DATA MINING

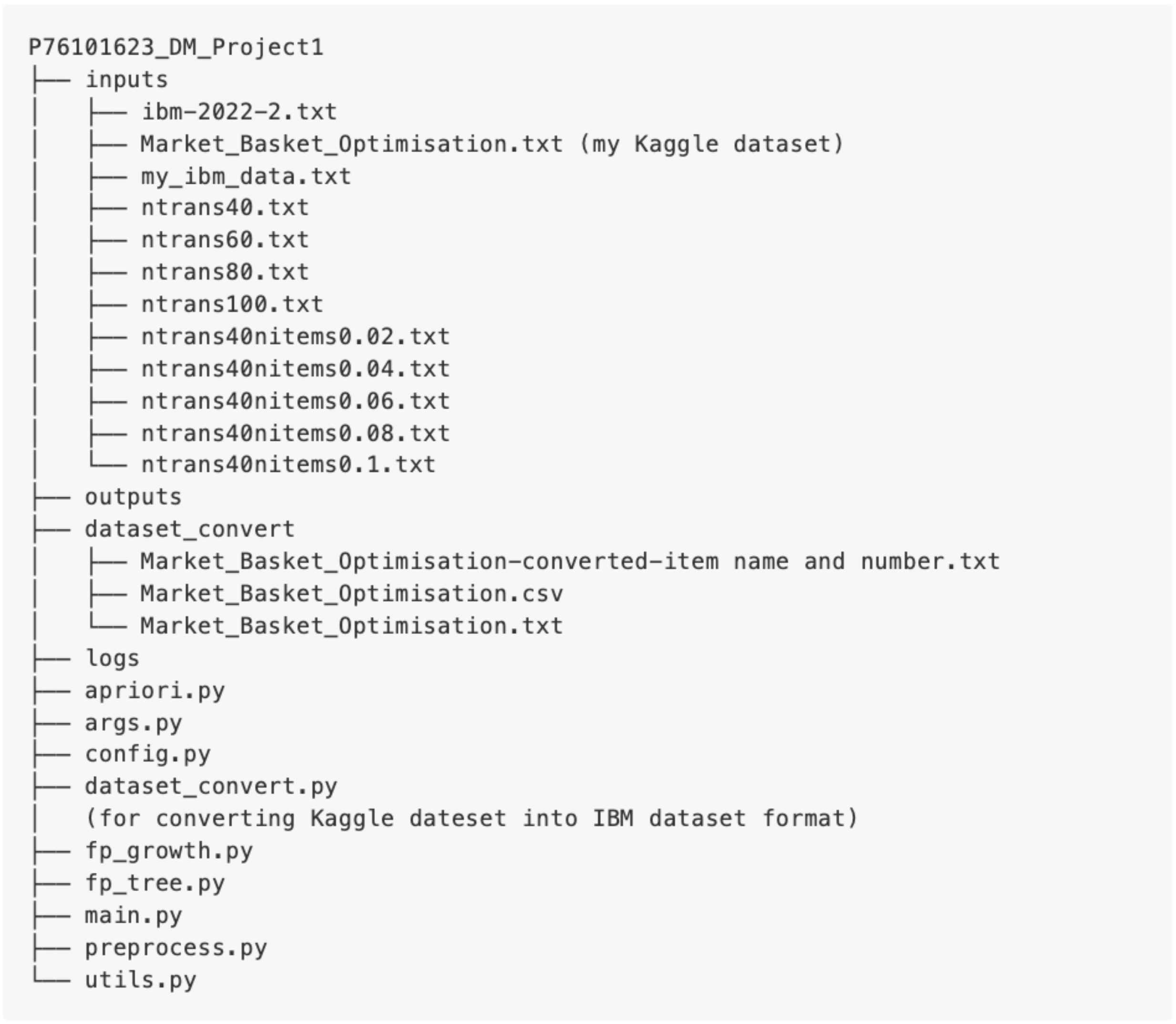
[CSIE 7111] FALL 2022

PROJECT 1

Student Name: 曾中柏
Student ID: P76101623

Table of Contents

PART 1 FILE STRUCTURE

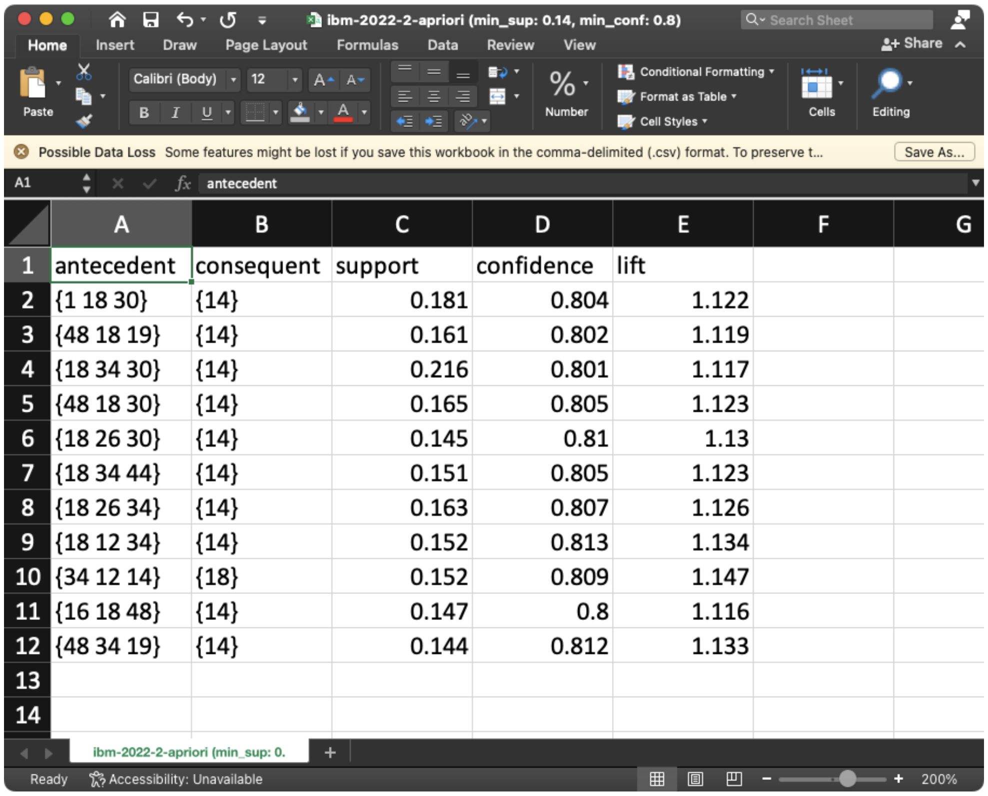


```
P76101623_DM_Project1
├── inputs
│   ├── ibm-2022-2.txt
│   ├── Market_Basket_Optimisation.txt (my Kaggle dataset)
│   ├── my_ibm_data.txt
│   ├── ntrans40.txt
│   ├── ntrans60.txt
│   ├── ntrans80.txt
│   ├── ntrans100.txt
│   ├── ntrans40nitems0.02.txt
│   ├── ntrans40nitems0.04.txt
│   ├── ntrans40nitems0.06.txt
│   ├── ntrans40nitems0.08.txt
│   └── ntrans40nitems0.1.txt
├── outputs
├── dataset_convert
│   ├── Market_Basket_Optimisation-converted-item name and number.txt
│   ├── Market_Basket_Optimisation.csv
│   └── Market_Basket_Optimisation.txt
├── logs
├── apriori.py
├── args.py
├── config.py
├── dataset_convert.py
│   (for converting Kaggle dateset into IBM dataset format)
├── fp_growth.py
├── fp_tree.py
├── main.py
├── preprocess.py
└── utils.py
```

Figure 1. File structure of this project

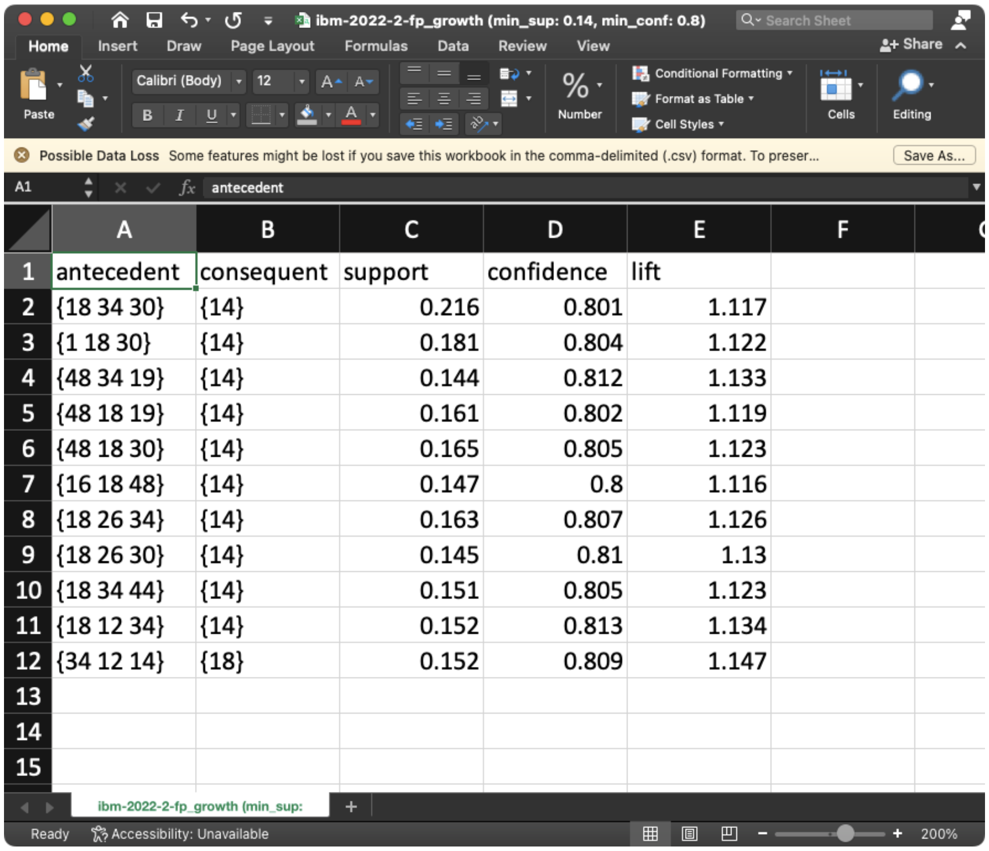
PART 2 2022-DM-RELEASE-TESTDATA-2

- I randomly picked the following threshold to try to generate 11 rules from this dataset:
    - min_sup = 0.14
    - min_conf = 0.8
- Both Apriori and FP-growth algorithms generated 11 rules (Figure 2 and Figure 3)
- Results were verified by Weka (Figure 4)


ibm-2022-2-apriori (min_sup: 0.14, min_conf: 0.8)
Possible Data Loss Some features might be lost if you save this workbook in the comma-delimited (.csv) format. To preserve t...
Save As...
ibm-2022-2-apriori (min_sup: 0.
Ready
Accessibility: Unavailable
200%

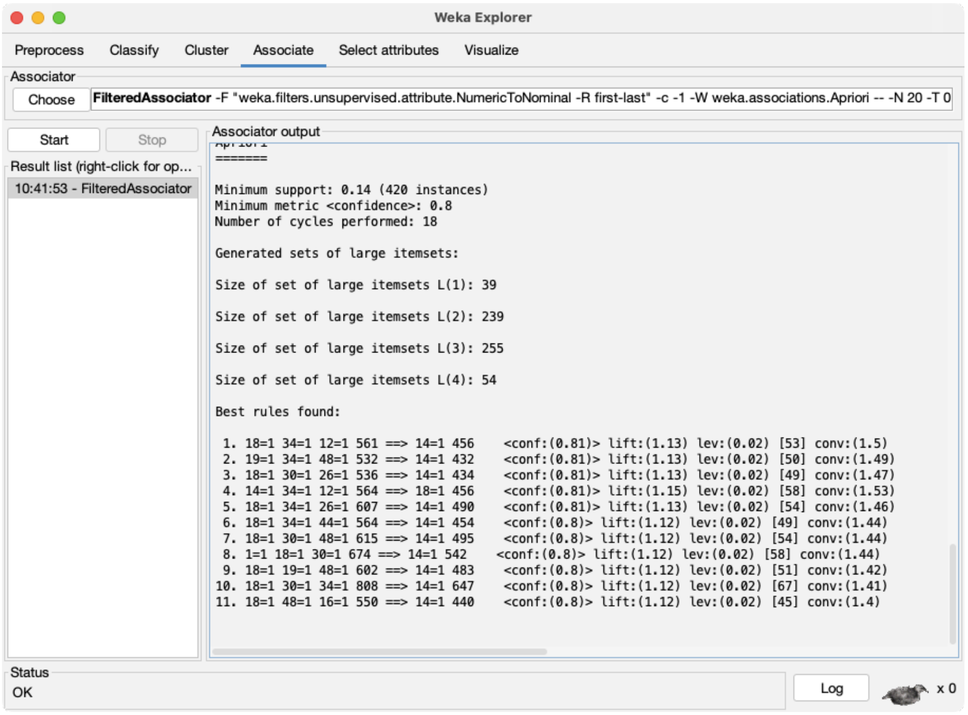
antecedent	consequent	support	confidence	lift
{1 18 30}	{14}	0.181	0.804	1.122
{48 18 19}	{14}	0.161	0.802	1.119
{18 34 30}	{14}	0.216	0.801	1.117
{48 18 30}	{14}	0.165	0.805	1.123
{18 26 30}	{14}	0.145	0.81	1.13
{18 34 44}	{14}	0.151	0.805	1.123
{18 26 34}	{14}	0.163	0.807	1.126
{18 12 34}	{14}	0.152	0.813	1.134
{34 12 14}	{18}	0.152	0.809	1.147
{16 18 48}	{14}	0.147	0.8	1.116
{48 34 19}	{14}	0.144	0.812	1.133

Figure 2. Apriori result with min_sup 0.14 and min_conf 0.8


ibm-2022-2-fp_growth (min_sup: 0.14, min_conf: 0.8)
Possible Data Loss Some features might be lost if you save this workbook in the comma-delimited (.csv) format. To preser...
Save As...
A1
antecedent
ibm-2022-2-fp_growth (min_sup:
Ready
Accessibility: Unavailable
200%

	A	B	C	D	E	F
1	antecedent	consequent	support	confidence	lift	
2	{18 34 30}	{14}	0.216	0.801	1.117	
3	{1 18 30}	{14}	0.181	0.804	1.122	
4	{48 34 19}	{14}	0.144	0.812	1.133	
5	{48 18 19}	{14}	0.161	0.802	1.119	
6	{48 18 30}	{14}	0.165	0.805	1.123	
7	{16 18 48}	{14}	0.147	0.8	1.116	
8	{18 26 34}	{14}	0.163	0.807	1.126	
9	{18 26 30}	{14}	0.145	0.81	1.13	
10	{18 34 44}	{14}	0.151	0.805	1.123	
11	{18 12 34}	{14}	0.152	0.813	1.134	
12	{34 12 14}	{18}	0.152	0.809	1.147	
13						
14						
15						

Figure 3. FP-growth result with min_sup 0.14 and min_conf 0.8



Associator: **FilteredAssociator** -F "weka.filters.unsupervised.attribute.NumericToNominal -R first-last" -c -1 -W weka.associations.Apriori -- -N 20 -T 0

Associator output

```
Apriori
=======

Minimum support: 0.14 (420 instances)
Minimum metric <confidence>: 0.8
Number of cycles performed: 18

Generated sets of large itemsets:

Size of set of large itemsets L(1): 39

Size of set of large itemsets L(2): 239

Size of set of large itemsets L(3): 255

Size of set of large itemsets L(4): 54

Best rules found:

 1. 18=1 34=1 12=1 561 ==> 14=1 456    <conf:(0.81)> lift:(1.13) lev:(0.02) [53] conv:(1.5)
 2. 19=1 34=1 48=1 532 ==> 14=1 432    <conf:(0.81)> lift:(1.13) lev:(0.02) [50] conv:(1.49)
 3. 18=1 30=1 26=1 536 ==> 14=1 434    <conf:(0.81)> lift:(1.13) lev:(0.02) [49] conv:(1.47)
 4. 14=1 34=1 12=1 564 ==> 18=1 456    <conf:(0.81)> lift:(1.15) lev:(0.02) [58] conv:(1.53)
 5. 18=1 34=1 26=1 607 ==> 14=1 490    <conf:(0.81)> lift:(1.13) lev:(0.02) [54] conv:(1.46)
 6. 18=1 34=1 44=1 564 ==> 14=1 454    <conf:(0.8)> lift:(1.12) lev:(0.02) [49] conv:(1.44)
 7. 18=1 30=1 48=1 615 ==> 14=1 495    <conf:(0.8)> lift:(1.12) lev:(0.02) [54] conv:(1.44)
 8. 1=1 18=1 30=1 674 ==> 14=1 542    <conf:(0.8)> lift:(1.12) lev:(0.02) [58] conv:(1.44)
 9. 18=1 19=1 48=1 602 ==> 14=1 483    <conf:(0.8)> lift:(1.12) lev:(0.02) [51] conv:(1.42)
10. 18=1 30=1 34=1 808 ==> 14=1 647    <conf:(0.8)> lift:(1.12) lev:(0.02) [67] conv:(1.41)
11. 18=1 48=1 16=1 550 ==> 14=1 440    <conf:(0.8)> lift:(1.12) lev:(0.02) [45] conv:(1.4)
```

Status
OK

Figure 4. Weka result with min_sup 0.14 and min_conf 0.8

PART 3 MY IBM DATASET ANALYSIS

3.1 Dataset description

File name: my_ibm_data.txt
Command that generated the dataset (on MacOS):

```
./gen lit -ntrans 20 -tlen 10 -nitems 0.018 -npats 5 -conf 0.2 -patlen 20 -randseed -42
```

3.2 High/low min-support

In the beginning, I'm not sure what values should be the high and low min-support for this dataset, so I ran FP-growth on this dataset with min-support ranging from 0.1 to 1.0 and min-confidence 0 (Figure 5).

In Figure 5, we can see that as min-support decreased to less than 0.4, the number of rules generated increased in an exponential-like pattern, whereas the number of frequent itemset is less sensitive to the decrement in min-support.

Therefore, we can see that if we use lower min-support, the exponentially increased number of rules could be a burden for computation.

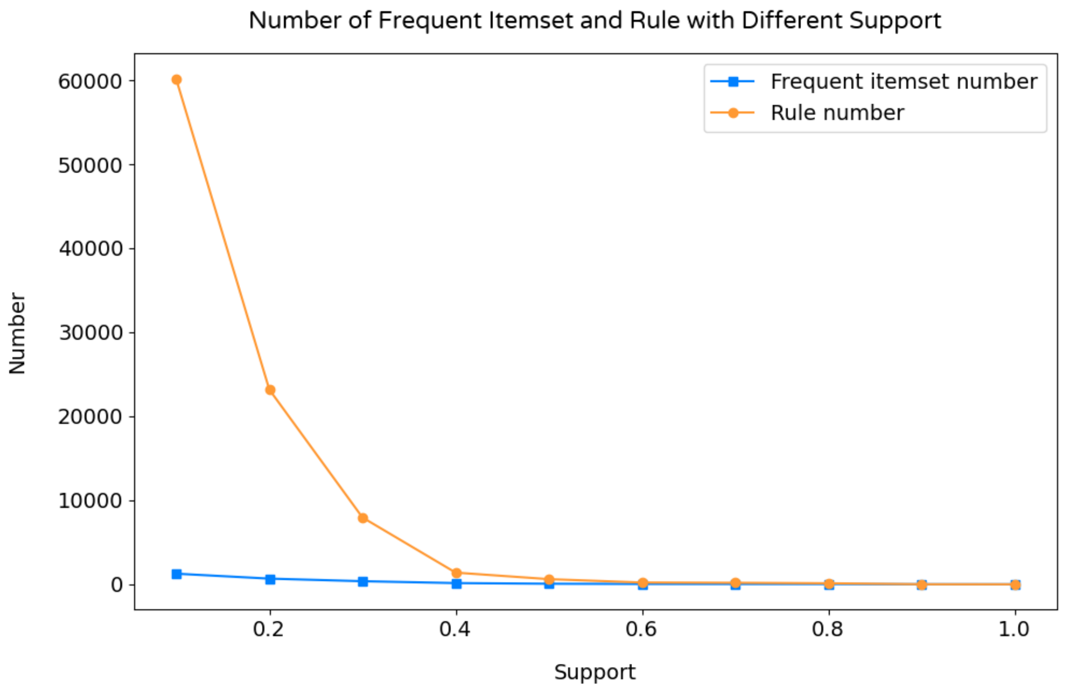


Figure 5. Number of frequent itemset and rule generating from different min-sup

3.3 High/low min-confidence

Next, I ran FP-growth on this dataset with min-support 0.2 and min-confidence ranging from 0 to 1.0 (Figure 6).

In Figure 6, we can see that as min-confidence increased more than 0.2, the number of rules generated decreased in a linear-like pattern.

As for the number of frequent itemset, because it's decided by min-support, and not surprisingly, the number of frequent itemset doesn't change at all under this condition.

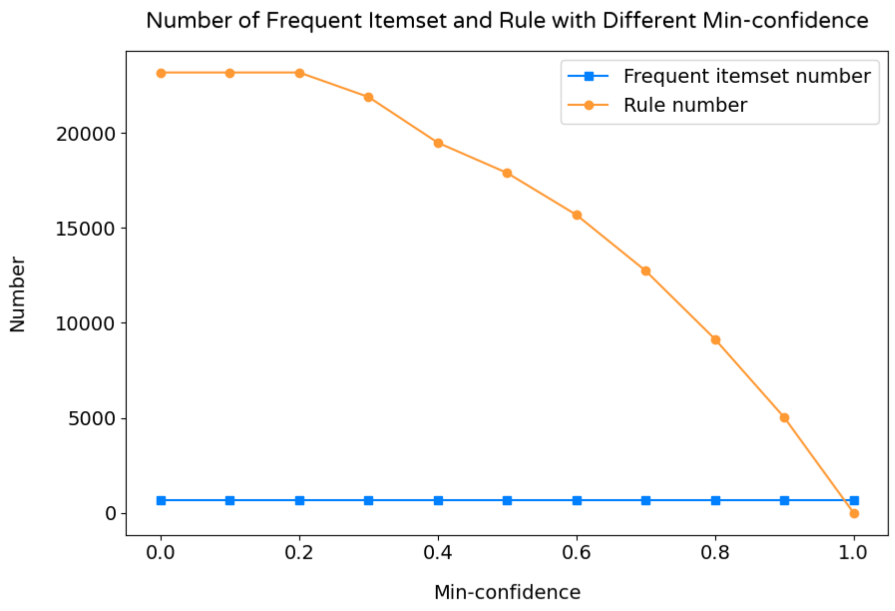


Figure 6. Number of frequent itemset and rule generating from different min-conf

3.4 High/low min-support and min-confidence

From the results of Figure 5 and Figure 6, I decided to pick 0.7 and 0.2 as high and low min-support. As for high and low min-confidence, I decided to pick 0.8 and 0.2, respectively.

Table 1. Result of high/low min-support and min-confidence

Min-sup	Min-conf	Freq Itemset #	Rule #	Average Lift	S.D. of Lift	Max Lift	Min Lift	Time (sec)
0.2	0.2	675	23174	1.549	0.473	2.257	0.799	2.091
0.2	0.8	675	9142	1.528	0.444	2.257	0.987	1.52
0.7	0	31	180	1.035	0.013	1.065	1.008	0.645
0.7	0.8	31	180	1.035	0.013	1.065	1.008	0.722

Next, I ran FP-growth on this dataset with the aforementioned thresholds, I recorded the number of frequent itemset, number of rules generated, average lift, standard deviation of lift, max lift, min lift, and execution time as shown in Table 1.

In Table 1, we can see that in high min-support groups, even if min-confidence increased from 0 to 0.8, the number of rules generated were 180. This could possibly mean that for this dataset, the confidences of rules that were generated from high min-support are pretty high inherently. Even if I picked 0 for min-confidence on purpose, this dataset doesn't contain low-confidence rule with high support. Moreover, we can tell that rules generated from high support are positively correlated since the minimum value of lift was greater than 1 (1.008).

Therefore, it's highly possible that we can say that rules generated from this dataset with high support are highly associated.

As for the low support groups, we can see that the number of rules decreased as the min-confidence increased, which was just as we expected.

Nevertheless, we can see that the maximum value of lift and average lift in low support groups are higher than in high support groups. This means that rules generated from low support groups could have high lift value, and such rules could be interesting to the users. In order to decide a rule is interesting or not, we can assess the rule either in subjective or objective ways. Moreover, how much we understand the domain knowledge about the topic could also play an imperative role in decision making. Nevertheless, due to the fact that this dataset is composed of just number, it's hard to tell whether or not such rules are of interest to the users. Lastly, by such observations, we can see that by applying min-support threshold, it's possibly that we could therefore eliminate rules of interest, which is a limitation of the support-confidence framework [1].

<u>Below are my observations from the 4 scenarios</u>:

- Low min-support and low min-confidence:

  This group generated the largest number of rules, the greatest values of average lift and maximum lift, and the value of minimum lift is the lowest. This means that those rules have a wide range of lift. Some rules could be highly positively correlated or highly negatively correlated.

  This group required the longest execution time.

- Low min-support and high min-confidence:

  With higher value of min-confidence, number of rules generated greatly decreased. The values of average lift and standard deviation decreased as well.

The execution time required was less than the previous group, because the number of rules generated decreased.

- High min-support and low min-confidence:

  The number of rules greatly decreased comparing to the previous 2 groups. The values of average lift, standard deviation of lift, and maximum lift were lower than the previous 2 groups, whereas the minimum value of lift is higher than the previous 2 groups. It seems that the value of lift starts to converge.

  This group requires less execution time than the previous 2 groups

- High min-support high min-confidence:

  Theoretically, the number of rules in this group should be the least. However, due to the characteristic of this dataset and the value for threshold, we can see that this group has the same number of rules with the previous one.

3.5 Execution time for Apriori and FP-growth

Next, I ran Apriori and FP-growth on this dataset, and recorded the execution time.

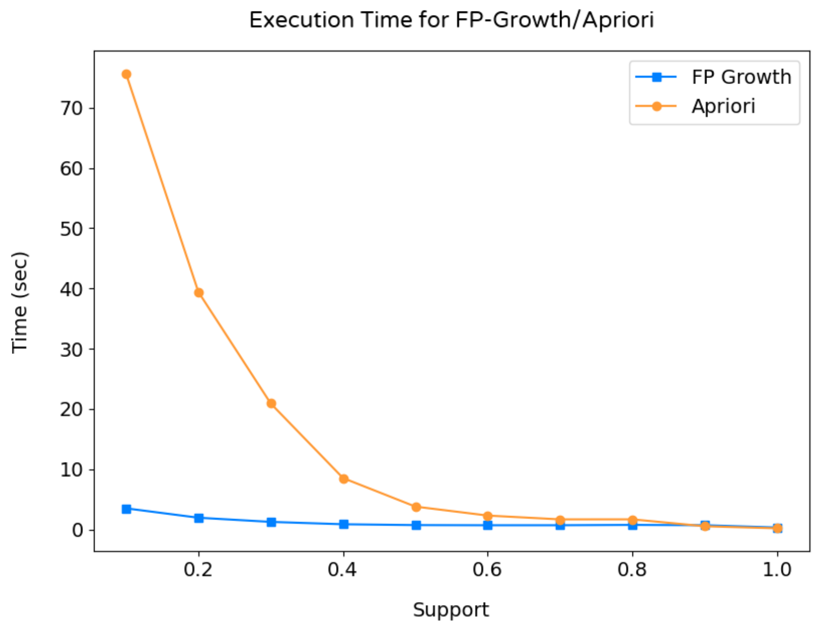


Figure 7. Execution time for Apriori and FP-growth

In Figure 7, we can see that Apriori algorithm is very sensitive to min-support. As the threshold of min-support decreases, the execution time increases in an exponential-like pattern.

3.6 Memory usage & execution time in different transaction size

In order to see how the transaction size can affect memory usage while running association analysis, I created 4 other datasets with the argument `-ntrans 40`, `-ntrans 60`, `-ntrans 80`, `-ntrans 100`, respectively, while other arguments remained the same. The transaction sizes of these 5 datasets are 13841, 27518, 41209, 54869, and 68558.

Next, I ran Apriori on these 5 datasets with min_sup of 0.5 and min_conf of 0.5 (Table 2). The reason why I selected Apriori was because the execution time of Apriori was more sensitive to transaction size comparing to FP-growth, we can see more detailed changes in execution time.

Table 2. Apriori results of datasets with different transaction number

ntrans	Transaction #	Freq Itemset #	Rule #	Average Lift	S.D. of Lift	Max Lift	Min Lift	Time (sec)	Memory (KB)
20	13841	67	612	1.061	0.033	1.144	1.001	4.151	10354.996
40	27518	66	606	1.061	0.032	1.142	1.0	7.815	20526.318
60	41209	66	606	1.061	0.032	1.14	1.0	11.519	29990.782
80	54869	67	612	1.061	0.032	1.14	1.001	15.327	40730.26
100	68558	67	612	1.06	0.032	1.14	1.001	19.393	50174.948

In Figure 8, we can see that as transaction size increased, memory usage also increased. As for execution time, In Figure 9, we can see that the execution time increased in linear pattern as the transaction size increased. These observations are just as expected.

Due to the fact that I changed only one argument to create these 5 datasets, it could be the reason why the execution time increased in linear pattern. In fact, the number of frequent itemset and number of rules generated from these 5 datasets are pretty similar (see Table 2). I wonder what would happen if I alter arguments that increase/decrease the frequent itemset size.

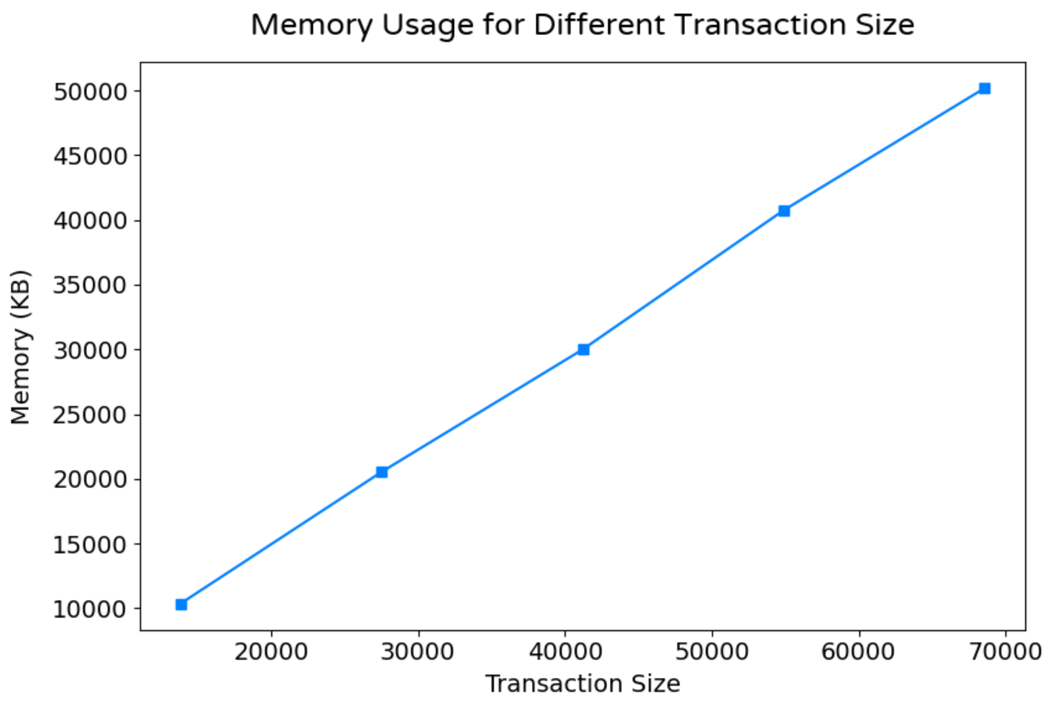


Figure 8. Apriori memory usage for datasets with different transaction size

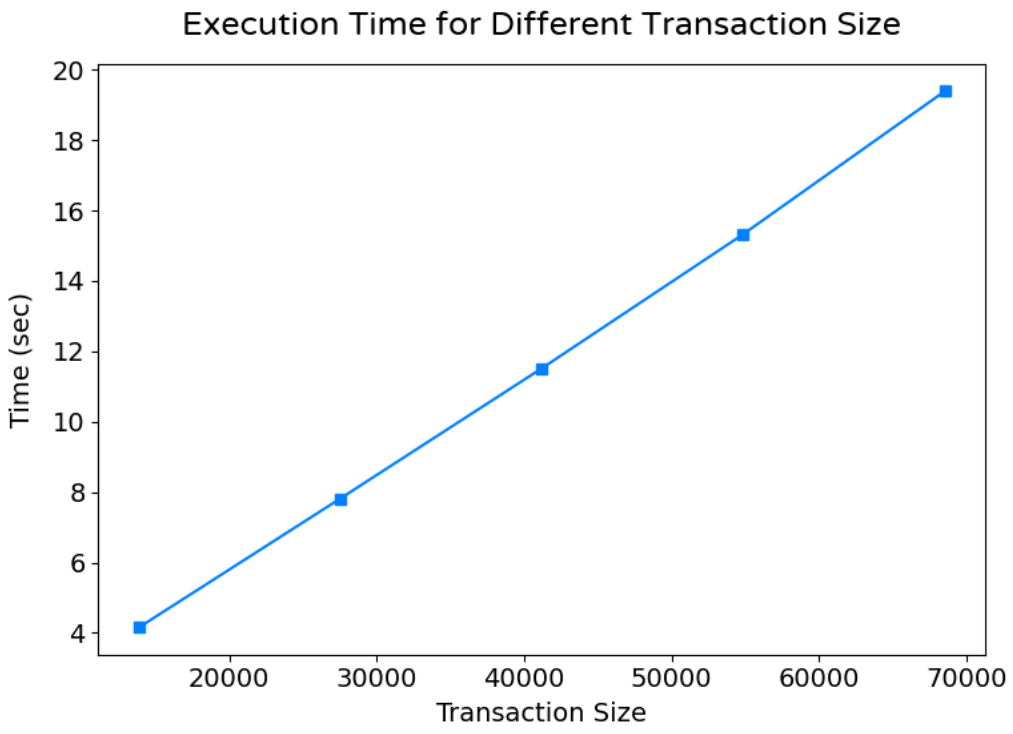


Figure 9. Apriori execution time for datasets with different transaction size

3.7 Number of different items

Therefore, in order to conduct my little experiment, I created a new dataset by the following command:

```
./gen lit -ntrans 40 -tlen 10 -nitems 0.02 -npats 5 -conf 0.2 -patlen 20 -randseed -42
```

Next, I created 4 other datasets with argument `-nitems 0.04`, `-nitems 0.06`, `-nitems 0.08`, and `-nitems 0.1`, while other arguments remained the same. Next, I ran Apriori on those datasets with min-support of 0.5 and min-confidence of 0.5 (Table 3).

Table 3. Apriori results of datasets with varying different item number

nitems	Transaction #	Different Item #	Freq Itemset #	Rule #	Average Lift	Max Lift	Min Lift	Time (sec)	Memory (KB)
0.02	27307	16	163	1446	1.083	1.405	1.011	22.21	21007.876
0.04	27298	28	99	748	1.142	1.431	1.001	14.718	21861.018
0.06	27349	36	49	202	1.121	1.204	1.023	9.969	22634.594
0.08	27341	39	29	60	1.124	1.206	1.003	10.266	23777.08
0.1	27247	44	17	12	1.143	1.167	1.12	9.137	24113.43

In Table 3, we can see that as nitems increased, number of different items increased as well. However, frequent itemset number and rule number decreased as different item number increased. It's no surprise that since different number increased and transaction number didn't increase, number of frequent itemset will be decreased.

As for execution time (Figure 10), we can see that as different item number increased, execution time decreased. I think it's due to the decreased number of frequent itemset.

The most interesting thing was the memory usage (Figure 11), we can see that memory usage increased as different item number increased. I originally thought that since frequent itemset number decreased, memory usage should decrease as well, but the result was not as the same as I thought. Next, I analyzed the number of C_k and L_k created during Apriori of each dataset, and the result is shown in Table 4 and Table 5. I thought maybe it was because more different items could create a greater number of C_k and, therefore, more memory space was required. Nevertheless, result in Table 4 and Table 5 didn't show any evidence supporting my assumption. So far, I'm still not sure what caused this result.

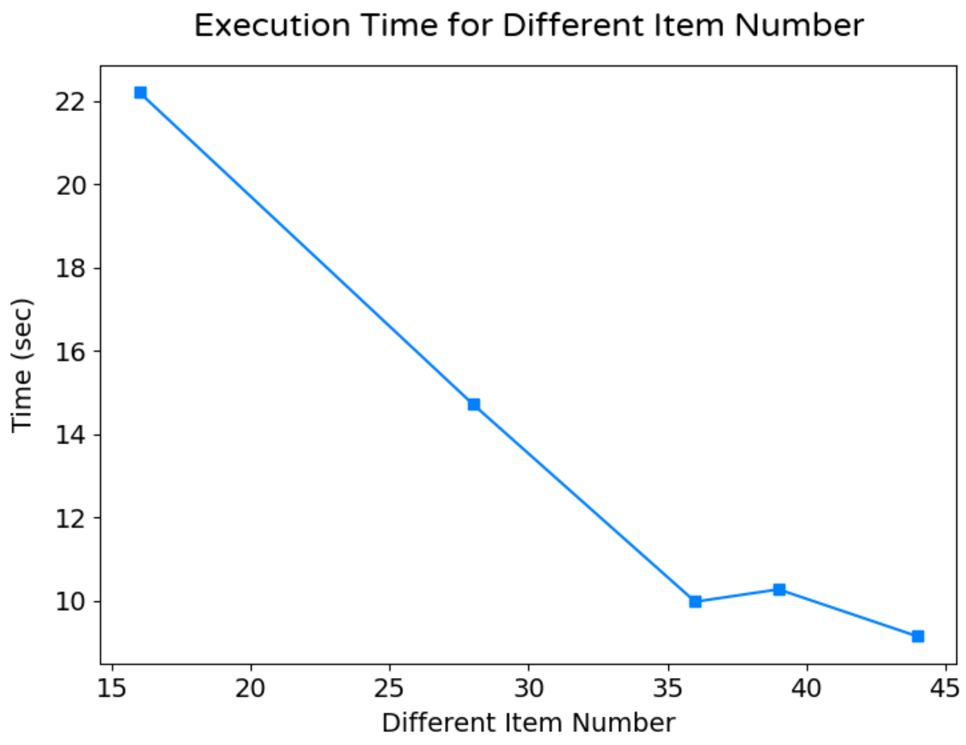


Figure 10. Apriori execution time for datasets with different item numbers

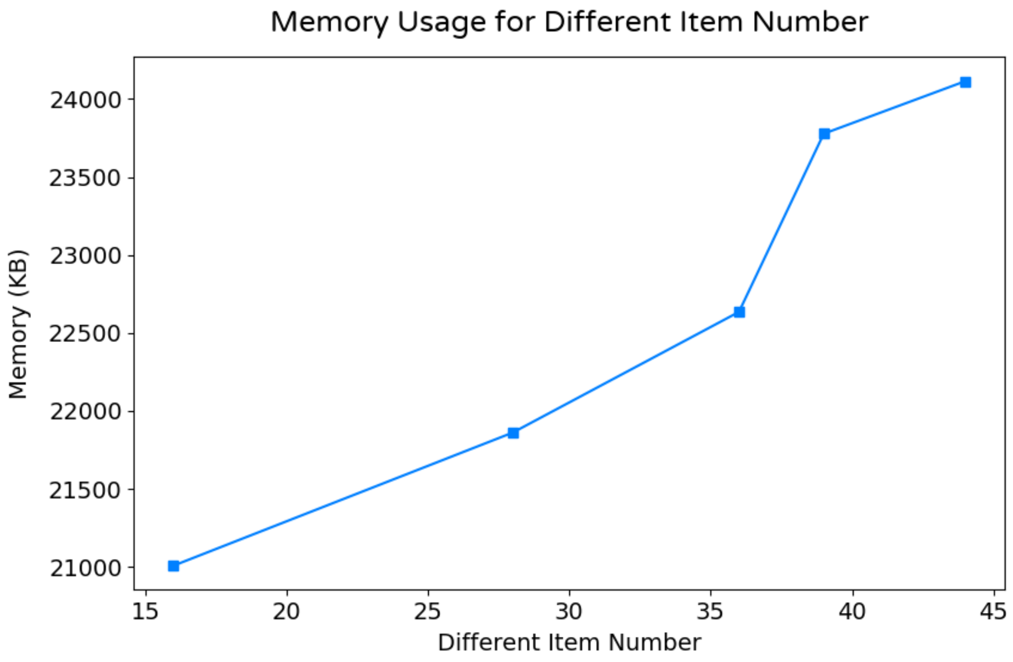


Figure 11. Apriori memory usage for datasets different item numbers

Table 4. Number of C_k created during Apriori

nitems	C_1	C_2	C_3	C_4	C_5	C_6	Sum
0.02	16	45	86	40	13	1	201
0.04	29	55	47	19	6	1	157
0.06	37	66	14	5	1	0	123
0.08	40	78	10	1	0	0	129
0.1	45	78	1	0	0	0	124

Table 5. Number of L_k created during Apriori

nitems	L_1	L_2	L_3	L_4	L_5	L_6	Sum
0.02	10	40	59	40	13	1	163
0.04	11	33	29	19	6	1	99
0.06	12	21	10	5	1	0	49
0.08	13	11	4	1	0	0	29
0.1	13	3	1	0	0	0	17

PART 4 KAGGLE DATASET ANALYSIS

4.1 Dataset description

Dataset: Market Basket Optimisation
https://www.kaggle.com/datasets/d4rklucif3r/market-basket-optimisation

4.2 High/low min-support and min-confidence

By doing similar steps in 3.2 and 3.3, I picked 0.04 and 0.004 as high and low min-sup, and 0.3 and 0.05 as high and low min-conf.

Table 6. Result of high/low min-support and min-confidence

Min-sup	Min-conf	Freq itemset #	Rule #	Average lift	S.D. of Lift	Max lift	Min Lift	Time (sec)
0.004	0.05	959	2149	1.759	0.602	4.844	0.39	4.451
0.004	0.3	959	364	2.034	0.514	4.701	1.264	4.509
0.04	0.05	35	10	1.455	0.199	1.748	1.189	1.177
0.04	0.3	35	4	1.522	0.173	1.748	1.347	1.354

In Table 6, we can see that in both high and low min-support groups, the values of average lift increased as the min-confidence increased from 0.05 to 0.3. This observation wasn't observed in the previous IBM dataset. This could be the reason that with the increment in min-confidence, rules with lower value of lift were excluded, and, therefore, the average lift increased. This assumption corresponds to the changes in minimum lift, which increased in both high/low min-support groups as min-confidence increased.

Furthermore, in low min-support groups, the value of minimum lift increased from 0.39 to 1.264. This observation indicates that as min-confidence increased, negatively correlated (lift < 1) rules were excluded.

As a result, we can say that for this dataset, we can obtain more correlated rules with higher min-confidence threshold.

Below are my observations from the 4 scenarios:

- Low min-support and low min-confidence:

  This group generated the largest number of rules, the greatest value of maximum lift, and the value of minimum lift is the lowest. Just as I mentioned in the IBM dataset, this means that those rules have a wide range of lift. Some rules could be highly positively correlated or highly negatively correlated.

- Low min-support and high min-confidence:

  Just the same as we observed in the IBM dataset, with higher value of min-confidence, number of rules generated decreased, which was as we expected. The only difference was that the value of average lift increased.

- High min-support and low min-confidence:

  The number of rules greatly decreased comparing to the previous 2 groups. The values of average lift, standard deviation of lift, and maximum lift were lower than the previous 2 groups, whereas the minimum value of lift is higher than the previous 2 groups.

- High min-support high min-confidence:

  This group had the least number of rules. The value of average lift is higher than the high min-support and low min-confidence group, and the reason is described previously.

4.3 Rules generated from high min-support and high min-confidence group

Table 7. Rules generated with min_sup 0.04 and min_conf 0.3

Antecedent	Consequent	Support	Confidence	Lift
{35}	{15}	0.06	0.343	1.439
{40}	{15}	0.053	0.321	1.347
{26}	{15}	0.048	0.37	1.554
{62}	{15}	0.041	0.417	1.748

Table 8. Rules generated with min_sup 0.04 and min_conf 0.3

Antecedent	Consequent	Support	Confidence	Lift
Spaghetti	Mineral water	0.06	0.343	1.439
Chocolate	Mineral water	0.053	0.321	1.347
Milk	Mineral water	0.048	0.37	1.554
Ground Beef	Mineral water	0.041	0.417	1.748

Table 7 and Table 8 are rules generated from high min-support and high-confidence group. We can see that these 4 rules all imply the purchase of mineral water. Next, for the purpose of confirming these 4 rules are misleading strong association rules or not, I checked the support of {mineral water}, which is 0.238, and that means the probability of purchasing mineral water is 0.238. Nevertheless, if we purchase spaghetti, or chocolate, or milk, or ground beef, the probability of purchasing mineral water will increase from 0.238 to 0.343, 0.321, 0.37, and 0.417, respectively. This result showed that the purchase of one of these items in antecedent can increase the likelihood of purchasing mineral water, and, therefore, it's positively correlated. This observation corresponds to the values of lift of these 4 rules, which are all greater than 1, indicating positively correlated.

4.4 Size of frequent k-itemset in different min-supports

Next, I'd like to analyze the effect of different min-supports on the size of each k-itemset, so I performed FP-growth on the dataset with min-support of 0.001, 0.005, and 0.01, respectively.

In Figure 12, we can see that with lower min-support, bigger size of frequent k-itemset will be obtained, especially 2-itemset and 3-itemset. With bigger frequent k-itemset size, the computational complexity increases as well. Therefore, we can see that this is also a factor that could contribute to the burden while performing association analysis with low min-support.

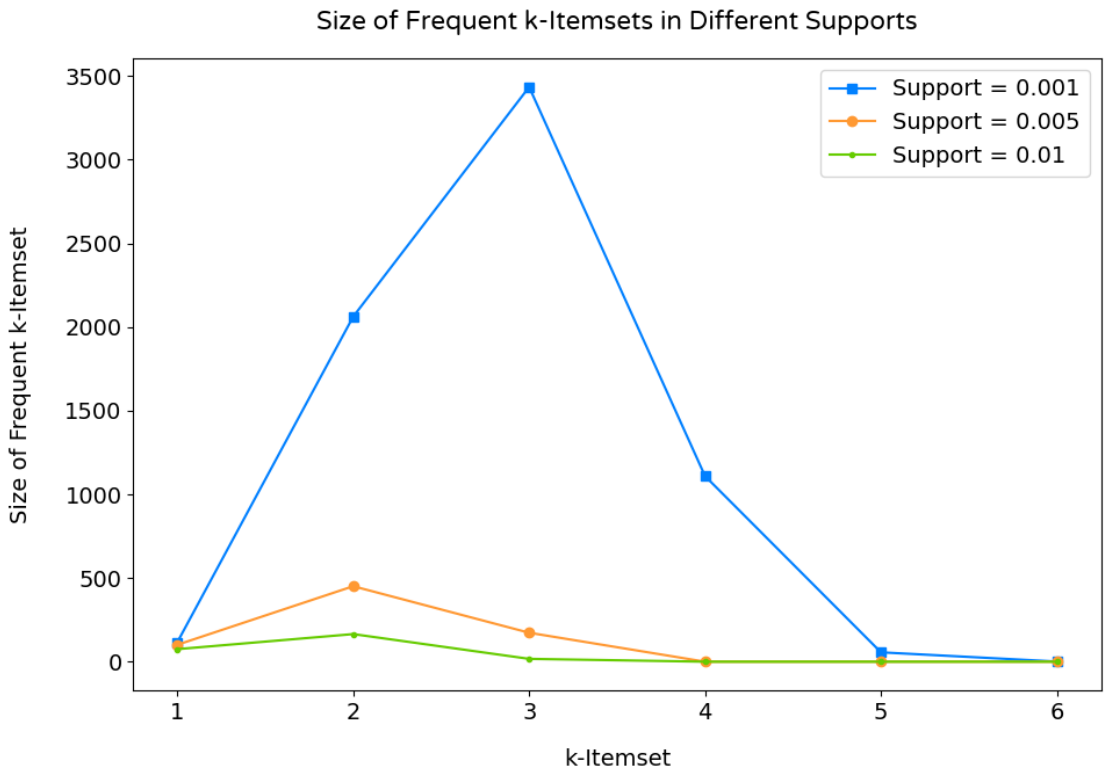
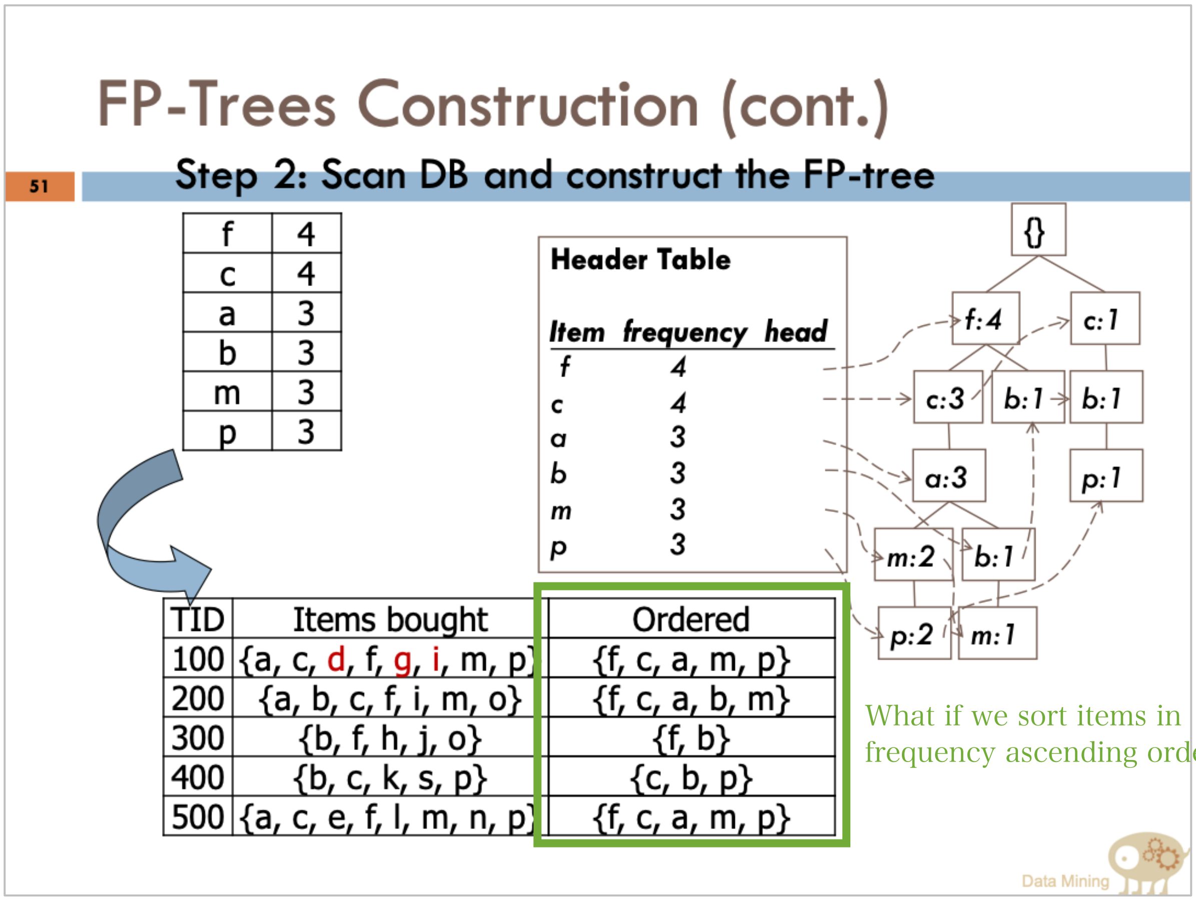


Figure 12. Size of frequent k-itemset generated from different min-support

4.5 What if we sort items in frequency ascending order?



FP-Trees Construction (cont.)

51

Step 2: Scan DB and construct the FP-tree

f	4
c	4
a	3
b	3
m	3
p	3

Header Table

Item	*frequency*	*head*
f	*4*	
c	*4*	
a	*3*	
b	*3*	
m	*3*	
p	*3*	

TID	Items bought	Ordered
100	{a, c, d, f, g, i, m, p}	{f, c, a, m, p}
200	{a, b, c, f, i, m, o}	{f, c, a, b, m}
300	{b, f, h, j, o}	{f, b}
400	{b, c, k, s, p}	{c, b, p}
500	{a, c, e, f, l, m, n, p}	{f, c, a, m, p}

What if we sort items in frequency ascending order?

Figure 13. One step in FP-tree construction

Out of curiosity, I wonder what would happen if we sort items in each transaction in frequency ascending order (see Figure 13)?

So, I conducted this little experiment, and the result is shown in Figure 14. In Figure 14, we can see that the execution time of sorting in frequency ascending order was more sensitive to min-support. More execution time was required as min-support decreased. Nevertheless, both versions of FP-growth algorithms generated the same association rules. More details about the execution time will be discussed in Discussion 5.1.

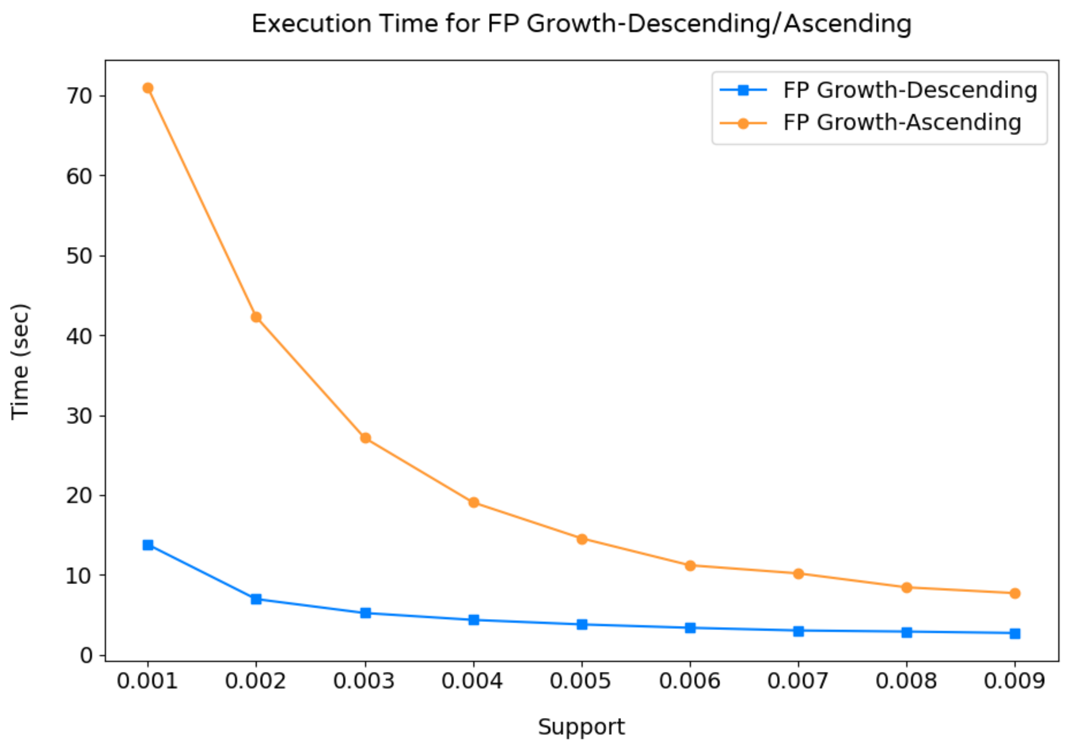


Figure 14. Execution time for FP-growth with descending/ascending order

PART 5 DISCUSSION

5.1 FP-growth with items sorted in frequency ascending/descending order

Below are the possible reasons I figured out why it takes much more time for FP-growth with items sorted in frequency ascending order to execute.

If we sort items in frequency ascending order and use such ordered items to construct FP tree, we will get a tree that has less frequent items closer to root and more frequent items closer to leaf. So, the tree will be very broad, and children of the root are those very less frequent items. As a result, when we try to count the support count of each item on prefix paths, it will take much more time to do the counting, especially for those more frequent items.

Moreover, the root branches with less frequent items, and such branches are less shared with other items. We can expect that the size of such FP tree is bigger than descending-ordered tree. Therefore, more memory is needed to perform this version of FP growth.

5.2 Memory issue in powerset calculation

One issue I encountered while doing this project was the out-of-memory issue. In the beginning, my first version of FP-growth was slightly different from the original FP-growth. Below are the steps of my first version of FP-growth:

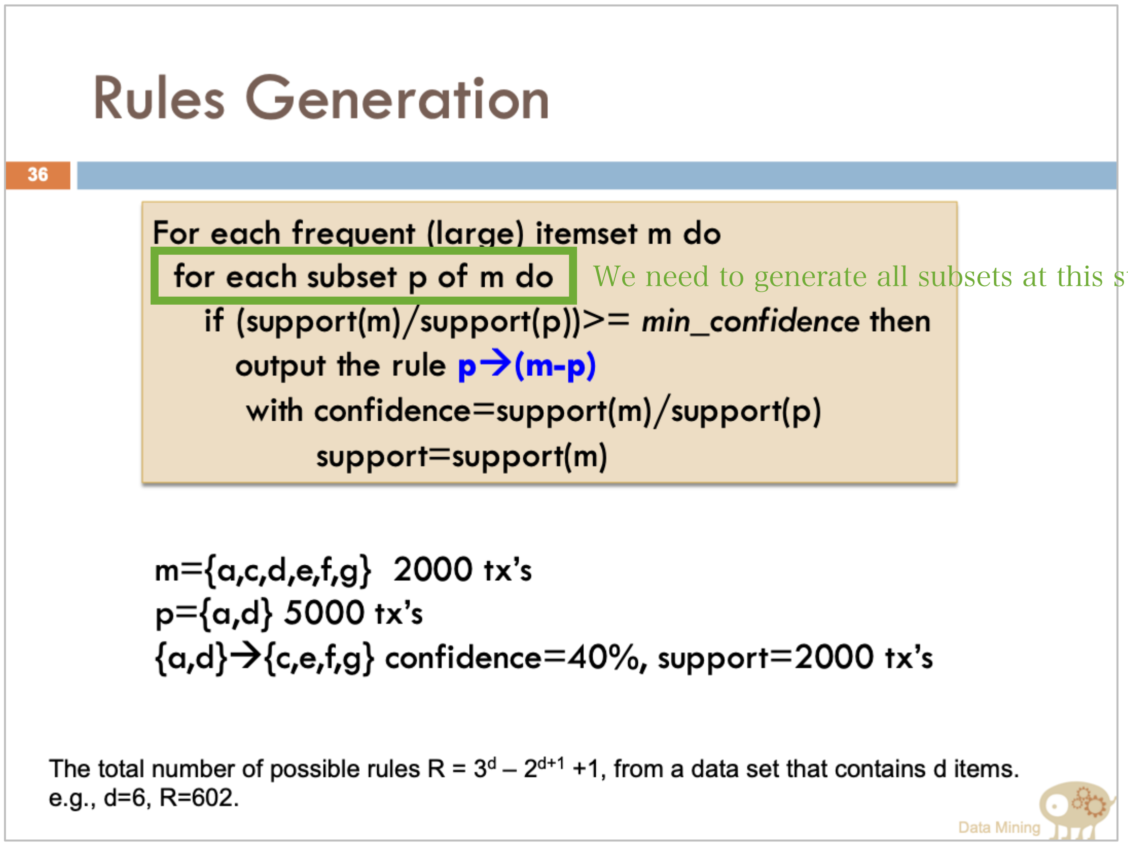
1. After FP tree construction, gather all the prefix paths for each item.
2. For each prefix path of an item, create all the subsets containing items on the prefix path.
    a. Store the subsets in a dictionary with the subset as key and the count as value.
    b. If a set is already in the dictionary, accumulate the count.
3. After storing all the subsets of each prefix path, loop through the dictionary and discard the set whose count is less than minimum support count.
4. The remaining sets in the dictionary are frequent itemset.

This method did actually work for most datasets, but with the release of the IBM-2022-2 dataset, this method encountered out-of-memory issue. I found it was the ‘creating all the subset’ in step 2 that caused the memory issue.

$$|A| = n \implies |P(A)| = 2^n$$

Due to the theorem of powerset, memory usage in the ‘creating all the subset’ in step 2 grows in exponential pattern. The longest prefix path for IBM-2022-2 dataset contained 27 items, and $2^{27} = 134{,}217{,}728$. A 1-element set takes 216 bytes of memory in my MacBook Air (Early 2015), and it takes $216 \times 2^{27} = 28{,}991{,}029{,}248 \text{ byte} \approx 29 \text{ GB}$ memory space for 2^{27} sets. Unfortunately, my MacBook Air (Early 2015) has only 8 GB memory. Furthermore, the 29 GB is calculated based on only 1-element set, and it definitely consumes much more memory space in powerset calculation.

It was the first time that I realized how a good/bad algorithm can affect the efficiency of solving a problem. And, fortunately, this problem was solved by using other ways to implement FP-growth.



Rules Generation

36

```
For each frequent (large) itemset m do
  for each subset p of m do
    if (support(m)/support(p))>= min_confidence then
      output the rule p→(m-p)
        with confidence=support(m)/support(p)
             support=support(m)
```

We need to generate all subsets at this step

m={a,c,d,e,f,g} 2000 tx's
p={a,d} 5000 tx's
{a,d}→{c,e,f,g} confidence=40%, support=2000 tx's

The total number of possible rules $R = 3^d - 2^{d+1} + 1$, from a data set that contains d items.
e.g., d=6, R=602.

Figure 15. One step in rules generation

Nevertheless, it occurs to me that while generating rules, there’s one step that we also have to create all the subset from a frequent itemset, which is labeled in Figure 15. As a result, I’m curious if there’s such a dataset that contains frequent itemset of very big size, this issue still occurs. Despite the fact that we have some technique to prune the number of rules to generate, such situation could still occur.

I tried to find papers talking about such issue, and I did find some papers that aimed to solve memory issue. However, I didn't have enough time to read those papers thoroughly.

5.3 High/low min-support and min-confidence

According to the above 2 datasets, the characteristics of rules generated from high/low min-support/min-confidence are listed below:

- High min-support and high min-confidence:

  Rules generated from this setting have higher probability to be seen among the dataset due to the higher support. The high confidence suggests a high probability of co-occurrence between items in antecedent and items in consequent [1]. Nevertheless, high min-support and high min-confidence doesn't actually indicate causality [1]. We have to use other metrics to help us interpret the generated association rules, such as lift. Usually, the number of rules generated from this group is less than the number of rules generated from other 3 groups

- High min-support and low min-confidence:

  Usually, the number of rules generated from this group is more than the high min-sup and high min-conf group, but, for some datasets, the numbers of rules in these 2 groups are the same. For such dataset, it could mean that rules generated from high min-support usually also have high confidence value.

- Low min-support and high min-confidence:

  This group generates the second largest number of rules among these 4 groups. Usually, the values of average lift, standard deviation of lift, maximum of lift are greater than the previous 2 groups, and the value of minimum lift is also usually smaller than the previous 2 groups. It's because this group generates more rules, and those rules are with a variety of lift.

- Low min-support and low min-confidence:

  This group generates the largest number of rules among these 4 groups. The value of average lift is not necessary to be higher than the low min-support and high min-confidence group, because this group could generate too many rules with low values of lift, and, therefore, decreases the value of average lift. Usually, this group generates the maximum and minimum values of lift. For the 2 datasets I used in this project, rules with higher value of lift were generated from low support groups. This observation could imply rules of interest could be generated from low min-support threshold.

PART 6 CONCLUSIONS

Both Apriori and FP growth are important association analysis algorithms. Apriori requires much more execution time due to multiple scanning of dataset, whereas FP growth scan only once and less execution time is required. Despite the fact that the support-confidence framework plays a crucial role in mining association rules, using other metrics, such as lift, could help us determine whether a rule is of interest.

Execution time and memory usage could be potential issues while doing association analysis. Several factors could contribute to the occurrence of such issues, such as number of transactions, number of frequent items in each transaction, too lower value for min-support, how we implement the algorithm, and hardware resources, etc. Despite that fact that such event doesn't occur often, we should still be careful about those situations while performing association analysis.

ACKNOWLEDGEMENTS

Last but not least, I would like to thank TAs in this course. While doing this project, I encountered some stupid problems. Thankfully, TAs were always willing to answer my questions and give me useful suggestions, which helped me a lot. I really appreciate TAs' patience and help!

References

1. Tan, P.N., Steinbach, M., Karpatne, A., & Kumar, V. (2019). Association Analysis: Basic Concepts and Algorithms. In Tan, P.N., Steinbach, M., Karpatne, A., & Kumar, V. (Ed.), *Introduction to Data Mining, 2nd Edition* (pp. 213-306). New York, YK: Pearson Education Limited.
2. Han, J., Kamber, M., & Pei, J. (2012). Mining Frequent Patterns, Associations, and Correlations: Basic Concepts and Methods. In Han, J., Kamber, M., & Pei, J. (Ed.), *Data Mining: Concepts and Techniques, 3rd Edition* (pp. 243-278). Waltham, MA: ELSEVIER.
3. Slimani, T., & Lazzez, A. (2014). Efficient analysis of pattern and association rule mining approaches. *arXiv preprint arXiv:1402.2892*.
4. Hikmawati, E., Maulidevi, N. U., & Surendro, K. (2021). Minimum threshold determination method based on dataset characteristics in association rule mining. *Journal of Big Data*, *8*(1), 1-17.

APPENDIX A

Item-and-number sheet for the Market Basket Optimisation dataset from Kaggle

1: shrimp	41: chicken	81: corn
2: almonds	42: oil	82: yogurt cake
3: avocado	43: fresh tuna	83: mint
4: vegetables mix	44: tomatoes	84: butter
5: green grapes	45: black tea	85: asparagus
6: whole weat flour	46: extra dark chocolate	86: French wine
7: yams	47: protein bar	87: salt
8: cottage cheese	48: red wine	88: tea
9: energy drink	49: pasta	89: barbecue sauce
10: tomato juice	50: pepper	90: mayonnaise
11: low fat yogurt	51: shampoo	91: zucchini
12: green tea	52: rice	92: carrots
13: honey	53: sparkling water	93: mushroom cream sauce
14: salad	54: ham	94: candy bars
15: mineral water	55: body spray	95: chili
16: salmon	56: pancakes	96: mashed potato
17: antioxydant juice	57: grated cheese	97: nonfat milk
18: frozen smoothie	58: white wine	98: water spray
19: spinach	59: toothpaste	99: chocolate bread
20: olive oil	60: parmesan cheese	100: mint green tea
21: burgers	61: fresh bread	101: eggplant
22: meatballs	62: ground beef	102: blueberries
23: eggs	63: escalope	103: bacon
24: chutney	64: herb & pepper	104: fromage blanc
25: turkey	65: tomato sauce	105: gluten free bar
26: milk	66: magazines	106: dessert wine
27: energy bar	67: strawberries	107: flax seed
28: whole wheat rice	68: strong cheese	108: hand protein bar
29: whole wheat pasta	69: pickles	109: sandwich
30: french fries	70: cake	110: babies food
31: soup	71: hot dogs	111: melons
32: light cream	72: brownies	112: cauliflower
33: shallot	73: cereals	113: green beans
34: frozen vegetables	74: clothes accessories	114: ketchup
35: spaghetti	75: bug spray	115: bramble
36: pet food	76: muffins	116: burger sauce
37: cookies	77: light mayo	117: oatmeal
38: cooking oil	78: gums	118: asparagus
39: champagne	79: soda	119: cream
40: chocolate	80: cider	120: napkins