Report 2: Discovery of Frequent Itemsets and Association Rules

Group 34: Sofia Krylova, Iosif Koen

November 21, 2022

1 Introduction

In a previous home assignment, we dealt with examining data for "similar" items. However, the problem of finding frequent itemsets differs from the similarity problem.

Here we are interested in the absolute number of baskets that contain a particular set of items, while previously we wanted items that have a large fraction of their baskets in common, even if the absolute number of baskets is small. The difference leads to a new class of algorithms for finding frequent itemsets.

Here comes the A-Priori Algorithm, which works by eliminating most large sets as candidates by looking first at smaller sets and recognizing that a large set cannot be frequent unless all its subsets are.

This, the aim of this home assignment is to to implement the A-Priori algorithm for finding frequent itemsets with support at least s in a dataset of sales transactions. Support of an itemset is the number of transactions containing the itemset.

We did not use any dedicated data frameworks. The task is implemented in Python.

2 Data source

As an incoming data, we used a sale transaction dataset that was provided for this assignment.

3 Solutions and Evaluation

3.1 A-Priory

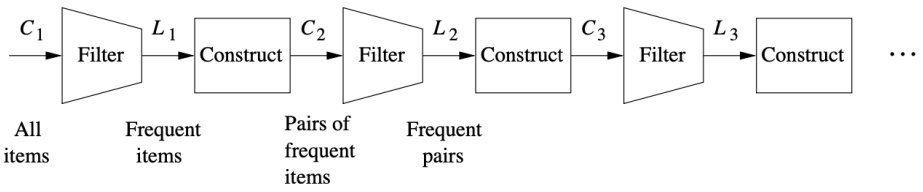
The A-Priori Algorithm is designed to reduce the number of pairs that must be counted, at the expense of performing two passes over data, rather

than one pass. It is used for finding frequent pairs without counting all pairs, which lets us find larger frequent itemsets without an exhaustive count of all sets.

Thus, one pass is taken for each set-size k. If no frequent itemsets of a certain size are found, then there can be no larger frequent itemsets, and the algorithm stops.

The pattern of moving from one size k to the next size $k + 1$ can be summarized as follows. For each size k, there are two sets of itemsets:

1. Ck is the set of candidate itemsets of size k – the itemsets that we must count in order to determine whether they are in fact frequent.
2. Lk is the set of truly frequent itemsets of size k.



This algorithm is implemented in the Apriory.py file. Keys are sorted for a better key comparison. For the *support* value *1500*, the results are the following:

```
Items  232: 351 -> 1641
Items  233: 405 -> 1525
Items  234: 893 -> 1947
Items  235: 85 -> 1555
Items  236: 450 -> 2082
Items  237: 769 -> 1622
```

Time, that was spent on this algorithm:

```
Spent time 3.72s.
```

Smaller the *support* value, longer it will take to find frequent items.
Support = 1000

```
Items  6: ('368', '829') -> 1194
Items  7: ('217', '346') -> 1336
Items  8: ('39', '704') -> 1107
Items  9: ('704', '825') -> 1102
Items  1: ('39', '704', '825') -> 1035

Spent time 5.101s.
```

Support = 800

```
Items  44: ('923', '947') -> 859
Items  1: ('227', '390', '722') -> 907
Items  2: ('217', '283', '346') -> 827
Items  3: ('217', '33', '346') -> 802
Items  4: ('217', '346', '515') -> 809
Items  5: ('283', '33', '346') -> 802
Items  6: ('283', '346', '515') -> 806
Items  7: ('39', '704', '825') -> 1035

Spent time 6.1s.
```

Support = 200

```
Items  63: ('108', '438', '486', '684', '71', '766', '8', '958') -> 200
Items  64: ('108', '486', '684', '71', '75', '766', '8', '958') -> 200
Items  65: ('242', '438', '486', '684', '71', '75', '766', '8') -> 200
Items  66: ('242', '438', '486', '684', '71', '75', '766', '958') -> 202
Items  67: ('438', '486', '684', '71', '75', '766', '8', '958') -> 200
Items  68: ('239', '278', '520', '72', '775', '822', '832', '970') -> 218
Items  1: ('1', '314', '470', '523', '66', '823', '874', '884', '980') -> 247

Spent time 153.543s.
```

3.2 Extra effort: Association rules

Now, when we have sets of frequent items from the previous step, we can generate association rules with confidence at least c.

An association rule is an implication $X \to Y$, where X and Y are itemsets such that

$$X \cap Y = \emptyset. \tag{1}$$

Support of rule $X \to Y$ is the number of transactions that contain

$$X \cup Y. \tag{2}$$

Confidence of rule $X \to Y$ is the following:

$$\frac{X \cup Y}{X}. \tag{3}$$

Number of rules also depends on the *support* and confidence c values: more frequents items we found on the previous step, more rules with confidence c we can create during this step.

Support = 1000, c = 0.5

```
Rule 1: (('39', '704'), '825') -> 0.935
Rule 2: (('39', '825'), '704') -> 0.872
Rule 3: (('704', '825'), '39') -> 0.939
Rule 4: ('227', '390') -> 0.577
Rule 5: ('704', '39') -> 0.617
Rule 6: ('704', '825') -> 0.614
```

Support = 800, c = 0.5

```
Rule 1: (('227', '390'), '722') -> 0.865
Rule 2: (('227', '722'), '390') -> 0.912
Rule 3: (('390', '722'), '227') -> 0.87
Rule 4: (('217', '283'), '346') -> 0.893
Rule 5: (('217', '346'), '283') -> 0.619
Rule 6: (('283', '346'), '217') -> 0.909
```

4 How to run

To run, you can access *test.py* class in our project. It only contains a main method, that calls methods for generating frequent itemsets and association rules. You can set the *support* and confidence c values in the main method. You would need Python 3+ (we used 3.9) to start a program.

```
if __name__ == '__main__':
    support = 200
    confidence = 0.5
    apriory = APriory(support)
    frequent_items = []

    start = time.time()
    # Created transactions and supported singletons
    ln = APriory.process_file(apriory)

    # Run until no supported sets are left
    n = 2
    while len(ln) > 0:
        k = 0
        for item in ln:
            k += 1
            print("Items  " + str(k) + ": " + str(item) + " -> "
                      + str(ln.get(item)))
        frequent_items.append(ln)
        ln = APriory.generate_ln_sets(apriory, ln, n)
        n += 1
```

```
end = time.time() - start
print("Spent time " + str(round(end, 3)) + "s.")

rules = Associations.get_rules(Associations(), frequent_items, confidence)
k = 0
for rule in rules:
    k += 1
    print("Rule " + str(k) + ": " + str(rule) + " -> "
               + str(rules.get(rule)))
```

5 Conclusions

In this homework assignment, we learned the A-Priory algorithm, how to find frequent items and generate association rules.

The accuracy and time of this algorithm depend on the initial *support* and confidence c values we set.