資料探勘 HW1 報告

報告說明：

藉由 IBM Quest Synthetic Data Generator 產生一些 dataset 以分析 Support 與 Confidence 的高低如何影響從 dataset 選出的 data 關聯度以及從 Kaggle 下載 dataset 進行測試與分析。

報告架構：

A. Apriori algorithm & FP-growth algorithm 程式架構簡介
B. 自己生產的 IBM Data Generator Dataset 使用參數說明
C. 使用 Apriori algorithm & FP-growth algorithm 測試 IBM Dataset 結果
D. 觀察 High support, high confidence High support, low confidence Low support, low confidence Low support, high confidence 這四種情境下得到的結果與分析
E. Kaggle dataset 測試與分析
F. 撰寫作業遇到的困難與解決方式
G. 結論

Apriori algorithm & FP-growth algorithm 程式架構與簡介

將 apriori algorithm & FP-growth algorithm 實作方式放在 my_cool_algorithm.py

－my_cool_algorithm.py
　－def apriori(dataset, a)
　－def FP_growth(dataset, a)
　－rule_generation(min_conf, frequent_itemsets, total_transaction_conts)

✧ def apriori(dataset, a) 程式說明

1. 讀取完 dataset 後計算出各 transaction 對應到的 itemsets 以及計算出 transaction 總數量。
2. 計算出 C1 與 L1 以找出 frequent item
3. 利用 L1 計算出 C2，並刪除不滿足 min_sup 的 candidate 以篩選出 L2。重複這個 step 直到 Lk 只剩下小於一個 candidate 為止。
4. 將 step3 計算出的 frequent itemsets 套入 rule generation function，計算出各 frequent itemset 的 support, confidence, lift，並將未達到 min_sup & min_conf 的 frequent item 移除。

✧ def FP_growth(dataset, a) 程式說明

1. 讀取完 dataset 後計算出各 transaction 對應到的 itemsets 以及計算出 transaction 總數量。
2. 計算出 C1 與 L1 以找出 frequent item
3. 將 step2 找出的 frequent item 從出現頻率高排到出現頻率低，並根據排序過的 frequent item 重新建立各 transaction 的 itemsets，並由出現頻率高排到頻率低。
4. 將 step3 取得排序過的 transaction 建立成 tree，其中 tree 的 node 會記錄 frequent item 的 count, child node list, name。若 node 在 path 中未出現則建立 node，並將此 node 放入父點的 child node list。若 node 在 path 中出現過則將 count + 1。
5. 依照 node 出現的頻率大小建立 header table，紀錄各 frequent item node 的位置方便之後讀取。
6. 從出現頻率最低的 frequent item 開始 trace 有出現此 frequent item 的 path，直到所有 frequent item 的 path 建立完成。
7. 循著各 frequent item paths，刪除不合 min_sup 的 node，並重新建立 frequent item paths 直到所有 frequent item 的 paths 重新建立完成，並找出 frequent itemsets。
8. 將 step6 計算出的 frequent itemsets 套入 rule generation function，計算出各 frequent itemset 的 support, confidence, lift，並將未達到 min_sup & min_conf 的 frequent item 移除。

✧ def rule_generation(min_conf, frequent_itemsets, total_transaction_conts)

1. 將 frequent itemsets 中所有 frequent itemset 做組合，找出所有可能性。
   Example：(0,1,2)：{0}->{1,2}, {1}->{0,2}, {2}->{0,1},{0,1}->{2}, {1,2}->{0}, {0,2}->{1}
2. 計算出 support & confidence，若未達到 min_sup & min_conf 則刪除。
3. 將所有符合 min_sup & min_conf 的 frequent itemsets 計算出各自的 lift

自己生產的 IBM Data Generator Dataset 使用參數說明

Dataset：IBM Quest Synthetic Data Generator

參數設定：共兩組 datasets

Dataset 1

- ✓ ntrans：20
- ✓ tlen：8
- ✓ nitems：0.1
- ✓ npats：50
- ✓ corr：0.4
- ✓ conf：0.8

Dataset 2

- ✓ ntrans：0.3
- ✓ tlen：8
- ✓ nitems：0.03
- ✓ npats：20
- ✓ corr：0.4
- ✓ conf：0.8

使用 Apriori algorithm & FP-growth algorithm 測試 IBM Dataset 結果

Dataset: -ntrans 20 -tlen 8 -nitems 0.1 -npats 50 -corr 0.4 -conf 0.8 (Dataset 1)

參數設定：min_sup = 0.2, min_conf = 0.98

Apriori algorithm

	antecedent	consequent	support	confidence	lift
2	{7 69 39}	{86}	0.203	0.983	1.792
3	{69 86 7}	{39}	0.203	0.983	2.367
4	{51 69 39}	{86}	0.204	0.98	1.787
5	{69 93 39}	{86}	0.203	0.982	1.79
6	{51 69 7}	{39}	0.202	0.983	2.366
7	{69 93 7}	{39}	0.2	0.982	2.366
8	{51 69 7}	{86}	0.202	0.983	1.791
9	{69 93 7}	{86}	0.201	0.984	1.793
10	{51 7 39}	{86}	0.203	0.982	1.79
11	{51 86 7}	{39}	0.203	0.982	2.365
12	{7 93 39}	{86}	0.202	0.983	1.792
13	{93 86 7}	{39}	0.202	0.982	2.365
14	{51 93 39}	{86}	0.202	0.981	1.788
15	{51 93 7}	{39}	0.2	0.982	2.365
16	{51 93 7}	{86}	0.2	0.982	1.791
17	{69 7}	{39}	0.207	0.982	2.365
18	{69 7}	{86}	0.207	0.982	1.791
19	{7 39}	{86}	0.208	0.982	1.79
20	{86 7}	{39}	0.208	0.981	2.363
21	{51 7}	{39}	0.207	0.981	2.363
22	{93 7}	{39}	0.205	0.981	2.363
23	{51 7}	{86}	0.207	0.981	1.789
24	{93 7}	{86}	0.205	0.982	1.791
25	{7}	{39}	0.212	0.981	2.362
26	{7}	{86}	0.212	0.981	1.788

FP-growth algorithm

	antecedent	consequent	support	confidence	lift
2	{51 93 7}	{39}	0.2	0.982	2.365
3	{51 93 7}	{86}	0.2	0.982	1.791
4	{69 93 7}	{39}	0.2	0.982	2.366
5	{69 93 7}	{86}	0.201	0.984	1.793
6	{7 93 39}	{86}	0.202	0.983	1.792
7	{93 86 7}	{39}	0.202	0.982	2.365
8	{51 69 7}	{39}	0.202	0.983	2.366
9	{51 69 7}	{86}	0.202	0.983	1.791
10	{51 7 39}	{86}	0.203	0.982	1.79
11	{51 86 7}	{39}	0.203	0.982	2.365
12	{7 69 39}	{86}	0.203	0.983	1.792
13	{69 86 7}	{39}	0.203	0.983	2.367
14	{51 93 39}	{86}	0.202	0.981	1.788
15	{69 93 39}	{86}	0.203	0.982	1.79
16	{51 69 39}	{86}	0.204	0.98	1.787
17	{93 7}	{39}	0.205	0.981	2.363
18	{93 7}	{86}	0.205	0.982	1.791
19	{51 7}	{39}	0.207	0.981	2.363
20	{51 7}	{86}	0.207	0.981	1.789
21	{69 7}	{39}	0.207	0.982	2.365
22	{69 7}	{86}	0.207	0.982	1.791
23	{7 39}	{86}	0.208	0.982	1.79
24	{86 7}	{39}	0.208	0.981	2.363
25	{7}	{39}	0.212	0.981	2.362
26	{7}	{86}	0.212	0.981	1.788

Apriori & FP-growth 都找到 25 條 rules

以 Weka 做驗證：共找到 25 條 rules

```
Best rules found:

 1. 7=1 69=1 93=1 3880 ==> 86=1 3816    <conf:(0.98)> lift:(1.79) lev:(0.09) [1687] conv:(26.95)
 2. 7=1 39=1 93=1 3906 ==> 86=1 3839    <conf:(0.98)> lift:(1.79) lev:(0.09) [1696] conv:(25.93)
 3. 7=1 69=1 86=1 3933 ==> 39=1 3865    <conf:(0.98)> lift:(2.37) lev:(0.12) [2231] conv:(33.33)
 4. 7=1 39=1 69=1 3933 ==> 86=1 3865    <conf:(0.98)> lift:(1.79) lev:(0.09) [1707] conv:(25.73)
 5. 7=1 51=1 69=1 3904 ==> 39=1 3836    <conf:(0.98)> lift:(2.37) lev:(0.12) [2214] conv:(33.08)
 6. 7=1 51=1 69=1 3904 ==> 86=1 3836    <conf:(0.98)> lift:(1.79) lev:(0.09) [1694] conv:(25.54)
 7. 7=1 69=1 93=1 3880 ==> 39=1 3812    <conf:(0.98)> lift:(2.37) lev:(0.12) [2200] conv:(32.88)
 8. 7=1 69=1 4004 ==> 39=1 3933    <conf:(0.98)> lift:(2.37) lev:(0.12) [2270] conv:(32.52)
 9. 7=1 69=1 4004 ==> 86=1 3933    <conf:(0.98)> lift:(1.79) lev:(0.09) [1736] conv:(25.11)
10. 7=1 51=1 93=1 3881 ==> 86=1 3812    <conf:(0.98)> lift:(1.79) lev:(0.09) [1683] conv:(25.03)
11. 7=1 93=1 3980 ==> 86=1 3909    <conf:(0.98)> lift:(1.79) lev:(0.09) [1725] conv:(24.96)
12. 7=1 86=1 93=1 3909 ==> 39=1 3839    <conf:(0.98)> lift:(2.37) lev:(0.12) [2215] conv:(32.19)
13. 7=1 51=1 93=1 3881 ==> 39=1 3811    <conf:(0.98)> lift:(2.36) lev:(0.12) [2199] conv:(31.96)
14. 7=1 51=1 86=1 3933 ==> 39=1 3862    <conf:(0.98)> lift:(2.36) lev:(0.12) [2228] conv:(31.94)
15. 7=1 39=1 51=1 3933 ==> 86=1 3862    <conf:(0.98)> lift:(1.79) lev:(0.09) [1704] conv:(24.66)
16. 39=1 69=1 93=1 3930 ==> 86=1 3858    <conf:(0.98)> lift:(1.79) lev:(0.09) [1702] conv:(24.3)
17. 7=1 39=1 4035 ==> 86=1 3961    <conf:(0.98)> lift:(1.79) lev:(0.09) [1747] conv:(24.29)
18. 7=1 86=1 4036 ==> 39=1 3961    <conf:(0.98)> lift:(2.36) lev:(0.12) [2285] conv:(31.05)
19. 7=1 93=1 3980 ==> 39=1 3906    <conf:(0.98)> lift:(2.36) lev:(0.12) [2253] conv:(31.03)
20. 7=1 51=1 4008 ==> 39=1 3933    <conf:(0.98)> lift:(2.36) lev:(0.12) [2268] conv:(30.84)
21. 7=1 51=1 4008 ==> 86=1 3933    <conf:(0.98)> lift:(1.79) lev:(0.09) [1734] conv:(23.81)
22. 7=1 4114 ==> 86=1 4036    <conf:(0.98)> lift:(1.79) lev:(0.09) [1779] conv:(23.51)
23. 39=1 51=1 93=1 3927 ==> 86=1 3852    <conf:(0.98)> lift:(1.79) lev:(0.09) [1697] conv:(23.33)
24. 7=1 4114 ==> 39=1 4035    <conf:(0.98)> lift:(2.36) lev:(0.12) [2326] conv:(30.07)
25. 39=1 51=1 69=1 3961 ==> 86=1 3883    <conf:(0.98)> lift:(1.79) lev:(0.09) [1710] conv:(22.64)
```

Dataset: -ntrans 0.3 -tlen 8 -items 0.03 -npats 20 -corr 0.4 -conf 0.8 (Dataset 2)

參數設定： min_sup = 0.6 min_conf = 0.98

Apriori algorithm

1	antecedent	consequent	support	confidence	lift
2	{3 11 28}	{29}	0.604	1	1.112
3	{3 28 29}	{11}	0.604	0.989	1.167
4	{3 28}	{11 29}	0.604	0.983	1.231
5	{28 29 14}	{11}	0.608	0.989	1.167
6	{11 28 21}	{29}	0.615	0.994	1.106
7	{3 28}	{11}	0.604	0.983	1.16
8	{3 11}	{29}	0.622	0.989	1.1
9	{3 29}	{11}	0.622	0.984	1.161
10	{3 21}	{29}	0.601	0.994	1.106
11	{3 28}	{29}	0.611	0.994	1.106
12	{21 14}	{11}	0.604	0.989	1.167
13	{28 14}	{11}	0.622	0.989	1.167
14	{29 14}	{11}	0.646	0.984	1.162
15	{11 21}	{29}	0.628	0.989	1.1
16	{21 14}	{29}	0.604	0.989	1.099
17	{28 21}	{29}	0.628	0.995	1.106
18	{3}	{29}	0.632	0.984	1.094
19	{21}	{29}	0.653	0.984	1.095

FP-growth algorithm

1	antecedent	consequent	support	confidence	lift
2	{3 11 28}	{29}	0.604	1	1.112
3	{3 28 29}	{11}	0.604	0.989	1.167
4	{3 28}	{11 29}	0.604	0.983	1.231
5	{11 28 21}	{29}	0.615	0.994	1.106
6	{28 29 14}	{11}	0.608	0.989	1.167
7	{3 21}	{29}	0.601	0.994	1.106
8	{3 28}	{11}	0.604	0.983	1.16
9	{3 28}	{29}	0.611	0.994	1.106
10	{3 11}	{29}	0.622	0.989	1.1
11	{3 29}	{11}	0.622	0.984	1.161
12	{21 14}	{11}	0.604	0.989	1.167
13	{21 14}	{29}	0.604	0.989	1.099
14	{28 21}	{29}	0.628	0.995	1.106
15	{11 21}	{29}	0.628	0.989	1.1
16	{28 14}	{11}	0.622	0.989	1.167
17	{29 14}	{11}	0.646	0.984	1.162
18	{3}	{29}	0.632	0.984	1.094
19	{21}	{29}	0.653	0.984	1.095

Apriori & FP-growth 都找到 18 條 rules

以 Weka 做驗證：共找到 18 條 rules

```
Best rules found:

 1. 3=1 11=1 28=1 174 ==> 29=1 174    <conf:(1)> lift:(1.11) lev:(0.06) [17] conv:(17.52)
 2. 21=1 28=1 182 ==> 29=1 181    <conf:(0.99)> lift:(1.11) lev:(0.06) [17] conv:(9.16)
 3. 11=1 21=1 28=1 178 ==> 29=1 177    <conf:(0.99)> lift:(1.11) lev:(0.06) [16] conv:(8.96)
 4. 3=1 28=1 177 ==> 29=1 176    <conf:(0.99)> lift:(1.11) lev:(0.06) [16] conv:(8.91)
 5. 3=1 21=1 174 ==> 29=1 173    <conf:(0.99)> lift:(1.11) lev:(0.06) [16] conv:(8.76)
 6. 11=1 21=1 183 ==> 29=1 181    <conf:(0.99)> lift:(1.1) lev:(0.06) [16] conv:(6.14)
 7. 3=1 11=1 181 ==> 29=1 179    <conf:(0.99)> lift:(1.1) lev:(0.06) [16] conv:(6.08)
 8. 14=1 28=1 181 ==> 11=1 179    <conf:(0.99)> lift:(1.17) lev:(0.09) [25] conv:(9.22)
 9. 14=1 28=1 29=1 177 ==> 11=1 175    <conf:(0.99)> lift:(1.17) lev:(0.09) [25] conv:(9.01)
10. 14=1 21=1 176 ==> 11=1 174    <conf:(0.99)> lift:(1.17) lev:(0.09) [24] conv:(8.96)
11. 14=1 21=1 176 ==> 29=1 174    <conf:(0.99)> lift:(1.1) lev:(0.05) [15] conv:(5.91)
12. 3=1 28=1 29=1 176 ==> 11=1 174    <conf:(0.99)> lift:(1.17) lev:(0.09) [24] conv:(8.96)
13. 21=1 191 ==> 29=1 188    <conf:(0.98)> lift:(1.09) lev:(0.06) [16] conv:(4.81)
14. 14=1 29=1 189 ==> 11=1 186    <conf:(0.98)> lift:(1.16) lev:(0.09) [25] conv:(7.22)
15. 3=1 185 ==> 29=1 182    <conf:(0.98)> lift:(1.09) lev:(0.05) [15] conv:(4.66)
16. 3=1 29=1 182 ==> 11=1 179    <conf:(0.98)> lift:(1.16) lev:(0.09) [24] conv:(6.95)
17. 3=1 28=1 177 ==> 11=1 174    <conf:(0.98)> lift:(1.16) lev:(0.08) [24] conv:(6.76)
18. 3=1 28=1 177 ==> 11=1 29=1 174    <conf:(0.98)> lift:(1.23) lev:(0.11) [32] conv:(8.91)
```

Time & Memory Records – Apriori algorithm

	Dataset 1	Dataset 2
Time	0.560063 sec	0.004000 sec
Memory	13.524514 MB	0.228676 MB

Time & Memory Records – FP-growth algorithm

	Dataset 1	Dataset 2
Time	0.148997 sec	0.002001 sec
Memory	16.99488 MB	0.304271 MB

Time & Memory Analysis

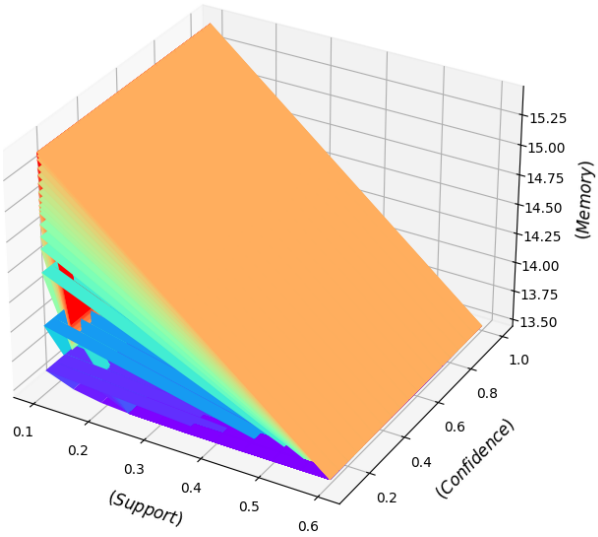
從上面兩個 dataset 所耗費的時間與 Memory 來看，可以發現 Apriori algorithm 所耗費的時間幾乎都比 FP-growth algorithm 還多，但花費的記憶體空間則較 FP-growth 少，由此可見 FP-grow 藉由建立 tree 可以先過濾掉不合 min_sup 的一些 candidate，以降低執行時間，但代價則是當 item 種類較多時，或是共同的 item 數量較少時，path 數量會增加，建立 tree 的所需的記憶體空間不小。

反之，Apriori algorithm 則是會計算出所有可能的 candidate，最後放入 generation rule 時再刪除不合 min_sup & min_conf 的 candidate，因此所需的記憶體空間會較 FP-growth 小(不需要建 tree)，但耗費時間較多。

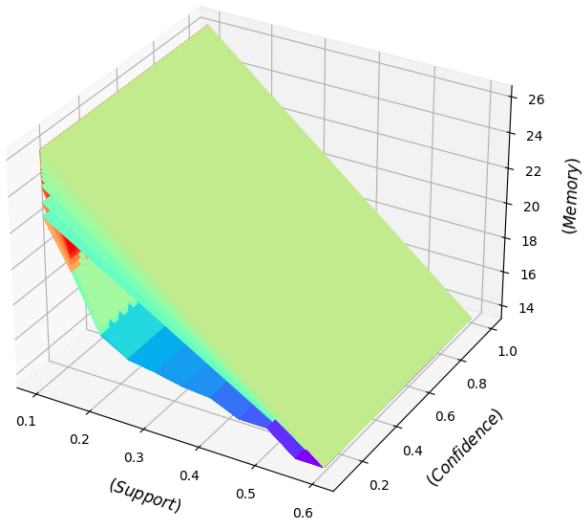
Memory Analysis about different min_sup & min_conf

Dataset 1

Apriori algorithm

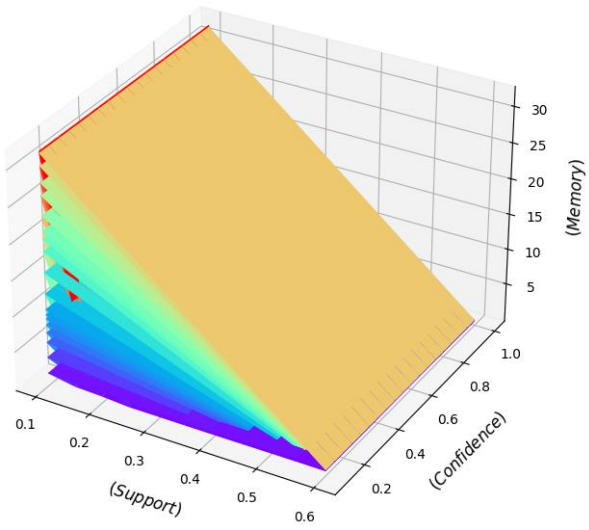


FP-growth algorithm

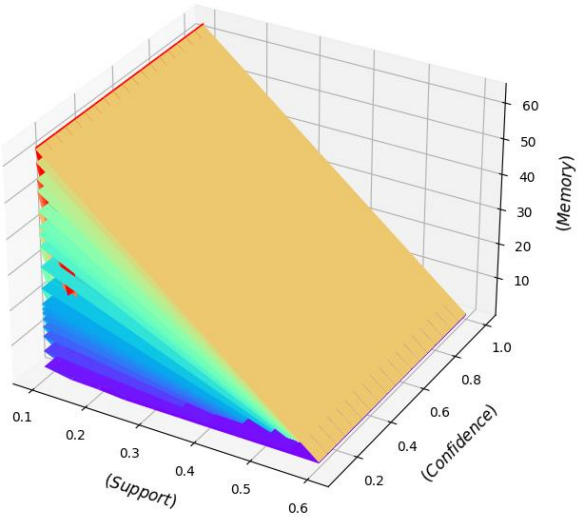


Dataset 2

Apriori algorithm



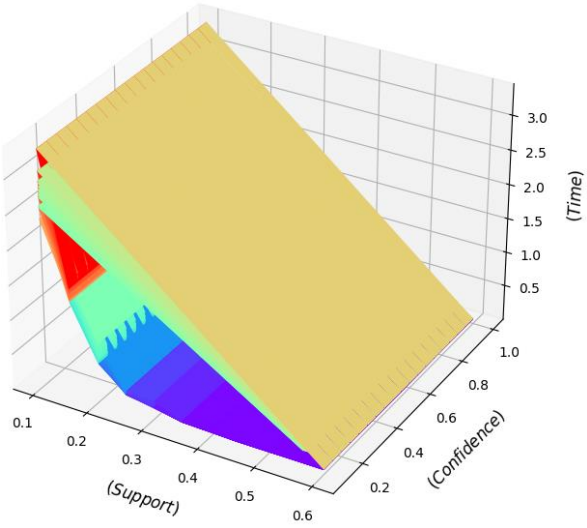
FP-growth algorithm



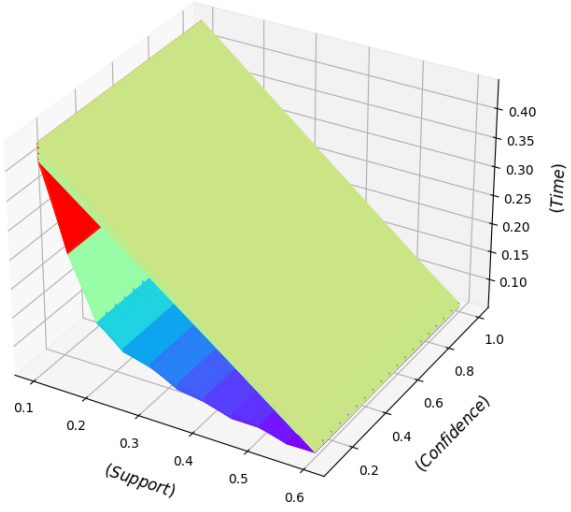
Time Analysis about different min_sup & min_conf

Dataset 1

Apriori algorithm

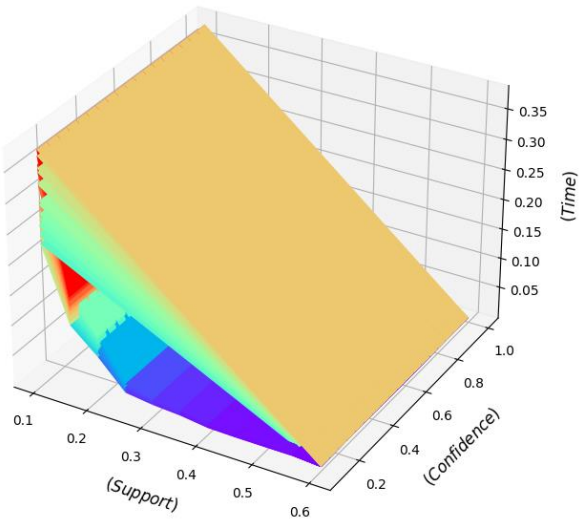


FP-growth algorithm

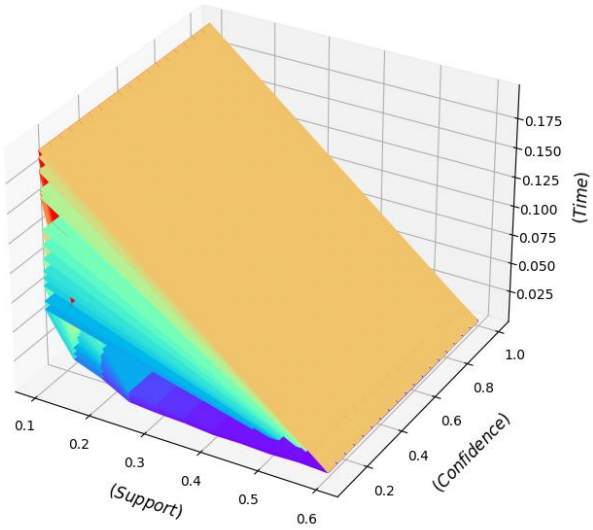


Dataset2

Apriori algorithm



FP-growth algorithm



說明與推測：

Memory:

Dataset 1 中的 apriori algorithm 在 min_sup 較小時，所需的記憶體空間較 FP-growth 小一些，但當 min_sup 越大時，可能因為在生產 Lk 的過程中，符合 min_sup 的 candidate 數量相較於 min_sup = 0.1 時還要少，因此最後再產生 rule 時使用到的 combination 次數降低，造成所需記憶體空間降低

而 FP-growth algorithm 也是相同的道理，min_sup 越大在建樹前不合 min_sup 的 item 就會先被刪除，導致建 tree 所需記錄的 node 數降低，所需尋找的 pattern 也會相對降低，故 min_sup 越大則所需的記憶體空間會降低。

而 min_conf 從圖中可得知無論 min_conf 大小都不會影響到所需的記憶體空間，影響記憶體空間主要的因素是 min_sup。可能是因為 min_conf 是在產生 rule 時才需考慮的因素，但這兩個 algorithm 最主要耗費的時間是在尋找 frequent itemsets 上，因此 min_conf 不會造成顯著影響。

在 Dataset 1 & Dataset2，由於 Dataset 1 transaction 總數較多且 item 種類較多，因此 min_sup 越小時 Dataset 2 反而會需要較多的記憶體空間去做計算 frequent itemsets。推測是因為 Dataset 1 transaction 總數較多，且 item 種類也較多，在要達成 min_sup 會是相對較高的條件較為困難，因為每種出現的 item 頻率會降低，且 min_sup 分母為 transaction 總數，總數較大整體機率也會被稀釋，要符合 min_sup 的 frequent item 數量會較少，故所需記憶體容量也會較少，反之 Dataset 2 所需的記憶體容量則會較大，因為 transaction 總數較少且 item 種類也較少，出現頻率會較高，導致 candidate 產生的數量會增加許多，所需記錄的空間也會增加。

Time:

從圖中可得知，Dataset 1 中 apriori algorithm 計算所需時間較 FP-growth 多，因為 apriori algorithm 會將所有 candidate 找出來，再將不合 min_sup 的從中剔除。但 FP-growth 則在建樹時就先將不合 min_sup 的 frequent item 刪除，去除不必要的計算，因此 FP-growth 和課本理論一樣，利用空間換取時間的方式使整體程式執行時間加快許多，且不需尋找全部的 candidate。

在 Dataset 1 & Dataset2，同樣是 apriori algorithm，Dataset 2 計算完所需的時間較低，推測是因為 Dataset 2 的 transaction 總數較低，因此在需要計算出所有 candidate 的 apriori algorithm 情況下，整體計算時間會較少。

FP-growth 則可能是因為 pattern 設定上較 Dataset 1 小，且 transaction 總數較少，使得 FP-growth 整體運算速度也較 Dataset 1 快一些。

從圖中也得知，影響計算時間的長短主要是受 min_sup 影響，而非 min_conf，原因如同上段分析記憶體的說明。

觀察 High support, high confidence High support, low confidence Low support, low confidence Low support, high confidence 這四種情境下得到的結果與分析

Rules	Observed
High support, high confidence	強關聯，會是理想的結果。 但從許多的 datasets 觀察下來符合這樣的結果非常少。除非在 transaction 總數非常少且 item 種類也很少時可能會出現。可能較適合小零售商單日客人購買商品進行分析。
High support, low confidence	Item 出現頻率高，但可能 subpattern 很多導致最終 confidence 較低，item 之間關連性相對弱，實用性相對低。
Low support, low confidence	產生的 rule 會非常多，但要特別篩選才能找出較有關連的資料，實用性相對較低，甚至幾乎沒有。
Low support, high confidence	觀察許多 dataset 中最常出現的結果，transaction 總數多，或是 item 種類多時容易出現，因為會將 item 出現頻率降低。不過可以藉由 confidence 去篩選關聯較強的 item，在現實中實用性相對高，因為大部分資料 transaction 總數都較多。配合 lift 分析觀察會找出關聯性更高的結果。

- 除了 confidence 外，lift 也是很重要的關聯度強弱指標。若 lift 越大則表示 Item 之前的關聯越大，越有可能同時出現在一筆 transaction，反之則無。

- 個人認為 low support, high confidence 較常被使用，由於只要賣的商品種類或 transaction 數量夠多，要達到 high support 是非常困難的。即便是 low support，只要配合 confidence & lift 也可以找出 item 之間的關聯強弱。

Kaggle dataset 測試與分析

Dataset： Movie rating

Descripting：

Comprises of 100,000 ratings and 1,300 tag applications applied to 9,000 movies by 700 users.

Preprocess：

為了較為準確地可以推薦電影給使用者，故事先將 dataset 中每位使用者電影評分<4 分的電影 ID 都先刪除，留下使用者較喜歡的電影。

參數設定： min_sup = 0.11, min_conf = 0.9

Apriori algorithm

	antecedent	consequent	support	confidence	lift
2	{1210 1196 1198}	{260}	0.133	0.947	2.738
3	{1210 260 1198}	{1196}	0.133	0.957	3.398
4	{1210 2571 1196}	{260}	0.121	0.931	2.693
5	{1210 2571 260}	{1196}	0.121	0.988	3.507
6	{5952 4993 1196}	{260}	0.11	0.937	2.709
7	{5952 1196 260}	{4993}	0.11	0.974	4.109
8	{5952 1196}	{4993 260}	0.11	0.902	5.879
9	{5952 7153 260}	{4993}	0.112	0.974	4.111
10	{4993 260 7153}	{5952}	0.112	0.938	4.43
11	{7153 260}	{5952 4993}	0.112	0.915	4.871
12	{5952 7153 2571}	{4993}	0.118	0.929	3.922
13	{4993 2571 7153}	{5952}	0.118	0.952	4.498
14	{296 1210}	{1196}	0.113	0.916	3.251
15	{1196 589}	{260}	0.116	0.907	2.623
16	{1210 1196}	{260}	0.18	0.91	2.631
17	{1210 1198}	{260}	0.139	0.912	2.637
18	{4993 1210}	{260}	0.113	0.938	2.714
19	{1210 1198}	{1196}	0.14	0.922	3.272
20	{1210 1270}	{1196}	0.119	0.92	3.265
21	{1210 2571}	{1196}	0.13	0.926	3.286
22	{4993 1210}	{1196}	0.11	0.914	3.243
23	{5952 1196}	{260}	0.113	0.927	2.681
24	{2571 1196}	{260}	0.145	0.915	2.647
25	{4993 1196}	{260}	0.13	0.916	2.649
26	{7153 260}	{5952}	0.115	0.939	4.437
27	{5952 260}	{4993}	0.127	0.966	4.076
28	{7153 260}	{4993}	0.119	0.976	4.117
29	{1291 1196}	{1198}	0.116	0.929	3.5
30	{5952 1196}	{4993}	0.118	0.963	4.066
31	{7153 1196}	{4993}	0.113	0.987	4.165
32	{7153 2571}	{5952}	0.127	0.934	4.414
33	{5952 7153}	{4993}	0.165	0.917	3.871
34	{4993 7153}	{5952}	0.165	0.933	4.408
35	{5952 2571}	{4993}	0.134	0.909	3.836
36	{7153 2571}	{4993}	0.124	0.912	3.849
37	{1221}	{858}	0.156	0.921	3.472
38	{7153}	{5952}	0.18	0.917	4.332
39	{7153}	{4993}	0.177	0.902	3.805

FP-growth algorithm

	antecedent	consequent	support	confidence	lift
2	{5952 7153 2571}	{4993}	0.118	0.929	3.922
3	{4993 2571 7153}	{5952}	0.118	0.952	4.498
4	{5952 7153 260}	{4993}	0.112	0.974	4.111
5	{4993 260 7153}	{5952}	0.112	0.938	4.43
6	{7153 260}	{5952 4993}	0.112	0.915	4.871
7	{5952 4993 1196}	{260}	0.11	0.937	2.709
8	{5952 1196 260}	{4993}	0.11	0.974	4.109
9	{5952 1196}	{4993 260}	0.11	0.902	5.879
10	{1210 1196 1198}	{260}	0.133	0.947	2.738
11	{1210 260 1198}	{1196}	0.133	0.957	3.398
12	{1210 2571 1196}	{260}	0.121	0.931	2.693
13	{1210 2571 260}	{1196}	0.121	0.988	3.507
14	{1291 1196}	{1198}	0.116	0.929	3.5
15	{7153 2571}	{5952}	0.127	0.934	4.414
16	{5952 7153}	{4993}	0.165	0.917	3.871
17	{4993 7153}	{5952}	0.165	0.933	4.408
18	{7153 2571}	{4993}	0.124	0.912	3.849
19	{7153 260}	{5952}	0.115	0.939	4.437
20	{7153 1196}	{4993}	0.113	0.987	4.165
21	{7153 260}	{4993}	0.119	0.976	4.117
22	{5952 1196}	{4993}	0.118	0.963	4.066
23	{5952 2571}	{4993}	0.134	0.909	3.836
24	{5952 260}	{4993}	0.127	0.966	4.076
25	{5952 1196}	{260}	0.113	0.927	2.681
26	{1210 1270}	{1196}	0.119	0.92	3.265
27	{4993 1210}	{1196}	0.11	0.914	3.243
28	{4993 1210}	{260}	0.113	0.938	2.714
29	{4993 1196}	{260}	0.13	0.916	2.649
30	{1210 1198}	{1196}	0.14	0.922	3.272
31	{1210 1198}	{260}	0.139	0.912	2.637
32	{1210 1196}	{260}	0.18	0.91	2.631
33	{296 1210}	{1196}	0.113	0.916	3.251
34	{1210 2571}	{1196}	0.13	0.926	3.286
35	{1196 589}	{260}	0.116	0.907	2.623
36	{2571 1196}	{260}	0.145	0.915	2.647
37	{1221}	{858}	0.156	0.921	3.472
38	{7153}	{5952}	0.18	0.917	4.332
39	{7153}	{4993}	0.177	0.902	3.805

Apriori & FP-growth 都找到 38 條 rules

- 從上面 Apriori & FP-growth 分析結果可以看出，雖然 min_sup 被設為 0.1 以上才可以有幾筆結果，但由於 confidence 有達到一定水準，且 lift 也有>1，故找出來的 rule 還是有一定的可信度。推測可能是因為電影種類太多，造成 frequent item 出現的次數較低，才使整體 support 較低。但由於 confidence & lift 有到達一定水準，因此還是可以作為電影推薦系統的一種方式，但這個 dataset 只有根據 rating 去判斷，並沒有考量到性別、年齡等等因素，若再加入這些因素去分析或許可以推薦更符合使用者喜好的電影。

撰寫作業遇到的困難與解決方式

- 在寫這次作業的過程中，比較主要遇到的問題是在實作 FP-growth 時，pattern 較少的 dataset 都可以很快速、較 apriori algorithm 快的速度運行出結果。
  但在助教公布最新測資時發現 apriori algorithm 運行速度較 FP-growth 快上許多(0.9 sec)，FP-growth 則是完全無法跑出結果。
  最後發現是在找到 pattern 後沒有先把不合 min_sup 的 node 移除掉，在後續 combination 時會有過於大量的可能性，因此才無法跑出結果。才發現是先將不合 min_sup 的 node 先移除掉是有多重要的步驟。較小的 dataset 可能沒有問題，但遇到 pattern 較少，item 種類較少的 dataset，很容易造成樹的高度過高，combination 的次數會大幅增加，造成需要花費非常大量的時間與記憶體。
  除了這樣的狀況，其餘 dataset 幾乎都是 FP-growth 運行速度較 apriori algorithm 快上許多。

結論

- 經由這次的作業練習實作，學習到如何將 apriori algorithm & FP-growth algorithm 用程式碼實現，也理解 support & confidence 設定 high & low 各自帶來結果的參考價值，試用在 kaggle 上的 dataset 也有不錯的結果。也實際觀察 FP-growth & apriori algorithm 在耗費的時間與記憶體容量的差異，收穫很多。