Roll No = AA.SC.U3BCA2207138
Name: Dharmabhushan Dattajirao Powar

Assignment – 4

Given are the following five transactions on items {A, B, C, D, E}:

tid - items
1 - A,B
2 - A,B,D
3 - B,D,E
4 - B,C,D,E
5 - A,B,C

1. **Use the Apriori algorithm to compute all frequent itemsets, and their support, with minimum support 2. It is important that you clearly indicate the steps of the algorithm.**

To compute frequent itemsets using the Apriori algorithm, we follow these steps:

Step 1: Generate frequent itemsets of length 1 (singleton itemsets) with support ≥ minimum support.
Step 2: Generate candidate itemsets of length k+1 from frequent itemsets of length k.
Step 3: Prune candidate itemsets that contain subsets of length k with support < minimum support.
Step 4: Repeat steps 2 and 3 until no new frequent itemsets can be generated.
Let's proceed with these steps:

Step 1: Generate frequent itemsets of length 1

Count occurrences of each item:

A: 3
B: 5
C: 2
D: 3
E: 2

Frequent itemsets of length 1:

{A} (Support = 3)
{B} (Support = 5)
{C} (Support = 2)
{D} (Support = 3)
{E} (Support = 2)

Step 2: Generate candidate itemsets of length 2 from frequent itemsets of length 1

We'll generate all possible combinations of length 2 from the frequent itemsets of length 1: {A,B}, {A,C}, {A,D}, {A,E}, {B,C}, {B,D}, {B,E}, {C,D}, {C,E}, {D,E}.

Step 3: Prune candidate itemsets

We'll prune those combinations that contain subsets not present in frequent itemsets of length 1:

- {A,C}, {A,D}, {A,E}, {C,D}, {C,E}, {D,E} are pruned because they contain subsets with support < 2.

Step 4: Repeat Steps 2 and 3

We've pruned all itemsets of length 2, so we can stop here.
Frequent itemsets with minimum support of 2:

- {A} (Support = 3)
- {B} (Support = 5)
- {C} (Support = 2)
- {D} (Support = 3)
- {E} (Support = 2)
- {A, B} (Support = 3)
- {B, D} (Support = 3)
- {B, E} (Support = 2)
- {C, D} (Support = 2)
- {D, E} (Support = 2)

These are the frequent itemsets with support ≥ 2 using the Apriori algorithm.

2. Use the A-close algorithm to compute all closed frequent itemsets, with minimum support 2. It is important that you clearly indicate the steps of the algorithm.

To compute closed frequent itemsets using the A-close algorithm, we follow these steps:

Step 1: Generate frequent itemsets using the Apriori algorithm with minimum support.
Step 2: Traverse the lattice of frequent itemsets and identify closed frequent itemsets.
Let's proceed with these steps:

Step 1: Generate frequent itemsets using the Apriori algorithm with minimum support 2:
We already have the frequent itemsets computed from the Apriori algorithm:

- {A} (Support = 3)
- {B} (Support = 5)
- {C} (Support = 2)
- {D} (Support = 3)
- {E} (Support = 2)
- {A, B} (Support = 3)
- {B, D} (Support = 3)
- {B, E} (Support = 2)
- {C, D} (Support = 2)
- {D, E} (Support = 2)

Step 2: Traverse the lattice and identify closed frequent itemsets:
For each frequent itemset X, check if there exists no frequent itemset Y such that Y is a superset of X and has the same support as X. If such Y exists, then X is not closed. Otherwise, X is a closed frequent itemset.

- {A} is closed because there's no superset with the same support.
- {B} is closed because there's no superset with the same support.
- {C} is closed because there's no superset with the same support.
- {D} is not closed because {B, D} is also frequent with the same support.

- {E} is closed because there's no superset with the same support.
- {A, B} is not closed because {B} is also frequent with the same support.
- {B, D} is closed because there's no superset with the same support.
- {B, E} is closed because there's no superset with the same support.
- {C, D} is not closed because {B, D} is also frequent with the same support.
- {D, E} is closed because there's no superset with the same support.

Closed frequent itemsets with minimum support of 2:

- {A}
- {B}
- {C}
- {E}
- {B, D}
- {B, E}
- {D, E}

These are the closed frequent itemsets with support ≥ 2 using the A-close algorithm.

3. Use all closed frequent itemsets computed at (b) to construct a krimp codetable, and compute how often each itemset in the codetable is used in covering the database. Also compute the optimal code length for each itemset in the codetable. Note: Here you should use logarithms with base 2.

To construct a KRIMP (Krimp is a data compression algorithm) codetable using the closed frequent itemsets and compute the usage frequency and optimal code length for each itemset, we need to follow these steps:

1. Create a codetable using closed frequent itemsets.**
2. Compute the usage frequency of each itemset in covering the database.**
3. Compute the optimal code length for each itemset.**

Let's proceed with these steps:

Step 1: Create a codetable using closed frequent itemsets:**
The codetable is essentially a list of itemsets with their corresponding codes.

- {A} - Code: 0
- {B} - Code: 1
- {C} - Code: 2
- {E} - Code: 3
- {B, D} - Code: 4
- {B, E} - Code: 5
- {D, E} - Code: 6

Step 2: Compute the usage frequency of each itemset:**
We need to find out how many times each itemset appears in the database.

- {A} appears 3 times.
- {B} appears 5 times.
- {C} appears 2 times.
- {E} appears 2 times.
- {B, D} appears 3 times.

- {B, E} appears 2 times.
- {D, E} appears 2 times.

Step 3: Compute the optimal code length for each itemset:**
The optimal code length is given by `-log2(p)` where `p` is the probability of occurrence of the itemset.
For itemset {A}:

- Probability of occurrence = Frequency of {A} / Total transactions = 3 / 5 = 0.6
- Optimal code length = log2(0.6) ≈ 0.736 bits

Similarly, we compute the optimal code length for other itemsets.

- Optimal Code Lengths:**

  - {A}: 0.736 bits
  - {B}: 0.439 bits
  - {C}: 1 bit
  - {E}: 1 bit
  - {B, D}: 0.736 bits
  - {B, E}: 1 bit
  - {D, E}: 1 bit

These are the optimal code lengths for each itemset in the codetable. We can also compute the total code length by summing up the optimal code lengths multiplied by their frequencies and compare it to the original size of the database to assess compression efficiency.

4. Suppose that in addition to the transactions, we also have information about the price of each item. Consider constraints of the type

$$\max(I.price) - \min(I.price) \leq c$$

where max(I.price) denotes the maximum of the prices of the items in itemset I, and c is some positive constant. Suppose that we want to find all frequent itemsets that also satisfy a constraint of this type. Can we use this constraint in an Apriori style levelwise search to further prune the search space, while still guaranteeing completeness of the results? Explain your answer.

Yes, we can use constraints like max(I.price) − min(I.price) ≤ c in an Apriori-style levelwise search to further prune the search space while still guaranteeing completeness of the results.

Here's how:

1. Initial Pruning:
   - At the beginning of each level in the Apriori algorithm, we can compute the max and min prices of the items in the frequent itemsets generated in the previous level.
   - If the difference between the max and min prices of any frequent itemset exceeds the constraint (c), we can discard that frequent itemset since it cannot satisfy the constraint.

2. Subsequent Pruning:

    - As we generate candidate itemsets for the next level, we can utilize the information about max and min prices of the frequent itemsets from the previous level.
    - When generating candidate itemsets, we can prune those candidate itemsets that, when combined with any frequent itemset from the previous level, would violate the constraint.

By applying these pruning techniques, we can effectively reduce the search space while ensuring completeness of the results. This is because we are systematically eliminating itemsets that cannot satisfy the constraint, without missing any potential frequent itemsets that do satisfy the constraint.

However, it's important to note that the effectiveness of this approach depends on the distribution of prices and the chosen constraint (c). If the constraint is too restrictive or if the distribution of prices is such that many potential frequent itemsets are eliminated early on, it might lead to significant pruning of the search space but could potentially miss some infrequent itemsets that satisfy the constraint. On the other hand, if the constraint is too loose, it might not provide much pruning benefit. Therefore, it's crucial to choose an appropriate constraint based on the data characteristics and the desired level of pruning.