關聯規則探勘

李啟龍 博士

尿布啤酒傳說

數據分析領域有一個**「尿布啤酒**傳說**」**，就是美國大型超市Walmart的員工分析銷售資料時發現，星期五晚上常有尿布與啤酒同時購買的情形。原來是因為這個時段常有年輕父親來到超市，幫家裡的新生兒買尿布，同時為自己買啤酒。因此Walmart超市便將兩種看似毫不相關的商品一同陳列販售，最終也成功提升了銷售量。

過去幾乎沒有人看出這件事，Walmart的員工又是怎麼發現的？其實大多數人去超級市場採買時都有些規律，例如家庭主婦買菜的時候可能會順便買肉，當這樣的現象很普遍時，大家也都習以為常。既然從購物籃看不出端倪，那就比較每個消費者的購物清單，看看哪些東西多數人會同時買。這就是知識發現的其中一種手段，稱為關聯規則探勘**(association rule mining)**，而這個情境又被稱作購物籃分析(market basket analysis)。

至於怎麼具體的描述兩個元素的關係，我們可以用數學的形式表示，用幾個指標數值判定這個關聯規則的強弱。最直觀的問題就是**「有尿布的購物清單裡，同時有買啤酒的比例是多少？」，也就是算有尿布的清單裡有啤酒的條件機率，可以用「同時有尿布跟啤酒的購物清單」除以「有尿布的購物清單」求得。這個數值有個專有名詞，叫做信賴度(confidence)**，代表的是一筆資料中出現目標元素A時，同時出現目標元素B的比率。

假設A是某交易資料庫的項目之一，B是另一個項目，「A->B」為一條關聯規則，其信賴度的計算如下：

$$\text{信賴度} = \frac{A\text{與}B\text{同時出現的資料筆數}}{A\text{出現的資料筆數}}$$

如果兩者同時出現的資料筆數越高，信賴度通常也不低。可以注意到如果反過來問**「有啤酒的購物清單裡，同時有買尿布的比例是多少？」**時，信賴度分子不會變，但分母可能不同，所以兩個方向的信賴度通常不會一樣。

若「B->A」為一條關聯規則，其信賴度的計算如下：

$$\text{信賴度} = \frac{B\text{與}A\text{同時出現的資料筆數}}{B\text{出現的資料筆數}}$$

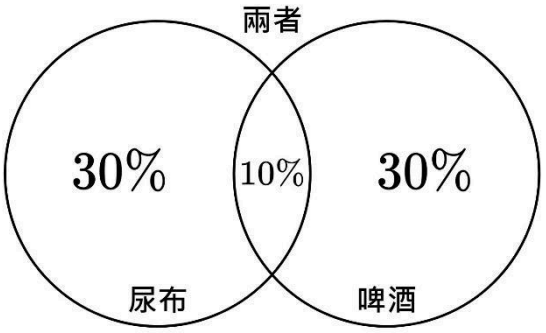
或許是因為買尿布的人比較少，同時買啤酒的比例才會比較高？信賴度固然是個簡單好用的指標，但其中一種元素在母群資料中的出現次數較少時，雖然因為分母較小，會得到比較高的信賴度，但也會讓關聯分析容易在整群資料中的代表性不足。我們需要再參考另一項指標，也就是兩種元素同時出現的資料比數在母群體中的比例，稱為支持度**(support)**，也就是此關聯規則的代表性強度。同樣地，「A->B」為一條關聯規則，其支持度的計算如下：

$$\text{支持度} = \frac{A\text{與}B\text{同時出現的資料筆數}}{\text{總資料筆數}}$$

另外，有時候也會參考提升度**(lift)**這項指標。提升度代表的是兩元素在出現機率上的相關性。若提升度為1，根據數學定義兩者的出現與否為獨立事件；若提升度大於1表示兩個事件成正相關，反之則成負相關。

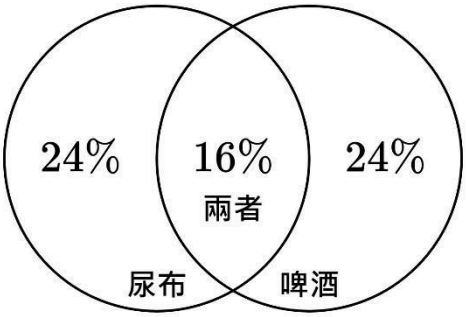
$$\text{提升度}=\frac{A\text{與}B\text{同時出現的比率}}{A\text{出現的比率}\times B\text{出現的比率}}$$

提升度的想像比較不直覺，我們用以下三張文氏圖來進行說明。我們舉個例子，假設賣場的所有購物單據中，40%的清單有購買尿布，同時也有40%的清單購買啤酒。



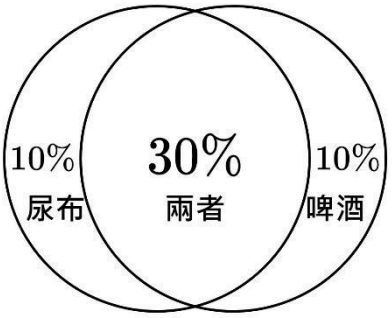
$$\text{提升度}=\frac{0.1}{0.4\times\ 0.4}=0.625\Rightarrow\textbf{負相關}$$

當僅10%的清單同時購買尿布和啤酒，提升度算下來只有0.625。這個狀況下，如果得到的結論是「買尿布就不太可能買啤酒」或「買啤酒就不太可能買尿布」都算合理，所以兩項商品的銷售可以說是呈現負相關。



$$\text{提升度}=\frac{0.16}{0.4\times\ 0.4}=1\Rightarrow\textbf{獨立事件}$$

當16%的清單同時購買尿布和啤酒，提升度恰好是1。交集機率(0.16)等於個別機率相乘(0.4×0.4)時為數學上認定的獨立事件，也就是購買啤酒和尿布互不影響，沒有明顯的正相關或負相關。



$$\text{提升度}=\frac{0.3}{0.4\times\ 0.4}=1.875\Rightarrow\textbf{正相關}$$

當高達30%的清單同時購買尿布和啤酒，提升度為1.875。這個狀況下，得到的結論就會是「買尿布就很有可能買啤酒」或「買啤酒就很有可能買尿布」，所以兩項商品的銷售呈現正相關。

□ 先驗演算法

當信賴度、支持度與提升度夠高時，這項關聯規則便具有足夠的正確性與代表性，兩個指標也衍伸出關聯規則探勘中相當經典的先驗演算法**(Apriori Algorithm)**。

這個算法一開始會先對三項指標訂一個門檻值，再從元素母群體中，枚舉任兩個互無交集的資料子集合，計算兩集合間的信賴度、支持度與提升度，當三項指標都比門檻值高，這兩個集合就會被探勘為潛在的關聯規則。注意這裡比較的是**「兩集合」而非「兩元素」，因為有些關聯規則從細項不好觀察，例如「買菜的時候順便買肉品」的規律性，會比「買空心菜的時候順便買豬肉」來得顯而易見。**

範例：購物籃分析

編號	購物清單紀錄
1	牛奶、麵包、果醬
2	蛋、豆漿
3	麵包、果醬、蛋
4	麵包、蛋
5	牛奶、麵包
6	麵包、蛋

假設上表是某個賣場的數筆交易紀錄，如果我們將信賴度、支持度與提升度的最低門檻分別定為30%、30%與1.2時，由先驗演算法我們可以探勘得**「買麵包時會買牛奶」、「買麵包時會買果醬」、「買牛奶時會買麵包」、「買果醬時會買麵包」四項待驗證的資訊。**

食品A	食品B	信賴度	支持度	提升度
麵包	牛奶	$\frac{2}{5}=40\%>30\%$	$\frac{2}{6}\cong 33\%>30\%$	$\frac{\frac{2}{6}}{\frac{5}{6}\times\frac{2}{6}}=1.2$
麵包	果醬	$\frac{2}{5}=40\%>30\%$	$\frac{2}{6}\cong 33\%>30\%$	$\frac{\frac{2}{6}}{\frac{5}{6}\times\frac{2}{6}}=1.2$
牛奶	麵包	$\frac{2}{2}=100\%>30\%$	$\frac{2}{6}\cong 33\%>30\%$	$\frac{\frac{2}{6}}{\frac{2}{6}\times\frac{5}{6}}=1.2$
果醬	麵包	$\frac{2}{2}=100\%>30\%$	$\frac{2}{6}\cong 33\%>30\%$	$\frac{\frac{2}{6}}{\frac{2}{6}\times\frac{5}{6}}=1.2$

從表格可以發現「買麵包時會買蛋」或是「買蛋時會買麵包」也看似是個潛在趨勢，但因為其提升度僅有0.9，故在這個門檻下不易被篩選出來。

食品A	食品B	信賴度	支持度	提升度
麵包	蛋	$\frac{3}{5}=60\%>30\%$	$\frac{3}{6}\cong 50\%>30\%$	$\frac{\frac{3}{6}}{\frac{5}{6}\times\frac{4}{6}}=0.9<1.2$
蛋	麵包	$\frac{3}{4}=75\%>30\%$		

這樣的演算法用程式要怎麼實現？先驗演算法本質上就是子集合的枚舉問題，不用太複雜的遞迴函式就能寫出來。不過Python有個名為apyori的套件，可以直接套用函式運算，那我們就進行一個簡單的實作。

程式實作：先驗演算法函式運用

【參考檔案】Apriori.ipynb
【訓練資料檔案】Market_Basket_Optimisation.csv

```
1 # 匯入套件
2 import pandas as pd
3 from apyori import apriori
```

讀入試算表並觀察資料

我們將使用的是機器學習網站Kaggle上的Market Basket Optimization資料集，更多的相關研究可以參考下方的網址。儲存資料的試算表已在章節實作的壓縮檔中，先用pandas套件讀入csv中的資料集並輸出觀察，可以發現每一列就是一筆筆長短不一的購物清單。這份試算表沒有標題列，故要指定header = None參數才能取用第一筆資料。

資料集來源：
https://www.kaggle.com/datasets/hemanthkumar05/market-basket-optimization

```
1 df = pd.read_csv('Market_Basket_Optimisation.csv', header =
  None)  # 讀入csv檔
2 df  # 輸出試算表
```

輸出

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	shrimp	almonds	avocado	vegetables mix	green grapes	whole weat flour	yams	cottage cheese	energy drink	tomato juice	low fat yogurt	green tea	honey	salad	mineral water	salmon	antioxydant juice
1	burgers	meatballs	eggs	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	chutney	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	turkey	avocado	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	mineral water	milk	energy bar	whole wheat rice	green tea	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
7496	butter	light mayo	fresh bread	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7497	burgers	frozen vegetables	eggs	french fries	magazines	green tea	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7498	chicken	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7499	escalope	green tea	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7500	eggs	frozen smoothie	yogurt cake	low fat yogurt	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

7500 rows × 20 columns

資料預處理

再來將這個試算表轉為二維list。因為不是每個購物清單都一樣長，所以要將空項(nan)去除。

```
1 # 資料預處理
2 data = df.values.tolist()
3 data = [[x for x in row if str(x) != 'nan'] for row in data]
```

資料探勘

將預處理後的資料集傳入apyori套件的apriori函式，就能得到對應的結果物件，此處我們可以調整三個門檻值的大小。我們輸出其中一組潛在規則，看看它的資料結構。

```
# apriori函式運算
associations = apriori(data, min_support = 0.01,
                       min_confidence = 0.4, min_lift = 1.5)
rules = list(associations)
print(rules[0])
```

輸出

```
RelationRecord(items=frozenset({'ground beef', 'mineral water'}),
support=0.040933333333333335,
ordered_statistics=[OrderedStatistic(items_base=frozenset({'ground
beef'}), items_add=frozenset({'mineral water'}),
confidence=0.41655359565807326, lift=1.7482663499919135)])
```

探勘規則呈現

每組潛在規則的資料結構如下。

- RelationRecord物件
  - items元素集合
  - support支持度
  - ordered_statistics陣列
    - OrderedStatistics物件
      - items_base子集合A
      - items_add子集合B
        (items_base + items_add = items)
      - confidence信賴度
      - lift提升度

對於對應的資料，我們也要有存取對應的變數，才能整理各個潛在規則並清楚地呈現。

```
# 輸出潛在知識
for rule in rules:
    for order_stat in rule.ordered_statistics:
        set_A = set(order_stat.items_base)
        set_B = set(order_stat.items_add)
        if len(set_A) == 0 or len(set_B) == 0:
            continue
        print(f'{set_A} => {set_B}')
        print((f'Confidence: {order_stat.confidence :.4f}'
               f' Support: {rule.support :.4f}'
               f' Lift: {order_stat.lift :.4f}'))
```

輸出

```
{'ground beef'} => {'mineral water'}
Confidence: 0.4166 Support: 0.0409 Lift: 1.7475
```

```
{'olive oil'} => {'mineral water'}
Confidence: 0.4190 Support: 0.0276 Lift: 1.7579
{'salmon'} => {'mineral water'}
Confidence: 0.4013 Support: 0.0171 Lift: 1.6833
{'soup'} => {'mineral water'}
Confidence: 0.4565 Support: 0.0231 Lift: 1.9150
{'chocolate', 'eggs'} => {'mineral water'}
Confidence: 0.4056 Support: 0.0135 Lift: 1.7017
{'chocolate', 'ground beef'} => {'mineral water'}
Confidence: 0.4740 Support: 0.0109 Lift: 1.9885
{'chocolate', 'milk'} => {'mineral water'}
Confidence: 0.4357 Support: 0.0140 Lift: 1.8278
{'chocolate', 'spaghetti'} => {'mineral water'}
Confidence: 0.4048 Support: 0.0159 Lift: 1.6981
{'ground beef', 'eggs'} => {'mineral water'}
Confidence: 0.5067 Support: 0.0101 Lift: 2.1256
{'milk', 'eggs'} => {'mineral water'}
Confidence: 0.4242 Support: 0.0131 Lift: 1.7798
{'frozen vegetables', 'milk'} => {'mineral water'}
Confidence: 0.4689 Support: 0.0111 Lift: 1.9672
{'frozen vegetables', 'spaghetti'} => {'mineral water'}
Confidence: 0.4306 Support: 0.0120 Lift: 1.8065
{'ground beef', 'milk'} => {'mineral water'}
Confidence: 0.5030 Support: 0.0111 Lift: 2.1103
{'ground beef', 'mineral water'} => {'spaghetti'}
Confidence: 0.4169 Support: 0.0171 Lift: 2.3947
{'ground beef', 'spaghetti'} => {'mineral water'}
Confidence: 0.4354 Support: 0.0171 Lift: 1.8265
{'spaghetti', 'milk'} => {'mineral water'}
Confidence: 0.4436 Support: 0.0157 Lift: 1.8610
{'olive oil', 'spaghetti'} => {'mineral water'}
Confidence: 0.4477 Support: 0.0103 Lift: 1.8781
{'spaghetti', 'pancakes'} => {'mineral water'}
Confidence: 0.4550 Support: 0.0115 Lift: 1.9089
```

可以調整apriori函式傳入的三個門檻值，看看會篩選出什麼樣的規則。當門檻值調太低時，會發現函式的運算比預期來得久。這是因為先驗演算法的執行時間是指數量級，可行的子集合大量變多，花費時間可能劇幅上升，以致後來出現不少改良先驗演算法，效率更高的關聯規則探勘手段。