

# Data Mining

project1 Association Analysis

孫啟慧

P78063033

# 一、實驗設計

我用 JAVA 實作了 Apriori 演算法，其中分為使用 Set 和 List 實現的暴力法，以及使用 FP-growth 來進行優化的 Apriori。利用 IBM Quest Synthetic Data Generator 生成的使用者交易資料來模擬實際交易，並作為 input 來分別測試兩種不同的 Apriori 演算法的性能差異。我還選取了 Kaggle 中 NHL Game Data 的 game.csv，通過運行自己寫的 Apriori 在其上，以生成相應的 association rule，來分析各個賽季的成績受哪些因素的影響。最後我們使用 WEKA 對以上的實驗資料集進行分析，在相同的 support 和 confidence 設定之下，得到相應的 association rule，從而對自己撰寫的 Apriori 程式的結果進行驗證。

## I. Input 的預處理：

### 1. IBM Quest Synthetic Data Generator 生成的資料。

該資料生成器包含多個指令，具體的預設參數顯示如下：

```
C:\Users\user\Downloads\IBM-Quest-Data-Generator.exe>"IBM Quest Data Generator.exe" /it -help
Command Line Options:
-ntrans number_of_transactions_in_000s (default: 1000)
-tlen avg_items_per_transaction (default: 10)
-nitems number_of_different_items_in_000s (default: 100)

-npats number_of_patterns (default: 10000)
-patlen avg_length_of_maximal_pattern (default: 4)
-corr correlation_between_patterns (default: 0.25)
-conf avg_confidence_in_a_rule (default: 0.75)

-fname <filename> (write to filename.data and filename.pat)
-ascii (default: True)
-randseed # (reset seed used generate to x-acts; must be negative)
-version (to print out version info)
```

生成的資料格式如下，包含多列的空格。

IBM-Quest-data.data - 記事本			
檔案(F)	編輯(E)	格式(O)	檢視(V) 說明(H)
	1	1	300
	1	1	3671
	1	1	6817
	1	1	8893
	1	1	9145
	1	1	10294
	1	1	12763
	1	1	12850
	1	1	15907
	1	1	30037
	1	1	32192
	1	1	35593
	1	1	35780
	1	1	37680
	1	1	40076
	1	1	42739
	1	1	46867
	2	2	6686
	2	2	9062
	2	2	13707
	2	2	14862
	2	2	18596
	2	2	23700

其中第一欄是 transaction id，第二欄是 customer id，第三欄則是商品 id。因為相同的 transaction id 和 customer id 代表同一個一個人的同一筆 transaction，從而我有寫一個處理資料的 InputProcess.java 文件（\Apriori Brute\src\InputProcess.java）對資料進行處理，處理後的資料如 IBM.data 所示，即每一列代表一次 transaction，用逗號分割的各個數字分別代表不同種類的商品 id，不同列是不同的 transaction。

IBM.data - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明(H)

300,3671,6817,8893,9145,10294,12763,12850,15907,30037,32192,35593,35780,37680,40076,42739,46867  
6686,9062,13707,14862,18596,23790,24082,24160,25493,30037,30830,39673,40507,45753,47668,47990,48281,49415,51207,51732,53710,55232  
1405,2912,3667,4341,9355,20657,25522,27901,28331,29285,36859,39386,41117,41754,42549,43554,44565,47376,49327,49482,52773,56677  
335,816,2743,2760,6563,6719,11914,19607,22023,24927,25231,26274,30598,35043,39145,41899,50670,57448,57519  
5899,8243,12470,18228,18521,19294,24622,39548,43295  
1321,1462,3175,4911,7059,13411,21411,21694,22090,25619,25821,25846,26067,31342,31595,35425,38759,40528,41427,41857,42625,49438,518  
1096,5365,7405,10254,16492,17817,18379,19660,20289,33024,35029,41130,42781,46337,52794,54537,56402

## 2. Kaggle 的 NHL Game Data 的 game.csv

Kaggle 作為一個資料建模和資料分析的競賽平臺，上面經常會有研究人員和企業發起不同的比賽，通過他們提供的資料，參賽者可以組隊參加比賽對其進行建模，提交的各個模型會得到其準確率的評判，從而決定出相應的名次。

我選取了 NHL Game Data 比賽中提供的 game.csv，來作為相應的 input data。該 csv 包含 game\_id, season, season type, away\_team\_id, home\_team\_id, outcome, home\_rink\_side\_start 等項目，由於資料類型較為複雜，故我只選取了 season, outcome, home\_rink\_side\_start 三個欄位，對其進行預處理之後如 game1.csv 所示。其中 season 代表相應的賽季，outcome 則是表示隊伍贏得比賽的情況，其中 home 代表主場隊伍獲勝，而 away 則是客場隊伍獲勝。Home\_rink\_side\_start 則是代表主場隊伍相較於 Time/Score keepers 的相對位置。

通過對預處理過後的 game1.csv 進行相應的關聯規則的分析，我期望可以得到每個賽季中相對位置對勝率的影響。

\*因為第一種 input 的原始資料分隔是空格，另外一個則是逗號，故寫的 Apriori 中資料的讀取部分有所調整，對

於兩種分隔形式均可識別。即整個程式的 input 既可以是.csv 也可以是用空格分割的資料形式。

### 3. Weka

Weka 是由紐西蘭懷卡托大學用 JAVA 開發的資料採擷軟體，我有參考網上的視頻嘗試著玩了一下，且 UI 介面可以很容易上手。雖然其他功能可以吃 csv 檔案，但是 association rule 則只能吃 ARFF 格式的檔案，好在 weka 本身有提供兩種方法來進行格式的轉換，一種是在 weka 的主功能表中可以找到“Simple CLI”模組，根據相應的命令列即可完成檔案類型的轉換。其指令為：

```
java weka.core.converters.CSVLoader filename.csv > filename.arff
```

而另外一種方法則是 weka3.5 之後的版本有提供“Arff Viewer”的模組，可以對相應的 csv 檔案進行流覽，然後可以選擇另存為 ARFF 檔。通過設定相同的參數可以對我自己實現的 Apriori 演算法和 FP-growth 演算法生成的 association rule 進行驗證。

## II. Brute 的 apriori :

演算法設計：

Step 1. 讀入 input，得到 1 item 的頻繁子集，並計算相應的 count 將其保存到 ArrayList 當中。

Step2. 迴圈遍歷 Step 1 中的得到的 ArrayList 當中，

通過判斷該子集的任兩個 itemset 是否相等來決定是否要 merge 相應的兩個 itemset，經過迴圈連結  $k-1$  次得到最終的  $k$  items 頻繁子集。

Step3. 對於 Step2 中最終生成的  $k$  items set 是否滿足最小 minSup (support count) 進行判斷，小於 minSup 的頻繁子集將會被刪掉。接著通過計算各子集之間的 confidence，並和給定的 minConf (min confidence) 進行比較，大於 minConf 的子集將會組成 association rule 進行輸出。

### III. 應用 Fp-growth 的 Apriori :

演算法設計：

Step 1. 讀入 input，對其進行掃描，生成 1 item 的頻繁子集，按照出現的次數降冪排列。如果次數一樣則按照 item 的字典序排列。

*\*為了減少演算法運行的 overhead，根據 Apriori 定理，非頻繁項的父集一定不是頻繁集，故這裡小於 minsSup (min support count) 的選項，我直接將之從保存 1 item 頻繁子集的 linked list 中刪除。*

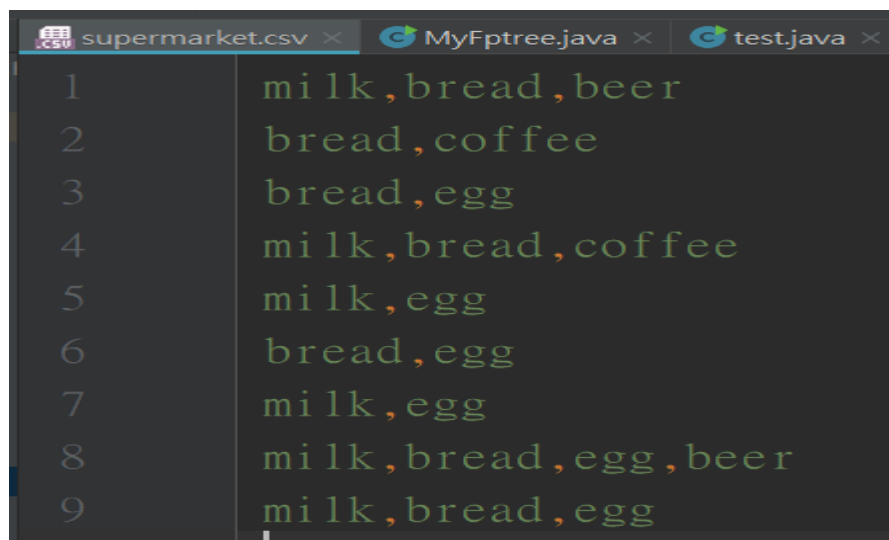
Step 2. 再次掃描 input，對於出現在 step 1 中生成的 1 item 頻繁子集的項按其順序重新排列，並建立 FpTree。首先設置 root 節點為 null，檢查該項中的元素是否有在 FpTree 中，若沒有則將該項加入到樹中，

若已經有分支存在則判斷新的項是否跟分支中的各個點重合，若重合則只更新相應點的 support，否則將創建新的點。同時這些點將會加入到 Step 1 生成的 linked list 當中去。

Step 3. FpTree 建立之後，從尾部掃描 Step1 中建立的 linked list，通過找尋其 homonym 節點和父節點，則可以生成所有的頻繁項集。

Step4. 對於每個頻繁項集分別求其 proper subset（真子集，即比其小的子集合），通過計算相應的 confidence，根據預先定義的 minConf 可以生成最後的關聯規則。

為了驗證所寫程式的正確性，我在調試的時候，以老師 moodle 中給的 fpgrowth 的投影片做 input（\Apriori fp-growth\src\supermarket.csv）



```
supermarket.csv x  MyFptree.java x  test.java x
1    milk,bread,beer
2    bread,coffee
3    bread,egg
4    milk,bread,coffee
5    milk,egg
6    bread,egg
7    milk,egg
8    milk,bread,egg,beer
9    milk,bread,egg
```

將 minsupport 設定為 3，minconfidence 設定為 0.25

可以得到如下頻繁子集和關聯規則。

```
"C:\Program Files\Java\jdk-9.0.1\bin\java" "-javaagent:D:\Program\IntelliJ IDE
=====print frequent set=====
coffee,bread 2
beer,bread 2
milk,bread 3
milk,egg 4
bread,milk,egg 2
egg,bread 4

=====print association rules=====
Rules:{egg => bread} Confidence:0.6666666666666666
Rules:{bread => egg} Confidence:0.5714285714285714
Rules:{milk => bread} Confidence:0.6
Rules:{bread => milk} Confidence:0.42857142857142855
Rules:{milk => egg} Confidence:0.8
Rules:{egg => milk} Confidence:0.6666666666666666
Rules:{milk => egg,bread} Confidence:0.4
Rules:{egg,bread => milk} Confidence:0.5
Rules:{coffee => bread} Confidence:1.0
Rules:{bread => coffee} Confidence:0.2857142857142857
Rules:{bread => egg,milk} Confidence:0.2857142857142857
Rules:{bread => beer} Confidence:0.2857142857142857
Rules:{milk,bread => egg} Confidence:0.6666666666666666
Rules:{egg => milk,bread} Confidence:0.3333333333333333

Process finished with exit code 0
Compilation completed successfully in 1s 801ms (a minute ago)
```

```
Rules:{bread => beer} Confidence:0.2857142857142857
Rules:{milk,bread => egg} Confidence:0.6666666666666666
Rules:{egg => milk,bread} Confidence:0.3333333333333333

Process finished with exit code 0
```

## 二、實驗結果

### 1. 暴力法的 Apriori

使用暴力法的 Apriori 實驗可以發現其耗時較長，主要的原因在於生成子集的時候各個子集之間的連接，需要反復掃描整個 input 會產生大量的 overhead，且連接之後會產生大量的資料，根據提前給定的 support 和 confidence 進行



pruning 也需要對生成的結果進行掃描和操作，故 time consuming 和 overhead 都比較大。

使用 IBM.data (input 部分的第一種)作為 input，在 support count 為 4 且 minConf 為 0.25 的時候會產生很多規則，如 (\Apriori Brute\src\Apriori-output.txt) 所示。通過改變不同的 support count 和 minConf 可以發現，隨著 support count 的升高或 minConf 的升高，得到的 association rule 會變少，當兩者一同升高的時候到一定程度可能出現沒有 association rule 的情況。

因為 IBM.data 中均為數位，故所有的 input 在以字元流的形式讀取之後，我有用 Integer.valueOf()將所有的內容一 Integer 的形式進行存儲和處理。故對於 kaggle 中的資料進行處理之前必須將所有的內容處理成 Integer 的形式。

## 2. 使用 FP-growth 的 Apriori

使用 FP-growth 來優化演算法可以減少相應的執行時間。因為掃描整個資料集的時候就會得到各個頻繁子集，按照其出現頻率降冪排列，小於給定 support 的子集直接不予以考慮同時。第二次掃描會建立 frequent pattern tree，通過 Fp-growth 進行 mining 考慮 confidence 得到相應的 rule。

使用 Calendar 中的 getTimeInMillis() 可以得到整個實驗完成的時間，單位為 ms。輸出的時間表明使用 FP-growth 可以優化整個演算法，在相同的設定之下，其耗費的時間較

少。

使用 IBM.data (input 部分的第一種)作為 input，在 support count 為 4 且 minConf 為 0.25 的時候會產生很多規則，如 (\ Apriori fp-growth \src\fp-tree-output.txt) 所示。

以 kaggle 得到的資料作為 input，在 support count 為 100 而 minconfidence 等於 0.54 的時候我有得到結果 (\Apriori fp-growth\src\fp-tree-game-output.txt)。通過分析相應的 association rule 可知 1213、1415 和 1718 賽季 home 隊在 Time/Score keepers 的右側其獲勝的幾率較大。而 1314 和 1617 賽季 home 隊在左側其獲勝的幾率較大。1516 年 home 隊和 away 隊基本贏率對半分。

```
[away,1718,left]      277
[away,1617,right]     316
[1415,left] 610
[left,1718,home]      338
=====

*****association rule*****
Rules:{1314,left --> home} Confidence:0.5441696113074205
Rules:{1314,away --> right} Confidence:0.5436573311367381
Rules:{1617,left --> home} Confidence:0.5811023622047244
Rules:{right --> home} Confidence:0.5473818646232439
Rules:{home,1415 --> right} Confidence:0.5481171548117155
Rules:{1213,right --> home} Confidence:0.5823665893271461
Rules:{1718,left --> home} Confidence:0.5495934959349593
Rules:{1617 --> home} Confidence:0.5580865603644647
Rules:{1213,left --> home} Confidence:0.5783475783475783
Rules:{1718,right --> home} Confidence:0.5544267053701016
Rules:{1718 --> home} Confidence:0.5576070901033974
Rules:{1314 --> right} Confidence:0.5411942554799698
Rules:{left --> home} Confidence:0.5495601173020528
Rules:{1617,away --> right} Confidence:0.5429553264604811
Rules:{1213 --> home} Confidence:0.5806451612903226
Rules:{1314 --> home} Confidence:0.5411942554799698
Rules:{1415,right --> home} Confidence:0.5543018335684062
Rules:{1415 --> home} Confidence:0.5435936315390447
*****
```

### 三、總結

FP-growth 的奧妙在於使用簡單的 tree 結構來表示原有資料的資訊，其 size 遠小於原來的結構，通過對資料結構進行遞迴可以完整 frequent pattern 的 mining，從而得到 association rule。另外由於 tree 結構的 size 有變小，故可以放入記憶體中極大地加快了運算。如果 tree 過大，則可以通過切分原始資料分塊進行 FP-growth 在進行整合。

### 四、寫作業中遇到的困難

因為兩種演算法資料結構和思路不一樣，所以其相應的計算頻繁子集的方法也不一樣。在撰寫 FP-growth 時關於分析關聯性的 getRelationRules 函數的時候因為考慮不周到，以至於獲得相應子集之後計算其 confidence 時，並沒有分情況考慮單子集和多子集的 confidence，故生成規則的時候有出現 null pointer，耗費了一段時間 debug。另外在最後結果輸出的時候因為其存儲為 map<map, string>的形式，故要特別改變格式進行輸出。

這次作業的完成不僅讓我更好的理解了 Apriori 演算法和 FP-growth 演算法，更是更進一步的熟悉了對於 linked list 和 map 的使用，對於 coding 的能力大有幫助。同時通過找尋相應的 kaggle input 讓我對於 association rule 的使用情況有了進一步的掌握。