Programming Project: Frequent Itemset Mining

Chunlei Zhou

6.7 Using Python implement frequent itemset mining algorithms. Compare the performance of each algorithm with UCI Adult Census Dataset.
Write a report to analyze the situations where one algorithm may perform better than the others, and state why.

The dataset contains more than 30,000 transactions with each transaction including 15 items. The following three algorithms are used in turn to mine the frequent itemset patterns of the UCI Adult Census Dataset.
I first chose 50% as the minimum support threshold. I would like to ensure that I identified all the possible frequent itemsets. However, if threshold is not high enough, there might be some useless rules in the set of patterns returned by the algorithms. I consider 50% can trade off properly between the size of frequent itemsets and the usefulness of the patterns returned. Thus, I chose 50% as the threshold.

(1) Apriori [AS94b]

The frequent patterns returned by Apriori algorithm is as follow:

[[{' White'}, {' Male'}, {'0'}, {' United-States'}, {' <=50K'}, {' Private'}],
[{' White', ' Male'}, {' White', '0'}, {' White', ' United-States'}, {' White', ' <=50K'}, {' White', ' Private'}, {' Male', '0'}, {' United-States', ' Male'}, {' United-States', '0'}, {' <=50K', '0'}, {' Private', '0'}, {' United-States', ' <=50K'}, {' United-States', ' Private'}, {' Private', ' <=50K'}],
[{' White', ' Male', '0'}, {' White', ' United-States', ' Male'}, {' White', ' United-States', '0'}, {' White', ' <=50K', '0'}, {' White', ' Private', '0'}, {' White', ' United-States', ' <=50K'}, {' White', ' United-States', ' Private'}, {' United-States', ' Male', '0'}, {' United-States', ' <=50K', '0'}, {' United-States', ' Private', '0'}, {' <=50K', ' Private', '0'}],
[{' White', ' Male', ' United-States', '0'}, {' White', ' <=50K', ' United-States', '0'}, {' White', ' Private', ' United-States', '0'}]]

The executing time of Apriori algorithm is 0.47719788551330566 seconds.

This algorithm performs best with minimum support equals to 50%. The returned patterns are sorted by the k (frequent k-itemset). The largest k is 4. The execution time, for now, of this algorithm is shorter than FP-Growth. However, if the minimum support threshold is lower, for example, 20%, the FP-Growth will use less time than Apriori. This is because with 50% minimum support threshold, the number of frequent patterns is relatively small, which make the Apriori algorithm less time consuming. However, FP-Growth algorithm still needs to build the FP-tree from top to bottom and identify frequent patterns from bottom to top. Thus, the latter is more time consuming than the former when the minimum support threshold is high.

(2) FP-growth [HPY00]

The frequent patterns returned by FP-growth algorithm is as follow:

[({' Male'}, 21775), ({' White', ' Male'}, 19162), ({' United-States', ' White', ' Male'}, 17644), ({' United-States', ' White', '0', ' Male'}, 17644), ({' White', '0', ' Male'}, 19162), ({' United-States', ' Male'}, 19477), ({' United-States', '0', ' Male'}, 19477), ({'0', ' Male'}, 21775), ({' Private'}, 22673), ({' Private', ' <=50K'}, 17712), ({' Private', ' <=50K', '0'}, 17712), ({' Private', ' White'}, 19384), ({' Private', ' White', ' United-States'}, 17714), ({' Private', ' White', ' United-States', '0'}, 17714), ({' Private', ' White', '0'}, 19384), ({' Private', ' United-States'}, 20119), ({' Private', '0', ' United-States'}, 20119), ({' Private', '0'}, 22673), ({' <=50K'}, 24698), ({' <=50K', ' White'}, 20680), ({' United-States', ' <=50K', ' White'}, 18904), ({' United-States', ' <=50K', ' White', '0'}, 18904), ({' <=50K', ' White', '0'}, 20680), ({' United-States', ' <=50K'}, 21984), ({' United-States', ' <=50K', '0'}, 21984), ({' <=50K', '0'}, 24698), ({' White'}, 27795), ({' United-States', ' White'}, 25606), ({' United-States', ' White', '0'}, 25606), ({' White', '0'}, 27795), ({' United-States'}, 29153), ({' United-States', '0'}, 29153), ({'0'}, 32537)]

The executing time of the FP-Growth algorithm is 7.8134400844573975 seconds.

This algorithm uses longest time to return all frequent patterns with minimum support equals to 50%. The number of frequent itemsets returned is almost the same with that returned by Apriori algorithm. FP-Grwoth algorithm and Apriori algorithm return similar frequent itemsets. The returned patterns are followed by their support count.

(3) Improvement of Apriori (Partition Algorithm)

The frequent patterns returned by the improved Apriori algorithm is as follow:

["{' White'}", "{' Male'}", "{'0'}", "{' United-States'}", "{' <=50K'}", "{' Private'}", "{' White', '0'}", "{' United-States', ' White'}", "{' Male', '0'}", "{' Male', ' United-States'}", "{' United-States', '0'}", "{' <=50K', '0'}", "{'0', ' Private'}", "{' <=50K', ' United-States'}", "{' United-States', ' Private'}", "{' <=50K', ' Private'}", "{' United-States', ' White', '0'}", "{' Male', ' United-States', '0'}", "{' <=50K', ' United-States', '0'}", "{' United-States', '0', ' Private'}", "{' <=50K', '0', ' Private'}", "{' Male', ' White', ' United-States', '0'}", "{' White', ' <=50K', ' United-States', '0'}"]

The executing time of Phase I is 1.9493610858917236 seconds.
The executing time of merge phase is 1.949631929397583 seconds.
The executing time of Phase II is 3.3824050426483154 seconds.
The executing time of the improved Apriori algorithm is 3.382446050643921 seconds.

The number of frequent itemsets returned by this algorithm is 10 less than that returned by the Apriori algorithm. This algorithm should use less time than the Apriori algorithm, but it takes even longer time than the unimproved version.

The partition algorithm is consisting of two phases. The first phase is to find all frequent patterns in each partition of the dataset and merge them to one frequent itemset list. The second phase was to return all itemsets in the frequent itemset list that have support count more than the minimum support threshold.
The partition size is calculated as follow:

```
The size of page in main memory is 4096 bytes.
The size of the dataset it 4.3MB.
Thus, the number of partitions should be at least 1043 and the partition size should
be at most 31.
```

Phase I was not short enough is because that for each partition, I call functions defined in Apriori algorithm. The dataset is not large enough, thus, it can be loaded to the main memory, though it cannot be loaded in one page. Thus, load partitions in one page one by one in turn may not improve the running time significantly. For the merge phase, the algorithm scans the identified frequent patterns for each partition one by one. Since there are a lot of partitions, the process is time consuming as well. The most time-consuming part is phase II. I used too many for loops to measure the frequency of each identified patterns. It seems to me that there is no better way than using for loops to return the final result.
One reason for the partition algorithm performing worse than the original algorithm is because that loading partitions to page cannot significantly improve the running time of Apriori algorithm, since the dataset can be loaded in main memory. If the dataset can be larger so Apriori algorithm has to read data from disk, than the partition algorithm will significantly improve the performance of Apriori algorithm. Another reason is that partition dataset causes phase II using too many for loops, which increases the time complexity of the algorithm.