的形式。

```
1 1 247
1 1 506
1 1 578
2 2 247
2 2 506
2 2 578
3 3 247
3 3 506
4 4 247
4 4 506
4 4 578
5 5 247
5 5 506
5 5 578
6 6 247
6 6 506
6 6 578
```

圖 五

TID	ItemSet
1	{247,506,578}
2	{247,506,578}
3	{247,506,247}
4	{247,506,578}
5	{247,506,578}
6	{247,506,578}

表 一

FP-growth:

有程式生成的 FP-growth(圖 六)

```
[['247', '506', '578'], ['247', '506', '578'], ['247', '506'], ['247', '506', '578'], ['247', '506', '578']]
  Null Set 1
    247 5
      506 5
        578 4
```

圖 六

Apriori:

```
{'247': 5, '506': 5, '578': 4}

frequent itemset:
 [['247'], ['506'], ['578'], ['247', '506'], ['247', '578'], ['506', '578'], ['247', '506', '578']]
```

Association analysis:

分別為自己寫的程式(圖 八)與 WEKA 的 FP-tree(圖 七)的關聯顯示,可以看到兩邊答案相同。

```
{'578'} >>> {'247'}
conf : 1.0
{'578'} >>> {'506'}
conf : 1.0
{'578'} >>> {'247', '506'}
conf : 1.0
{'506', '578'} >>> {'247'}
conf : 1.0
{'247', '578'} >>> {'506'}
conf : 1.0
{'247'} >>> {'578'}
conf : 0.8
{'247'} >>> {'506', '578'}
conf : 0.8
{'247'} >>> {'506'}
conf : 1.0
{'506'} >>> {'578'}
conf : 0.8
{'506'} >>> {'247', '578'}
conf : 0.8
{'506'} >>> {'247'}
conf : 1.0
{'247', '506'} >>> {'578'}
conf : 0.8
rules_count : 12
```

圖 八

```
=== Run information ===

Scheme:       weka.associations.FPGrowth -P 2 -I -1 -N 20 -T 0 -C 0.1 -D 0.05 -U 1.0 -M 0.1
Relation:     weka_IBM
Instances:    5
Attributes:   3
              247
              506
              578
=== Associator model (full training set) ===

FPGrowth found 12 rules (displaying top 12)

 1. [506=T]: 5 ==> [247=T]: 5   <conf:(1)> lift:(1) lev:(0) conv:(0)
 2. [247=T]: 5 ==> [506=T]: 5   <conf:(1)> lift:(1) lev:(0) conv:(0)
 3. [578=T]: 4 ==> [506=T]: 4   <conf:(1)> lift:(1) lev:(0) conv:(0)
 4. [578=T]: 4 ==> [247=T]: 4   <conf:(1)> lift:(1) lev:(0) conv:(0)
 5. [578=T]: 4 ==> [506=T, 247=T]: 4   <conf:(1)> lift:(1) lev:(0) conv:(0)
 6. [506=T, 578=T]: 4 ==> [247=T]: 4   <conf:(1)> lift:(1) lev:(0) conv:(0)
 7. [247=T, 578=T]: 4 ==> [506=T]: 4   <conf:(1)> lift:(1) lev:(0) conv:(0)
 8. [506=T]: 5 ==> [578=T]: 4   <conf:(0.8)> lift:(1) lev:(0) conv:(0.5)
 9. [247=T]: 5 ==> [578=T]: 4   <conf:(0.8)> lift:(1) lev:(0) conv:(0.5)
10. [506=T]: 5 ==> [247=T, 578=T]: 4   <conf:(0.8)> lift:(1) lev:(0) conv:(0.5)
11. [247=T]: 5 ==> [506=T, 578=T]: 4   <conf:(0.8)> lift:(1) lev:(0) conv:(0.5)
12. [506=T, 247=T]: 5 ==> [578=T]: 4   <conf:(0.8)> lift:(1) lev:(0) conv:(0.5)
```

圖 七

比較結果:

無論資料量大小皆與 WEKA 的分析結果一致，不過在自己 associate 的寫法中沒辦法像 WEKA 能夠把除了 True 或 False 的資料顯示出來，只能簡單的用存不存在去做對比。

Topic 3:

使用資料(Kaggle):

使用資料來源 ：

https://www.kaggle.com/rush4ratio/video-game-sales-with-ratings

資料目的：

此資料為判斷每種 Video Games 主機銷量與顧客反應，在這裡可以看到不同國家或不同地區的銷售狀況還有評論狀況還有遊戲分數等等資訊。

而我所使用的是它的遊戲主機和用戶評分的來做比較，可以看到由於分數的分數極為零散因此整顆數非常的巨大。

FP-growth（部分）:

經處理後的 WEKA 讀取文件（arff）：

文件名稱 ： @relation

Table Header 屬性 ： @attribute

屬性分為 :string , numberic , nominal, date

分別為 字串, 數字, 自定義屬性（ex. temperature {hot, mild, cool}）, 日期

```
@relation weka_

@attribute Name string
@attribute Platform string
@attribute Year_of_Release numeric
@attribute Genre string
@attribute Publisher string
@attribute NA_Sales numeric
@attribute EU_Sales numeric
@attribute JP_Sales numeric
@attribute Other_Sales numeric
@attribute Global_Sales numeric
@attribute Critic_Score numeric
@attribute Critic_Count numeric
@attribute User_Score numeric
@attribute User_Count numeric
@attribute Developer string
@attribute Rating string
@data
Name,Platform,Year_of_Release,Genre,Publisher,NA_Sales,EU_Sales,JP_Sales,Other_Sales,Global_Sales
Wii_Sports,Wii,2006,Sports,Nintendo,41,29,4,8,8253,76,51,8,322,Nintendo,E
Mario_Kart_Wii,Wii,2008,Racing,Nintendo,16,13,4,3,3552,82,73,83,709,Nintendo,E
Wii_Sports_Resort,Wii,2009,Sports,Nintendo,16,11,3,3,3277,80,73,8,192,Nintendo,E
New_Super_Mario_Bros,DS,2006,Platform,Nintendo,11,9,6,3,298,89,65,85,431,Nintendo,E
Wii_Play,Wii,2006,Misc,Nintendo,14,9,3,3,2892,58,41,66,129,Nintendo,E
```

結果與討論

IBM Data(FP):

Time : 0.04s user 0.02s system 70% cpu 0.088 total

Memory:

```
PID   COMMAND      %CPU  TIME     #TH   #WQ  #PORT MEM    PURG   CMPRS  PGRP PPID STATE    BOOSTS
6246  Python       0.0   00:00.05 1     0    10+   4284K+ 0B     0B     6246 1211 sleeping *0[1+]
```

IBM Data(Apriori):

Time : 0.04s user 0.02s system 67% cpu 0.086 total

Memory: 速度太快拍不到@@

Kaggle Data:

Time : 0.04s user 0.03s system 35% cpu 0.206 total

Memory:

PID	COMMAND	%CPU	TIME	#TH	#WQ	#PORT	MEM	PURG	CMPRS	PGRP	PPID	STATE	BOOSTS
7079	Python	0.0	00:00.06	1	0	10+	4564K+	0B	0B	7079	6408	sleeping	*0[1+]

這裡可以發現由於 Kaggle 所給的資料比 IBM 的還要多因此可以看到在 MEM 的地方，Kaggle 硬是比 IBM 多出 280K 左右的記憶體使用量。而運作時間也是 Kaggle 最慢筆 IBM 多了 0.1 秒左右而 Apriori 最快連 memory 變化都拍不到。不過這裡有一個很有趣的地方，就是雖然 Kaggle 的時間增加了但是在 system cpu 的方面卻比 IBM 少了一半的趴數。

心得:

由於樹的特性很適合利用 LinkList 的方式去寫因此，在建立 Tree 之前我創立了一個 Node 的資料型態，裡面可以放它的 Name、Parents、Children、Value，如此一來建立一棵樹就會變得方便許多。在之後的 FP-Growth 的相關操作方面，把老師投影片的演算法看過一步步照著設計即可。

在 IBM 資料的產生方面由於完全不懂他的運作方式，一開始沒有看網路上解說直接讓他產生資料就跑了半個小時，當初還覺得很厲害，原來這就是為甚麼別人說『訓練資料要訓練很久』，當下還覺得很興奮!但是實際跑過一次程式後發現根本不能跑，還要處理 Memory 的問題，因此上網找了他的相關參數，用了一個可以跑得數據即可。

而且在 WEKA 的使用方面也找了不少網路上的影片 or 資訊去看，看完後發現雖然它的介面沒有很炫泡，但是裡面功能卻是十分完善的，只要將資料換成它所適合的型態再給他的型態做設定就可以幫我們做簡單的資料預測了。