CS 470

Data Mining

Homework 2

Jason Ji

Collaborations: Discussed the idea for efficient approach with Richard Yang

Runtime Analysis

Dataset	T10I4D100K.txt						
Total number of transactions	100,000						
Minimum Support	500	700	900	1100	1300	1500	1700
Number of frequent itemsets	1073	603	421	357	289	237	193
Execution Time (second)	6.58	2.95	1.81	1.56	1.34	1.11	0.98

Dataset	mushroom.txt						
Total number of transactions	3698						
Minimum Support	2000	2200	2400	2600	2800	3000	3200
Number of frequent itemsets	2367	1087	527	223	95	63	15
Execution Time (second)	7.48	6.23	2.29	0.7	0.29	0.2	0.14

Analysis: from the data collected from two datasets we can see that as the minimum support increases, the total number of frequent items obtained by the apriori algorithm decreases, and the total execution time of the program decreases. There is an inverse relationship between minimum support and the number of frequent itemsets, also between minimum support and the execution time. This makes sense because a higher minimum support means more items are filtered out as algorithm runs, making the algorithm faster. In addition, there are less qualified itemsets at the end.

Optimization Method

When I first started the assignment, I tried to implement the algorithmm according to the psuedo code provided in our textbook. However, after completing it and testing it on the datasets provided, I realized that it works for dataset that has less total transactions such as mushroom.txt, but it takes too long to run on dataset that has a large total transaction count, such as T10I4D100K.txt.

After a lot of considerations and discussions, I realized that it is because I was traversing through the entire database (every transaction) to check for the support count of each potential itemset. This wouldn't work for dataset with large transaction count. Therefore I modified my algorithm. Instead of following the

psuedo code provided by the book that requires looping through all transactions multiple times, I generate the candidate frequent-k-itemsets from each transaction for each k, and remove infrequent-k-items from each transactions. I can do this because those infrequent-k-itemsets don't matter anymore as the frequent-(k+1)-itemsets won't contain them. In this way, I am able to reduce total size of each transaction, and reduce the number of traversal through all transactions for each k. This drastically speeded up the process.

Experience and Lessons Learned

I realized the importance of data structure design when doing data mining works. Because of the size of the input data, it is critical that we carefully plan out the ideal structure before implementation. In my case, I didn't think about it too much and just followed the directions provided by the textbook. Although I was able to get the correct results for smaller dataset, it is not efficient enough for larger dataset. As a result, I had to redesign my approach and code a different algorithm that looks a lot different from what I initially had. This lead to a lot of redundant work and waste of time.

In addition, during the optimization process I noticed that I was putting a loop inside another loop, but the inner loop is not dependent on the outer loop and that was making my program much slower. It sheds lights on how we need to be extremely careful on writing loops when doing data mining. A careless mistake like that may not matter for other programs that deal with smaller input, but becomes extremely problematic when process large data.