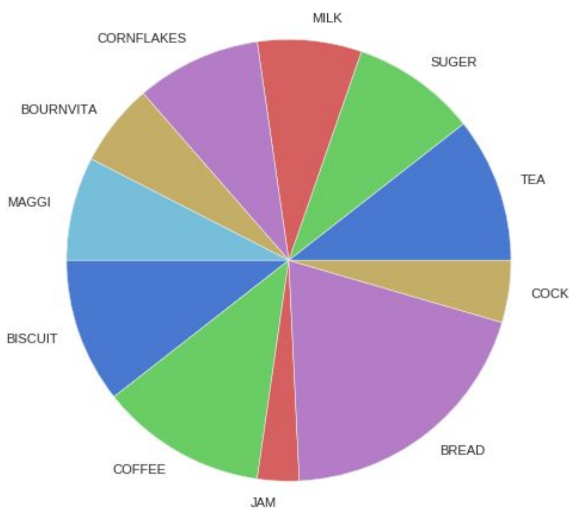
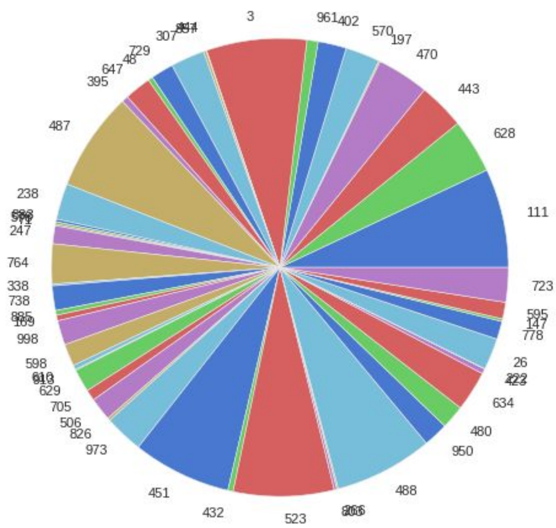
# Data Mining Project1

醫資所 Q56084072 陳怡君

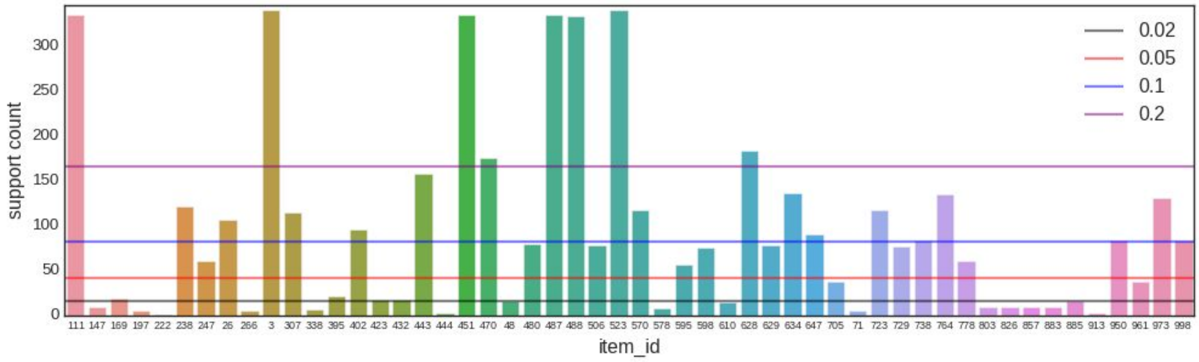
- Datasets
    - Kaggle: Supermarket/GroceryStoreDataSet.csv
        - Description: 20 transaction records , with 11 items.



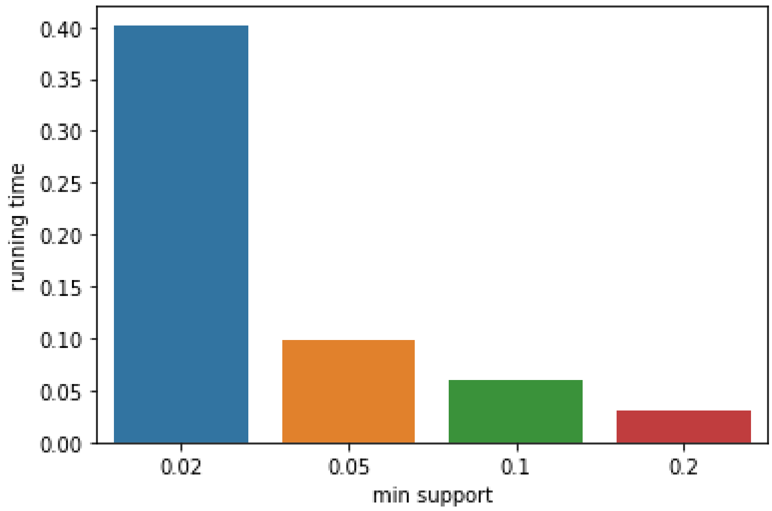

    - IBM:
        - Description: 1000 transaction records with average 5 items per transaction, 1000 items and 20 patterns



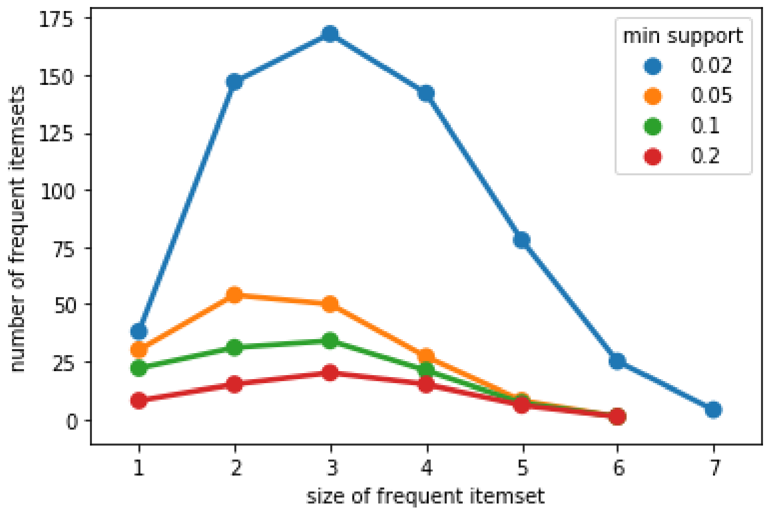
- Results
  - Performance of Apriori
    - With IBM datasets and min support 0.02, 0.05, 0.1 and 0.2.




(Figure 1-1. item_id vs. corresponding support count. Horizontal lines are min supports.)

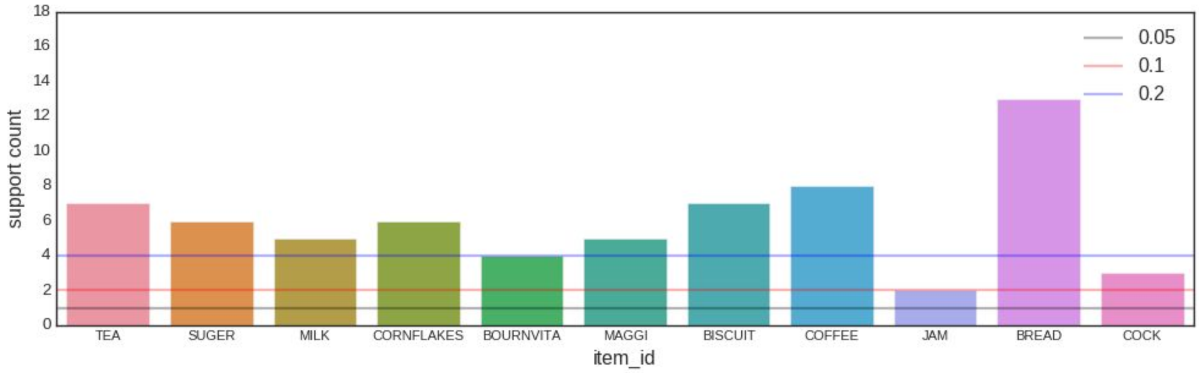



(Figure 1-2. min support vs. corresponding running time. )

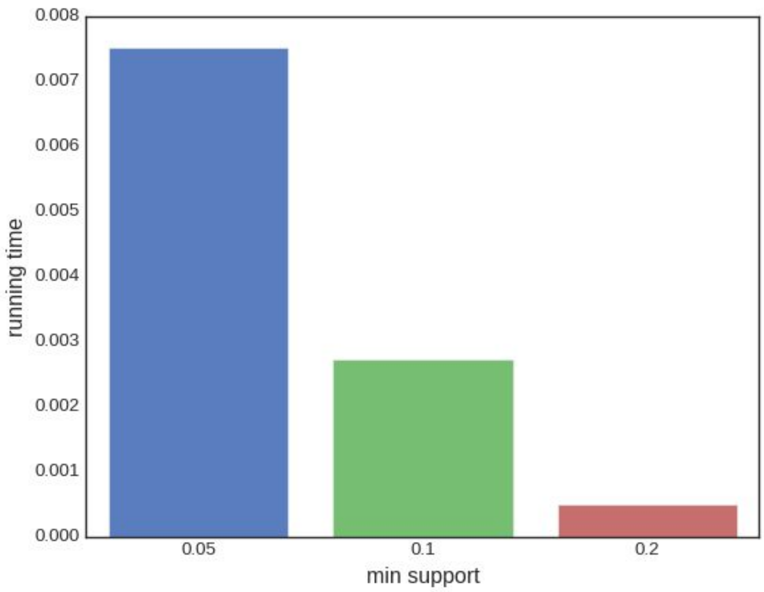



(Figure 1-3. size of frequent itemset vs. number of frequent itemsets. Different colors presents different min support.)

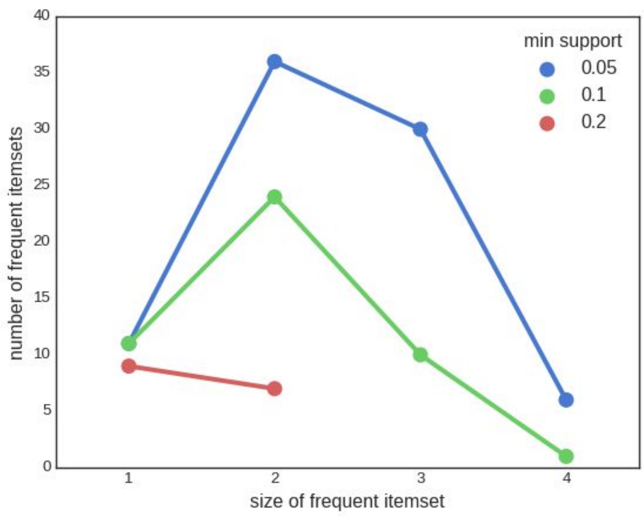
■ With kaggle datasets with min support 0.05, 0.1 and 0.2.




(Figure2-1. item_id vs. corresponding support count. Horizontal lines are min supports.)



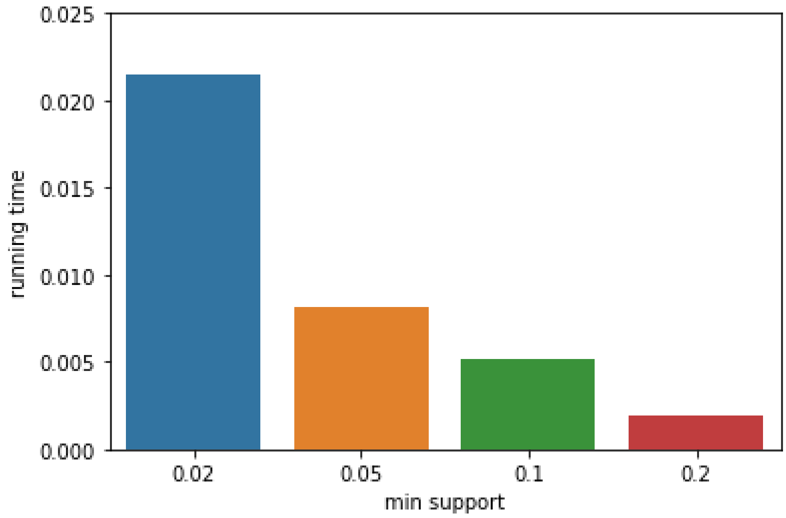

(Figure 2-2. min support vs. corresponding running time. )




(Figure 2-3. size of frequent itemset vs. number of frequent itemsets. Different colors presents different min support.)

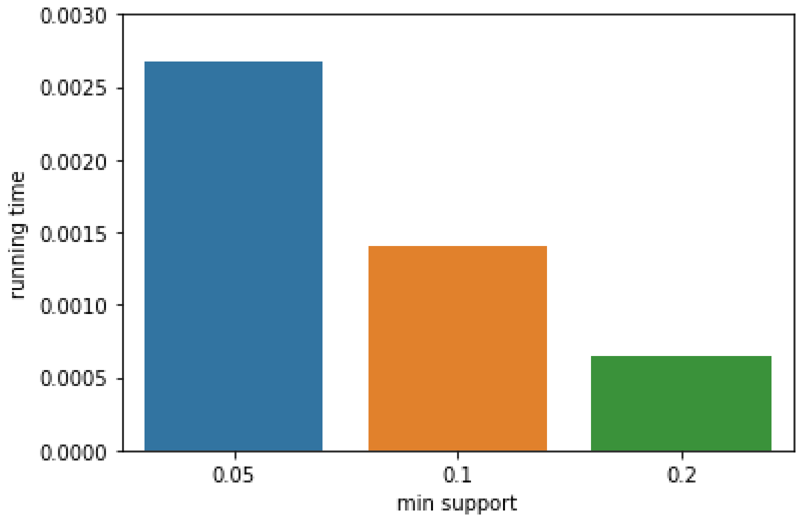
- ➔ Figure 1-1 and 2-1 show the thredsholds of different min support. Smaller min support means smaller thredshold, causing more items remained to be frequent 1-items.
- ➔ Therefore, from figure 1-2 and 2-2 and figure 1-3 and 2-3, smaller min support causing larger running time because of larger frequent itemsets.
- ➔ Figure 1-3 and 2-3 shows the number of frequent itemsets grows at first because of more k-items generated, and converges in the end because of fewer (k-1)-items.

- ○ Performance of Frequent Pattern -Growth
  - ■ With IBM datasets and min support 0.02, 0.05, 0.1 and 0.2.



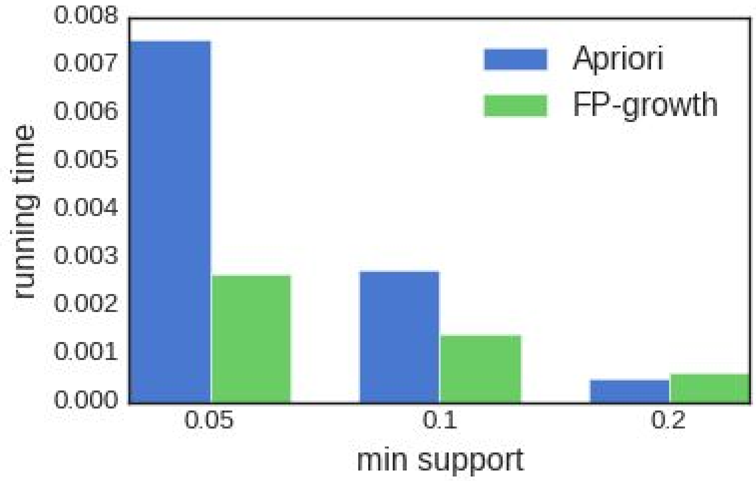

(Figure 3. min support vs. corresponding running time. Running on IBM datasets)

  - ■ With kaggle datasets with min support 0.05, 0.1 and 0.2.




(Figure 4. min support vs. corresponding running time. Running on kaggle datasets)

- Performance comparison
  - Comparing the running time of producing frequent itemsets.




(Figure 5. Apriori vs. FP-growth with corresponding running time of producing frequent itemsets)

- ➔ Apparently, FP-growth argorithm cost less running time to produce frequent itemsets.
- ➔ Apriori algorithm needs to scan database every time when computing support count of candidate itemsets, and it produces as more candidate itemsets as it can.
- ➔ FP-growth algorithm only need to scan overall database twice when pruning out un-frequent items and creating tree.
- ➔ FP-growth algorithm doesn't need to produce candidate itemsets, so it costs less running time. Therefore, FP-growth algorithm runs faster than Apriori algorithm.

- Verifying with WEKA
  - Verifying frequent itemsets.
    - Running on IBM datasets: WEKA vs. Apriori algorithm and FP-growth

```
Minimum support: 0.05 (41 instances)
Size of set of large itemsets L(1): 30

Size of set of large itemsets L(2): 54

Size of set of large itemsets L(3): 50

Size of set of large itemsets L(4): 27

Size of set of large itemsets L(5): 8

Size of set of large itemsets L(6): 1

Minimum support: 0.1 (83 instances)
Size of set of large itemsets L(1): 22

Size of set of large itemsets L(2): 31

Size of set of large itemsets L(3): 34

Size of set of large itemsets L(4): 21

Size of set of large itemsets L(5): 7

Size of set of large itemsets L(6): 1

Minimum support: 0.2 (166 instances)
Size of set of large itemsets L(1): 8

Size of set of large itemsets L(2): 15

Size of set of large itemsets L(3): 20

Size of set of large itemsets L(4): 15

Size of set of large itemsets L(5): 6

Size of set of large itemsets L(6): 1
```

Apriori

```
====min support:0.05====
Size of set of large itemsets L(1): 30
Size of set of large itemsets L(2): 54
Size of set of large itemsets L(3): 50
Size of set of large itemsets L(4): 27
Size of set of large itemsets L(5): 8
Size of set of large itemsets L(6): 1
====min support:0.1====
Size of set of large itemsets L(1): 22
Size of set of large itemsets L(2): 31
Size of set of large itemsets L(3): 34
Size of set of large itemsets L(4): 21
Size of set of large itemsets L(5): 7
Size of set of large itemsets L(6): 1
====min support:0.2====
Size of set of large itemsets L(1): 8
Size of set of large itemsets L(2): 15
Size of set of large itemsets L(3): 20
Size of set of large itemsets L(4): 15
Size of set of large itemsets L(5): 6
Size of set of large itemsets L(6): 1
```

FP-growth

```
====min support:  0.02 ====
size of all frequent itemsets:  513
====min support:  0.05 ====
size of all frequent itemsets:  164
====min support:  0.1 ====
size of all frequent itemsets:  105
====min support:  0.2 ====
size of all frequent itemsets:  62
```

(Figure 6. Producing frequent itemsets of IBM datasets by WEKA, Apriori and FP-growth.)

    - Running kaggle datasets: WEKA vs. Apriori algorithm and FP-growth

```
Minimum support: 0.05 (1 instances)
Size of set of large itemsets L(1): 11

Size of set of large itemsets L(2): 36

Size of set of large itemsets L(3): 30

Size of set of large itemsets L(4): 6

Minimum support: 0.1 (2 instances)
Size of set of large itemsets L(1): 11

Size of set of large itemsets L(2): 24

Size of set of large itemsets L(3): 10

Size of set of large itemsets L(4): 1

Minimum support: 0.2 (4 instances)
Size of set of large itemsets L(1): 9

Size of set of large itemsets L(2): 7
```

Apriori

```
Minimum support: 0.05 (1 instances)
Size of set of large itemsets L(1): 11

Size of set of large itemsets L(2): 36

Size of set of large itemsets L(3): 30

Size of set of large itemsets L(4): 6

Minimum support: 0.1 (2 instances)
Size of set of large itemsets L(1): 11

Size of set of large itemsets L(2): 24

Size of set of large itemsets L(3): 10

Size of set of large itemsets L(4): 1

Minimum support: 0.2 (4 instances)
Size of set of large itemsets L(1): 9

Size of set of large itemsets L(2): 7
```

FP-growth

```
====min support:  0.05 ====
size of all frequent itemsets:  83
====min support:  0.1 ====
size of all frequent itemsets:  46
====min support:  0.2 ====
size of all frequent itemsets:  16
```

(Figure 7. Producing frequent itemsets of kaggle datasets by WEKA, Apriori and FP-growth)

- ➔ From figure 6 and 7, Apriori algorithm and FP-growth algorithm programmed here can get the same result as WEKA.

- Verifying rules.
  - Running on IBM datasets: our method vs. WEKA
    - min support=0.4, confidence=0.9

our method

```
rule: ['523'] -> ['3']  confidence: 0.9765395894428153
rule: ['3'] -> ['523']  confidence: 0.9794117647058823
```

WEKA

```
FPGrowth found 2 rules (displaying top 2)

1. [3=1]: 340 ==> [523=1]: 333   <conf:(0.98)>
2. [523=1]: 341 ==> [3=1]: 333   <conf:(0.98)>
```

(Figure8. rules of IBM datasets genererated by our methods and WEKA.)

  - Running on kaggle datasets: our method vs. WEKA
    - min support=0.1, confidence=0.9

our method

```
rule: ['COCK', 'COFFEE', 'CORNFLAKES'] -> ['BISCUIT']  confidence: 1.0
rule: ['COCK', 'CORNFLAKES'] -> ['BISCUIT', 'COFFEE']  confidence: 1.0
rule: ['BISCUIT', 'COFFEE', 'CORNFLAKES'] -> ['COCK']  confidence: 1.0
rule: ['BISCUIT', 'COFFEE'] -> ['CORNFLAKES', 'COCK']  confidence: 1.0
rule: ['BISCUIT', 'COCK', 'CORNFLAKES'] -> ['COFFEE']  confidence: 1.0
rule: ['BISCUIT', 'COCK'] -> ['CORNFLAKES', 'COFFEE']  confidence: 1.0
rule: ['BISCUIT', 'COCK', 'COFFEE'] -> ['CORNFLAKES']  confidence: 1.0
rule: ['BOURNVITA', 'TEA'] -> ['BREAD']  confidence: 1.0
rule: ['BISCUIT', 'TEA'] -> ['MAGGI']  confidence: 1.0
rule: ['BISCUIT', 'MAGGI'] -> ['TEA']  confidence: 1.0
rule: ['BISCUIT', 'MILK'] -> ['BREAD']  confidence: 1.0
rule: ['BISCUIT', 'COFFEE'] -> ['CORNFLAKES']  confidence: 1.0
rule: ['COCK', 'CORNFLAKES'] -> ['BISCUIT']  confidence: 1.0
rule: ['BISCUIT', 'COCK'] -> ['CORNFLAKES']  confidence: 1.0
rule: ['COCK', 'CORNFLAKES'] -> ['COFFEE']  confidence: 1.0
rule: ['JAM', 'MAGGI'] -> ['BREAD']  confidence: 1.0
rule: ['JAM'] -> ['BREAD', 'MAGGI']  confidence: 1.0
rule: ['BREAD', 'JAM'] -> ['MAGGI']  confidence: 1.0
rule: ['BISCUIT', 'COFFEE'] -> ['COCK']  confidence: 1.0
rule: ['BISCUIT', 'COCK'] -> ['COFFEE']  confidence: 1.0
rule: ['JAM'] -> ['MAGGI']  confidence: 1.0
rule: ['JAM'] -> ['BREAD']  confidence: 1.0
```

WEKA

```
FPGrowth found 22 rules (displaying top 22)

 1. [JAM=1]: 2 ==> [BREAD=1]: 2   <conf:(1)> lift:(1.54) lev:(0.04) con
 2. [JAM=1]: 2 ==> [MAGGI=1]: 2   <conf:(1)> lift:(4) lev:(0.08) conv:(
 3. [TEA=1, BOURNVITA=1]: 2 ==> [BREAD=1]: 2   <conf:(1)> lift:(1.54) l
 4. [BISCUIT=1, MILK=1]: 2 ==> [BREAD=1]: 2   <conf:(1)> lift:(1.54) le
 5. [JAM=1]: 2 ==> [BREAD=1, MAGGI=1]: 2   <conf:(1)> lift:(6.67) lev:(
 6. [BREAD=1, JAM=1]: 2 ==> [MAGGI=1]: 2   <conf:(1)> lift:(4) lev:(0.0
 7. [MAGGI=1, JAM=1]: 2 ==> [BREAD=1]: 2   <conf:(1)> lift:(1.54) lev:(
 8. [COFFEE=1, BISCUIT=1]: 2 ==> [CORNFLAKES=1]: 2   <conf:(1)> lift:(3
 9. [COFFEE=1, BISCUIT=1]: 2 ==> [COCK=1]: 2   <conf:(1)> lift:(6.67) l
10. [BISCUIT=1, COCK=1]: 2 ==> [COFFEE=1]: 2   <conf:(1)> lift:(2.5) le
11. [CORNFLAKES=1, COCK=1]: 2 ==> [COFFEE=1]: 2   <conf:(1)> lift:(2.5)
12. [TEA=1, BISCUIT=1]: 2 ==> [MAGGI=1]: 2   <conf:(1)> lift:(4) lev:(0
13. [BISCUIT=1, MAGGI=1]: 2 ==> [TEA=1]: 2   <conf:(1)> lift:(2.86) lev
14. [BISCUIT=1, COCK=1]: 2 ==> [CORNFLAKES=1]: 2   <conf:(1)> lift:(3.3
15. [CORNFLAKES=1, COCK=1]: 2 ==> [BISCUIT=1]: 2   <conf:(1)> lift:(2.8
16. [COFFEE=1, BISCUIT=1]: 2 ==> [CORNFLAKES=1, COCK=1]: 2   <conf:(1)>
17. [COFFEE=1, BISCUIT=1, CORNFLAKES=1]: 2 ==> [COCK=1]: 2   <conf:(1)>
18. [BISCUIT=1, COCK=1]: 2 ==> [COFFEE=1, CORNFLAKES=1]: 2   <conf:(1)>
19. [COFFEE=1, BISCUIT=1, COCK=1]: 2 ==> [CORNFLAKES=1]: 2   <conf:(1)>
20. [CORNFLAKES=1, COCK=1]: 2 ==> [COFFEE=1, BISCUIT=1]: 2   <conf:(1)>
21. [COFFEE=1, CORNFLAKES=1, COCK=1]: 2 ==> [BISCUIT=1]: 2   <conf:(1)>
22. [BISCUIT=1, CORNFLAKES=1, COCK=1]: 2 ==> [COFFEE=1]: 2   <conf:(1)>
```

(Figure9. rules of kaggle datasets genererated by our methods and WEKA.)

- ➔ From figure 8 and 9, the rules generated by our method and WEKA are the same.

## Summary

- In the Apriori algorithm, the number of frequent itemsets grows at first and converges in the end.
  - For example, 5 1-items will generate 10 2-items, so the number of frequent itemsets grows fast at first. After pruning out the un-frequent itemsets, the number decreases, so the number of frequent itemsets converges in the end.
- Running time: Apriori > FP-growth.
  - Apriori needs more times to scan overall datasets.
  - Apriori needs candidate iemsets which are generated as more as possible.
  - FP-growth records all transactions into a tree, and the frequent itemsets can therefore be found quickly by looking up the path of tree nodes.