

# Project 1: Association Algorithms

F74109016 資訊系大四 葉惟欣

## 1.1 What do you observe in the below 4 scenarios?

What could be the reason?(For both support and confidence, you should set 0.05 for low and 0.2 or

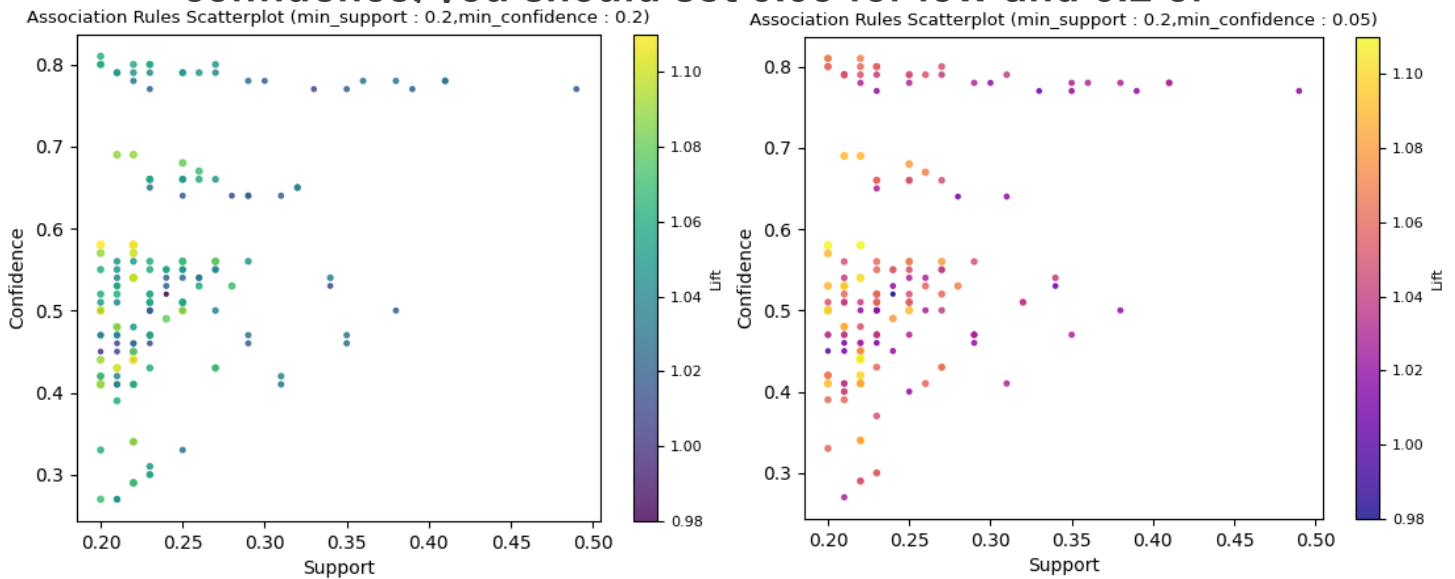


圖 1 : High support, high confidence

圖 2 : High support, low confidence  
(min\_sup = 0.2, min\_conf = 0.05)

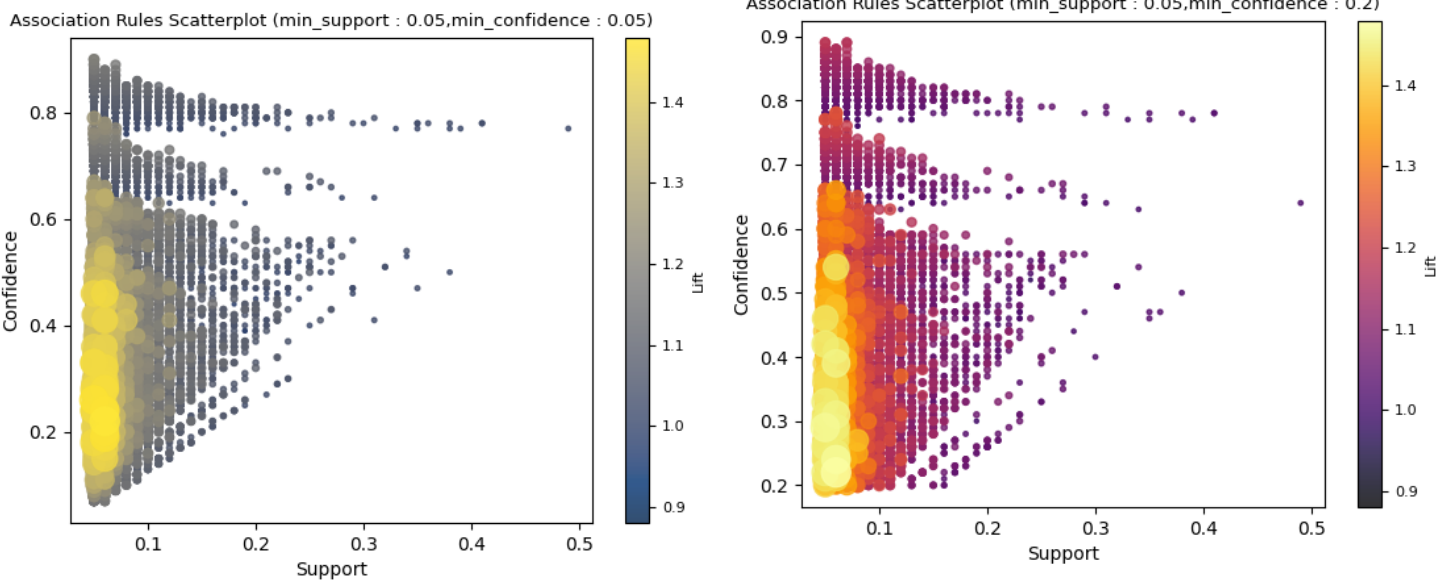


圖 3 : Low support, low confidence  
(min\_sup = 0.05, min\_conf = 0.05)

圖 4 : Low support, high confidence  
(min\_sup = 0.05, min\_conf = 0.2)

觀察 1：

分析不同 minimum support 與 minimum confidence 下跑出的關聯法則的 Support、Confidence、Lift，發現在 minimum support 較低的圖中，

**support 很低的關聯法則會出現，而且這些關聯法則通常存在較高的 Lift(圈圈較大)**，代表有些具高相關性有價值的資訊需透過降低 support 才能找到，且同時也會比較多點聚集在 support 低的地方。在 min-support 高的情況下，通常只會找到普遍卻較沒有資訊意義的關聯法則，所以 Lift 就比較低。

觀察 2：

此外圖中可以看到，四種情況下的關聯法則的分布右下角都是空的，代表**不太存在 support 相對高，但 confidence 低的關聯法則**。所以通常只會有 support 高 confidence 高的關聯法則，或 support 低 confidence 高 support 低 confidence 低的關聯法則則都有可能會出現。

## 1.2 Runtime statistics

Report the run time for both algorithms for the above 4 scenarios in a table.

Try to provide an explanation for the runtime statistics.

### 圖 5 : High support, high confidence (min\_sup = 0.2, min\_conf = 0.2)

```
(myenv) PS D:\GitHub\HWIDM\hw1_example_2023\hw1_example_2023> python main.py --dataset=ibm-2023-released.txt --min_sup=0.2 --min_conf=0.2
2023-10-16 21:15:36,344 [main.py:47] INFO: Arguments: Namespace(min_sup=0.2, min_conf=0.2, dataset='ibm-2023-released.txt')
min-sup : 0.2 min-conf : 0.2
Apriori - elapsed_time : 0.23252058029174805 sec
Apriori - Association-Rules: 326
FP-growth - elapsed_time : 0.4249086380004883 sec
FP-growth - Association-Rules: 326
```

min-sup: 0.2 min-conf: 0.2

Apriori - elapsed\_time : 0.23252058029174805 sec

Apriori - Association-Rules: 326

FP-growth - elapsed\_time : 0.4249086380004883 sec

FP-growth - Association-Rules: 326

### 圖 6 : High support, low confidence (min\_sup = 0.2, min\_conf = 0.05)

```
(myenv) PS D:\GitHub\HWIDM\hw1_example_2023\hw1_example_2023> python main.py --dataset=ibm-2023-released.txt --min_sup=0.2 --min_conf=0.05
2023-10-16 21:16:05,896 [main.py:47] INFO: Arguments: Namespace(min_sup=0.2, min_conf=0.05, dataset='ibm-2023-released.txt')
min-sup : 0.2 min-conf : 0.05
Apriori - elapsed_time : 0.21448540687561035 sec
Apriori - Association-Rules: 326
FP-growth - elapsed_time : 0.4496746063232422 sec
FP-growth - Association-Rules: 326
```

min-sup: 0.2 min-conf: 0.05

Apriori - elapsed\_time : 0.21448540687561035 sec

Apriori - Association-Rules: 326

FP-growth - elapsed\_time : 0.4496746063232422 sec

FP-growth - Association-Rules: 326

### 圖 7 : Low support, low confidence (min\_sup = 0.05, min\_conf = 0.05)

```
(myenv) PS D:\GitHub\HW1DM\hw1_example_2023\hw1_example_2023> python main.py --dataset=ibm-2023-released.txt --min_sup=0.05 --min_conf=0.05
2023-10-16 21:07:01,690 [main.py:47] INFO: Arguments: Namespace(min_sup=0.05, min_conf=0.05, dataset='ibm-2023-released.txt')
min-sup : 0.05 min-conf : 0.05
Apriori - elapsed_time : 9.783414363861084 sec
Apriori - Association-Rules: 59472
FP-growth - elapsed_time : 2.1212029457092285 sec
FP-growth - Association-Rules: 59472
```

min-sup : 0.05 min-conf : 0.05

Apriori - elapsed\_time : 9.783414363861084 sec

Apriori - Association-Rules: **59472**

FP-growth - elapsed\_time : 2.1212029457092285 sec

FP-growth - Association-Rules: **59472**

### 圖 8 : Low support, high confidence (min\_sup = 0.05, min\_conf = 0.2)

```
(myenv) PS D:\GitHub\HW1DM\hw1_example_2023\hw1_example_2023> python main.py --dataset=ibm-2023-released.txt --min_sup=0.05 --min_conf=0.2
2023-10-16 21:14:11,017 [main.py:47] INFO: Arguments: Namespace(min_sup=0.05, min_conf=0.2, dataset='ibm-2023-released.txt')
min-sup : 0.05 min-conf : 0.2
Apriori - elapsed_time : 8.836987972259521 sec
Apriori - Association-Rules: 41374
FP-growth - elapsed_time : 1.7510976791381836 sec
FP-growth - Association-Rules: 41374
```

min-sup : 0.05 min-conf : 0.2

Apriori - elapsed\_time : 8.836987972259521 sec

Apriori - Association-Rules: **41374**

FP-growth - elapsed\_time : 1.7510976791381836 sec

FP-growth - Association-Rules: **41374**

比較 Low support, low confidence 的情形與 Low support, high confidence 的情形，從這裡可以看到 minimum confidence 為 0.05 的效能比 minimum confidence 為 0.2 的效能來低，這很直覺畢竟 low minimum confidence 得到的 Association Rule 會較高。再來比較將 low minimum support 與 high minimum support 的兩兩比較四張圖時可以看到在 high minimum support 時 Apriori 只比 FP-growth 的方法效率還好兩倍。但是當 low minimum support 時 FP-growth 的方法效率還好快四倍以上。

(1) 從下面兩張圖可以看出

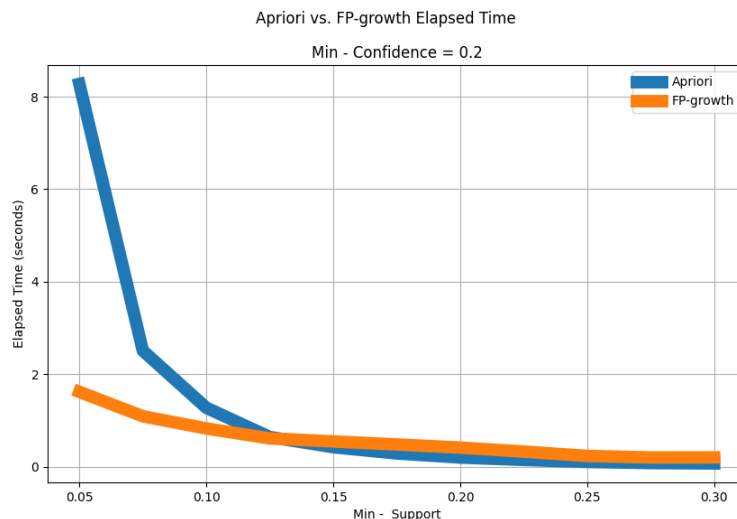


圖 9 : 在固定 Min-Confidence = 0.2 變動 Min-Support(0.05-0.3)情況下 Apriori 與 FP-growth

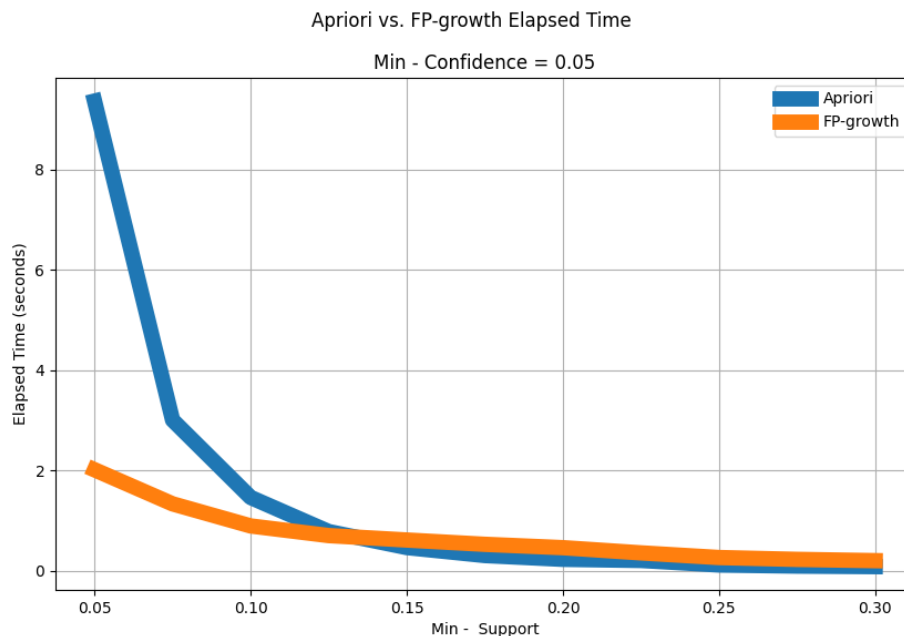


圖 10：在固定 Min-Confidence = 0.05 變動 Min-Sup(0.05-0.3)情況下 Apriori 與 FP-growth

接下來從實作過程分析兩個演算法

(1-1) 透過 Apriori 尋找關聯法則，在將 itemset 不斷聯集的過程中用 min-sup 過濾，逐步建構、擴大 candidate set 的數量。最後求出所有的 frequent itemset 用 min-conf 過濾以尋找關聯法則。再來用 FP-tree 的方法優化 Apriori 尋找關聯法則的方法。首先掃過所有的 transaction order，過程中用 suffix-tree 的概念建立共用 prefix-path 的 FP-tree 以及 Header-Table。Header-Table 會將 items 在 FP-tree 中的位置紀錄在 linked-list、並維護出現的次數。

(1-2) 而 FP-growth 的做法是從 FP-tree 中挖掘 frequent itemset，形同於第二次掃描資料。從 tree 的底部向上遞歸用 min-sup 過濾尋找 frequent itemset，最後建構出所有的 frequent itemset 由 minconf 過濾得所有關聯法則。由於只需掃描整個資料兩次，且建構 frequent itemset 的想法不經由產生 candidate set 得到，因此比 Apriori 頻繁掃描整個資料且還可能產生大量 candidate set 的方法效率高很多。也因此從上面兩個圖都可看到 FP-growth 比 Apriori 的效率好非常多

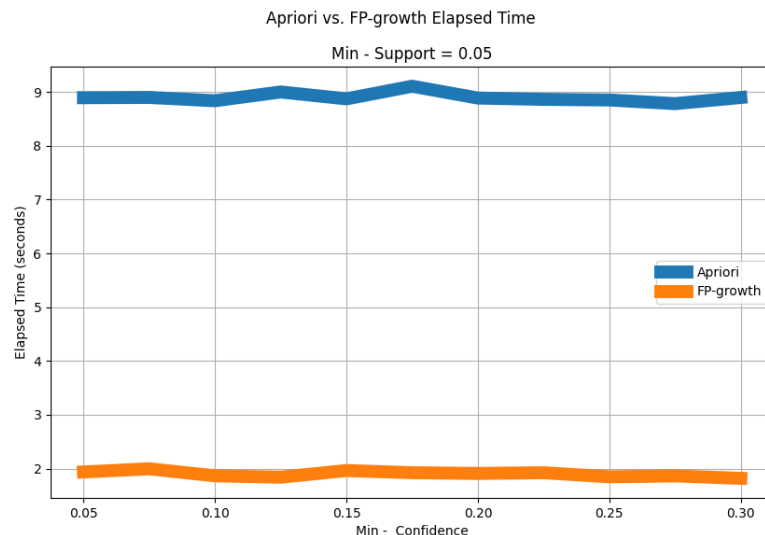


圖 11：在固定 Min-Support = 0.05 變動 Min-Conf(0.05-0.3)情況下 Apriori 與 FP-growth

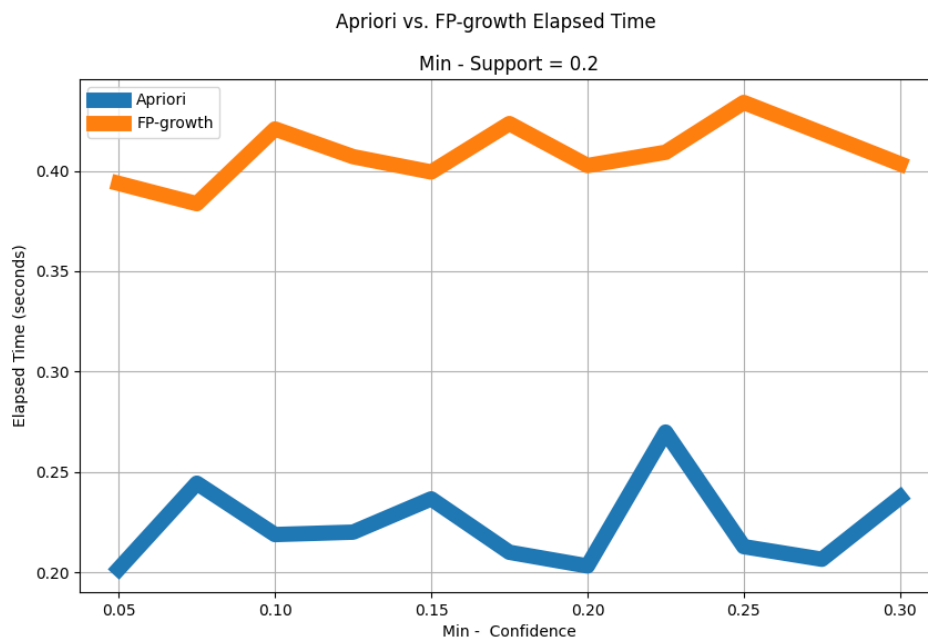


圖 12：在固定 Min-Support= 0.2 變動 Min-Conf(0.05-0.3)情況下 Apriori 與 FP-growth

(2) 從下面四個圖表分析來看

比較第三與第四張圖比較可以發現 minimum support 低(0.05)的時候 FP-growth 算得比 Apriori 較快而 minimum support 高時(0.2)的時候 Apriori 算的比 FP-Growth 快。分析理由如下：

因為 FP-Growth 需要建立 Tree。如果在產出 Association Rule 不多的情況下，建立樹，再建立 Frequent item Set 爬樹的過程的 cost 會大於 FP-growth 最後建立 Association Rule 帶來的效益。所以當 minimum support 高能建立的 Association Rule 相對來說比較少，可以由圖 5 與圖 6 中看到，也因此 FP-growth 的優勢無法在其中凸顯出來，而當圖 7 與圖 8 由於 Association Rule 非常多可看到 FP-growth 的優勢在效能上提升非常多。

## 1.3 Any topics you are interested in

### 1. 效率方面的優化

在實作的過程中我原先 FP-Growth 的效能跑的非常慢。後來我開始嘗試優化 FP-growth 的效能。當我發現我原先使用的是 list 去裝所有的 frequent item set，後來改成使用 set 代替 list 來存儲 frequent item set 與 conditional Pattern Base。set 在查找操作上比 list 更快。後來我跟同學討論，同學說 Searching in a set - can take anything between linear time over input to logarithmic time over input to almost constant time over the input length. 使用 set 而不是 list 時，通常會提高 search 的速度，因為 set 的查找複雜度是常數數量級  $O(1)$ ，而 list 的複雜度是線性數量級  $O(n)$ ，其中  $n$  是 list 的長度。而這項優化讓 FP-Growth 的執行效能提升了五倍

```

59472 rules
Apriori - elapsed_time : 10.675127029418945 sec
2993 minSup : 149.65
FP-growth - elapsed_time : 154.81591963768005 sec
rules: 59472

```

圖 13：在原先未使用 set 優化時的效能

由於後來的效能雖提升五倍，變成三十秒多，仍然比 Apriori 慢，後來我發現我在建立 Association Rule 的時候我先把 Frequent Item Set 找出來後去在去爬所有原先的 Item Set List，但我發現如此一來完全沒有建立 FP-Tree 的意義。所以後來我在爬樹的過程順便把每個 Frequent Item Set 的值的 support 建立好。如果原先沒有在 Frequent Item Set 的 Set 我則再用把 Frequent Item Set 拿去爬所有原先的所有的 Item Set 的 List 的方式去計算 support 的值。最後我也建立一個 dictionary 用來方便快捷得到 support 的值。為什麼特別想到要優化這一步的原因是因為我把尋找 Association Rule 的函數先註解掉發現效能到個位數的秒數，就開始覺得計算 support 的方式有問題了。

## 2. Support 與 Confidence 以及 Lit 的關係

Support (A & B)



$$\text{Confidence (A} \rightarrow \text{B)} = \frac{\text{Support (A \& B)}}{\text{support(A)}}$$

$$\rightarrow \text{Support(A)} = \frac{\text{Support (A \& B)}}{\text{confidence(A} \rightarrow \text{B)}}$$























$$\text{Confidence (B} \rightarrow \text{A)} = \frac{\text{Support (A \& B)}}{\text{support(B)}}$$

$$\rightarrow \text{Support(B)} = \frac{\text{Support (A \& B)}}{\text{confidence(B} \rightarrow \text{A)}}$$

$$\begin{aligned} \text{Lift} &= \frac{\text{Support (A \& B)}}{\text{Support(A)} * \text{Support(B)}} \\ &= \frac{\text{Confidence (A} \rightarrow \text{B)} * \text{Confidence (B} \rightarrow \text{A)}}{\text{Support(A \& B)}} \end{aligned}$$

圖 14：自製 Support, Confidence, Lit 關係圖

而我在分析 Support、Confidence、Lift 三者關係的過程中，發現關聯法則(antecedent->consequent) 的 confidence 會高有可能是因為 antecedent 出現次數少，而 consequent 常出現導致。因為 Confidence 為(antecedent 與 consequent 的 Support)除上(antecedent 的 Support)。antecedent 出現次數少導致分母小，所以除完的結果得到 Confidence 較大。

Transaction 1	   
Transaction 2	  
Transaction 3	 
Transaction 4	 
Transaction 5	   
Transaction 6	  
Transaction 7	 
Transaction 8	 

Sup :  $3/8 = 0.375$

Conf 1 :  $(3/8) / (4/8) = 0.75$

蘋果對beer信賴度高

Conf 2 :  $(3/8) / (6/8) = 0.5$

Beer對apple信賴度不高

Lift :  $(3/8) / ((4/8)*(6/8)) = 1$

Beer跟apple沒關係

Sup 1&2:  $4/8 = 0.5$

Conf 12 :  $(4/8) / (6/8) = 0.66$

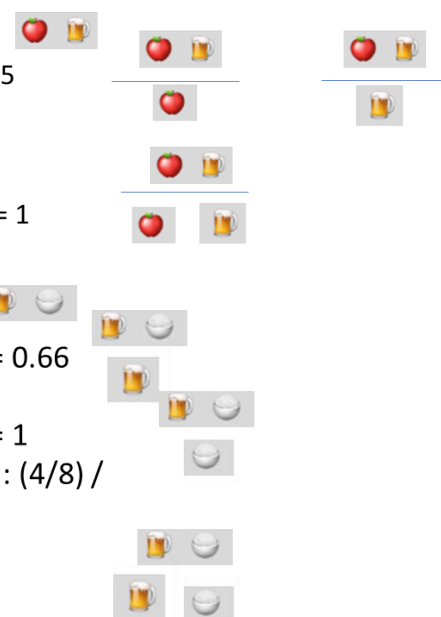
Beer對飯信賴度高

Conf 21 :  $(4/8) / (4/8) = 1$

飯對Beer信賴度高Lift :  $(4/8) /$

$((6/8)*(4/8)) = 4/3$

Beer跟飯有關係



透過這個例子發現觀察蘋果與啤酒的關係，蘋果由於很少出現，啤酒常出現，蘋果->啤酒會有高 confidence (0.75)但不一定高關聯，因為是啤酒出現太多次了，所以導致有蘋果就會常常看到啤酒。但反過來就不一定，所以要看真正的關聯性要除掉啤酒的數量  $\text{confidence} / \# \text{ of 啤酒} = \text{lift}$ ，這也就是為什麼要算 Lift 的原因因為要再去確認啤酒的數量，以排除是啤酒數量多的關係。

而像飯對啤酒的 Confidence 高，啤酒對飯的 Confidence 也很高，也因此兩者的 Lift 也很高。因為通常啤酒出現飯也會出現。而不是因為某一個項目高頻繁出現而導致有些物品一出現就會跟他重複出現。

換句話說這是因為 consequent 頻繁出現導致有 antecedent 就有 consequent 的機率高。然而有 consequent 則有 antecedent 的機率則不一定也同樣高。這代表一個高 confidence 的關聯法則，並不表示兩者間高度相關。要看兩個 itemset 的關聯性還需將關聯法則的 confidence 除掉 consequent 的 support，相當於排除 consequent 數量多的因素。而這也就是 Lift 的意義。