HW2 - Frequent Itemsets

Syed Arif Rahman<sarahman@kth.se>

Group 10

1. Introduction

The Apriori Algorithm is a classic algorithm in data mining and association rule learning. It is used to discover frequent itemsets in a dataset and generate association rules based on these itemsets. The assignment presented here implements the Apriori Algorithm in Java and is organized into several classes to facilitate modularity and readability.

2. Build and Run

The project follows a modular structure with each class serving a specific purpose in the Apriori algorithm workflow. To build and run the project, you would need to:

1. Set the desired support threshold (‘support’) and specify the path to the dataset (‘dataPath’) in the ‘main’ method of the ‘AprioriAlgorithm’ class.

2. Execute the ‘main’ method of the ‘AprioriAlgorithm’ class.

The dataset is read from the specified path, and the algorithm's execution is then reported step by step.

3. Solution

The assignment consists of six classes: ‘AprioriAlgorithm’, ‘AprioriAlgorithmPass’, ‘ConstructFItems’, ‘DoubleTonCounter’, ‘ConstructTripletons’, and ‘DoubleTonCounter’.

The assignment implements the Apriori Algorithm in a modular and organized manner. It follows the typical Apriori workflow, starting with the first pass, handling singleton and doubleton generation between passes, and finally constructing frequent tripletons.

1. Main Class (‘AprioriAlgorithm’):
 - The main class is responsible for orchestrating the entire Apriori Algorithm process.
 - It reads the dataset from a specified path, initializes the first pass of the algorithm, handles singleton and doubleton processing, and finally, constructs and prints frequent tripletons.

2. AprioriAlgorithmPass:
 - This class handles the first pass of the algorithm, creating forward and backward mappings, and counting the occurrences of each item in the dataset.

3. ConstructFItems:
 - It handles the generation of singleton and doubleton itemsets based on the support threshold.

4. DoubleTonCounter:
 - This class performs the second pass of the algorithm, counting occurrences of item pairs and creating a map for subsequent passes.

5. ConstructTripletons:
 - It constructs and filters frequent tripletons based on the support threshold.

6. DoubleTonMapper:
- A utility class providing a method to generate a map during the doubleton counter.

4. Results

The dataset we used to test our program is “T10I4D100K.dat”, which contains 100000 baskets and 870 individual items. Each item is a positive number. The baskets in the dataset are look like below-

```
Basket 1:{25,52,164,240,274,328,368,448,538,561,630,687,730,775,825,834}
Basket 2:{39,120,124,205,401,581,704,814,825,834}
Basket 3:{35,249,674,712,733,759,854,950}
.........
```

Our forward map and backward maps looks like below -

```
Forward Mapping:(index=item)
{0=0, 1=1, 2=2, 3=3, 4=4, 5=5, 6=6, 7=7, 8=8, 9=10, 10=11, 11=12, 12=13,...}
Backward Mapping:(item=index)
{0=0, 1=1, 2=2, 3=3, 4=4, 5=5, 6=6, 7=7, 8=8, 10=9, 11=10, 12=11, 13=12,...}
```

Total iteration of each item on whole BasketSet are looks like below -

```
Each Item iterate on basketset:
[594, 1535, 673, 531, 1394, 1094, 2149, 997, 3090, 1351, 525, 3415,... ]
```

We also measure times on milliseconds for every frequent Items. Below the output of time measuring for read whole basket, find out singleton, find out doubleton and find out tripleton:

```
[33848, 406, 2409, 447]
```

Doubleton took time because during find out doubleton we go through numbers of tasks. We also check with different support and got some exceptional results, output format be like - [Frequent ItemSet(SingleTon, Frequent Candidates(DoubleTon), Frequent ItemSet((DoubleTon), Frequent ItemSet((TripleTon):

```
Support:0.03

Output: [60, 1770, 0, 0]

Support:0.02

Output: [155, 11935, 0, 0]

Support:0.03

Output: [375, 70125, 9, 1]
```

From this output, we can say that only when support is below or equal to 0.01, we can achieve frequent item set consists of two or three items. Otherwise, only singleton frequent item set can be achieved.

5. Conclusion

The Apriori algorithm implemented in this Java assignment effectively identifies frequent itemsets within a given dataset. The modular design of the classes facilitates a clear and organized workflow, making the code easy to understand and maintain. The algorithm successfully handles singleton, doubleton, and tripleton itemsets, providing insights into frequent patterns present in the dataset.

The choice of a support threshold allows users to customize the level of significance for identifying frequent itemsets. The project demonstrates a successful application of the Apriori algorithm for mining association rules in large datasets, providing a foundation for further optimization and extension for more complex scenarios.