East West University

Course title: Data Mining
Course code: CSE477
Summer 2023
Section: 1

LAB Report - 1:

" Efficiency Analysis of Apriori Algorithm Using Two Datasets"

Submitted To:

Jesan Ahmed Ovi
Senior Lecturer,
Department of Computer Science and Engineering
East West University, Dhaka

Submitted By:

K. M. Safin Kamal
ID: 2020-1-60-235
Department of Computer Science and Engineering
East West University, Dhaka

Date of Submission: 01/ 08 / 2023

Apriori algorithm:

Apriori is a popular algorithm for frequent itemset mining in association rule mining. It scans a dataset multiple times to identify frequent itemsets, sets of items that appear together frequently. It uses a breadth-first search approach to efficiently generate candidates for frequent itemsets and prune infrequent ones. The algorithm employs the "Downward Clouser property," which states that any non-frequent itemset's supersets must also be infrequent. By iteratively increasing the itemset size, Apriori progressively identifies the most frequent itemsets. These frequent itemsets are then used to generate association rules, which highlight meaningful relationships between items in the dataset, aiding in market basket analysis and recommendation systems.

Dataset Description:

Two datasets, namely **chess** and **mushroom**, were utilized in the study. The chess dataset comprises **3196** transactions, each involving **75** items. In contrast, the mushroom dataset contains **8124** transactions with **119** items. The sizes of the chess and mushroom datasets are approximately 335 KB and 558 KB, respectively.

Implementation:

The Apriori algorithm is implemented in python language. Pycharm software is used for this. The configuration of the computer is Intel i7 13th 13700k (5.40 GHz) CPU, 32GB DDR5 (5200MHz) RAM, RTX 3060ti GDDR6 GPU and Windows 11 pro operating system.

Result Analysis:

Here we are going to analyze the results of the Apriori algorithm on two datasets. First, we will analyze on Chess dataset and then Mushroom dataset.

Dataset: Chess

The bar chart in figure 1 shows the number of frequent patterns for different values of "L" with a threshold of **60%,** where "L" represents the length of the itemsets. Each bar corresponds to a specific value of "L," and the height of the bar represents the number of frequent patterns found for that length. For threshold 60%, we have got total **14** number of L and total **254944** frequent patterns.

Here in figure-1, **L7** has the highest number of frequent patterns (about **57479**) compared to other L values. This means that the dataset contains a significant number of frequent patterns with a length of 7. As we move away from L7 in both directions (to the left and right), the number of frequent patterns decreases. This indicates that the dataset has fewer frequent patterns as we consider shorter patterns (L1, L2, L3, etc.) or longer patterns (L8, L9, L10, etc.) beyond L7. This suggests that the dataset contains an interesting range of frequent patterns with varying lengths, but the highest concentration is at L7. **L14** has the lowest number of frequent patterns, which is only **8**. The bar chart represents almost a symmetrical shape.

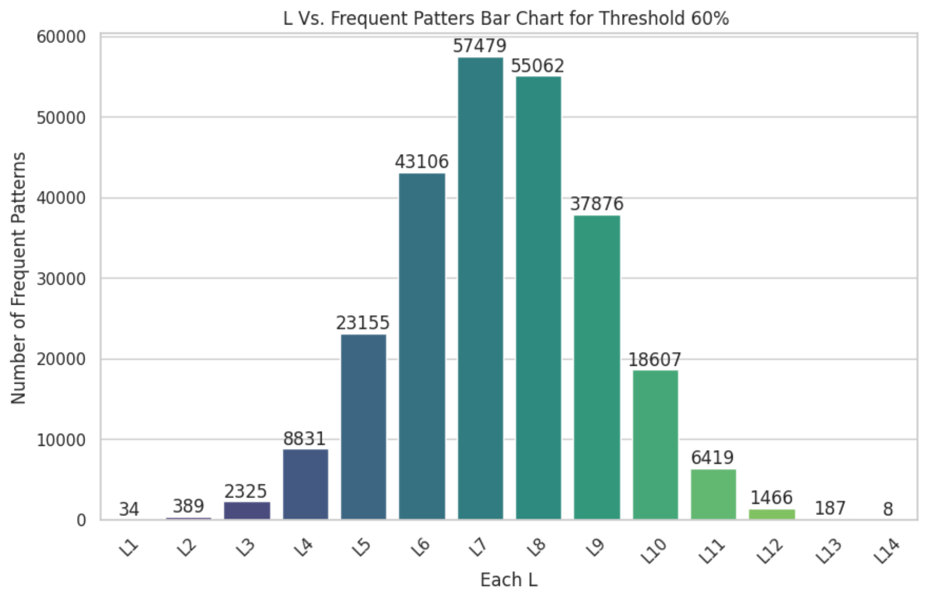

Figure 1: L Vs. Frequent Patters Bar Chart for Threshold 60% on Chess Dataset

The line plot in figure-2 displays the time taken in seconds (y-axis) for different threshold values (x-axis). Each point on the line represents the time taken for a specific threshold. The time taken to process the data decreases as the threshold increases. As we raise the minimum support threshold, the number of frequent patterns to be considered decreases, leading to faster processing. The line shows a decreasing trend, indicating that higher thresholds result in faster computation times. The steepest decline in time occurs between the threshold values of **"60%"** and **"80%".** This suggests that the initial increase in the threshold has a significant impact on reducing the time needed for the analysis. The time approaches zero as the threshold approaches **"90%"**. This suggests that with a very high threshold, the algorithm quickly identifies only a few frequent patterns, resulting in minimal computation time.

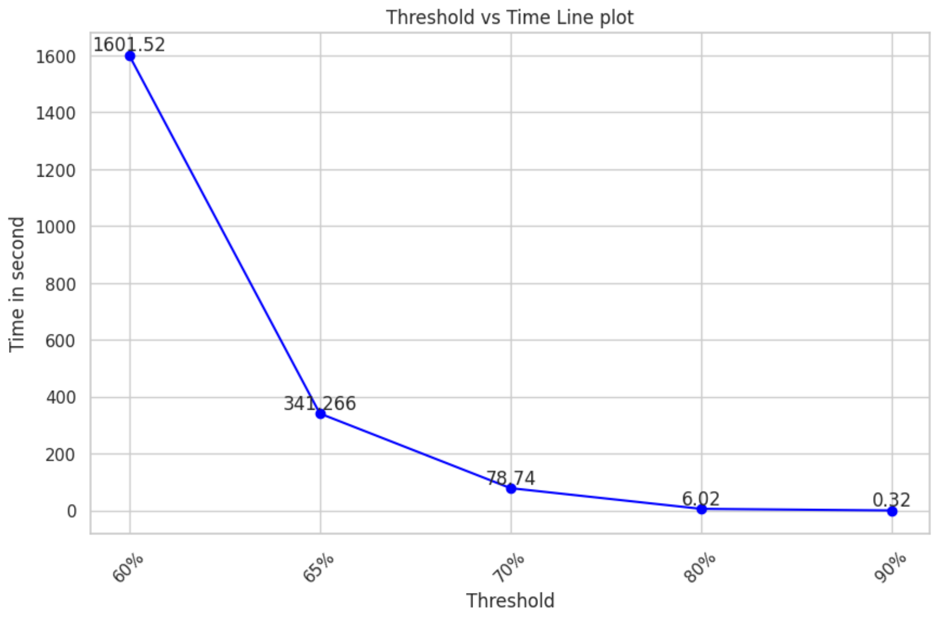

Figure 2: Threshold vs Time Line plot on Chess Dataset

Dataset: Mushroom

Similarly, like Chess dataset, we are also plotting the bar chart and line plot in Mushroom data.

We have made a bar chart on threshold **60%**. In this threshold, we have got total **5** number of L and total **51** frequent patterns.

From the bar chart in figure-3, on threshold 60%, the number of frequent patterns decreases as we move from shorter patterns (L1, L2, L3) to longer patterns (L4, L5). This means that the dataset contains more frequent patterns with shorter lengths compared to longer patterns. The bar for **L2** (length-2 patterns) is the highest (about **18**), indicating that there are more frequent single-item patterns compared to other lengths. The bar for L3 is also relatively high, suggesting that frequent patterns of length 3 are quite common in the dataset. As we move to L4 and L5**,** the number of frequent patterns drops significantly, implying that patterns of length 4 and 5 are less frequent and less common in the dataset. So, it illustrates that shorter pattern (L1, L2, L3) are more abundant in the dataset, while longer patterns (L4, L5) are less frequent.

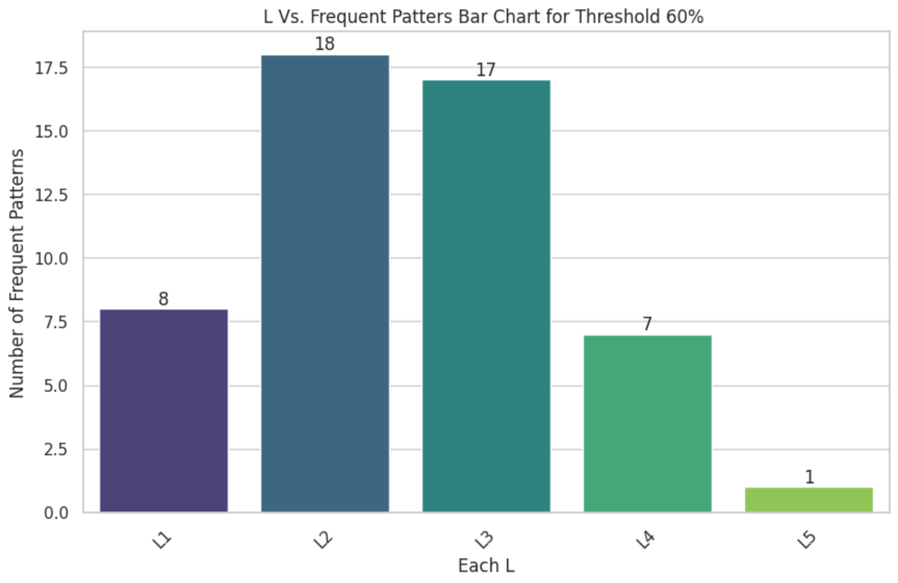

Figure 3: L Vs. Frequent Patters Bar Chart for Threshold 60% on Mushroom Dataset

In figure-4, we are showing a line plot where each point on the line represents the time taken for a specific threshold. The time taken to process the data decreases as the threshold increases. As we raise the minimum support threshold, the number of frequent patterns to be considered decreases, leading to faster processing. The line shows a decreasing trend, indicating that higher thresholds result in faster computation times. The steepest decline in time occurs between the threshold values of **"70%"** and **"80%"**. This suggests that the initial increase in the threshold has a significant impact on reducing the time needed for the analysis. The time approaches zero as the threshold approaches **"90%"**. This suggests that with a very high threshold, the algorithm quickly identifies only a few frequent patterns, resulting in minimal computation time. It illustrates that increasing the threshold leads to faster computation times, and choosing an appropriate threshold can significantly impact the efficiency of the algorithm.

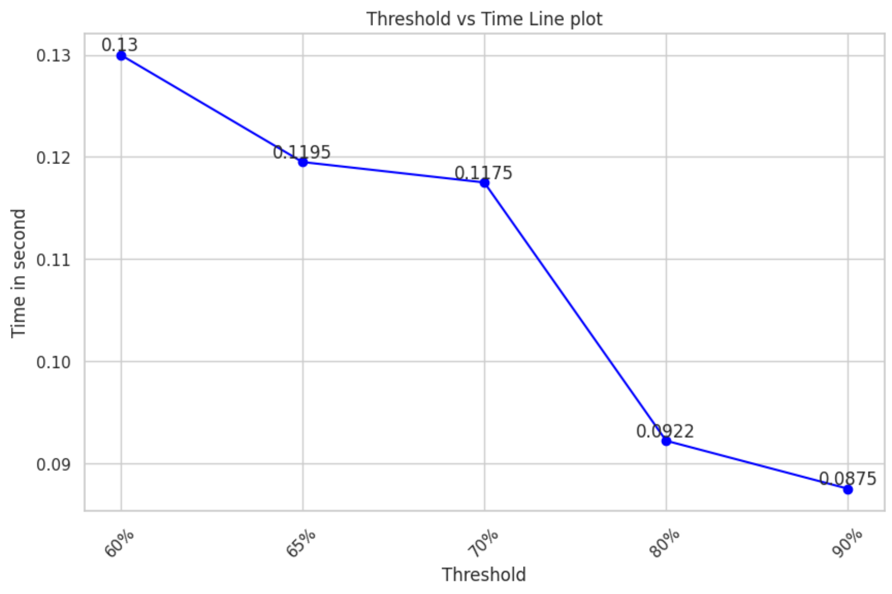

Figure 4: Threshold vs Time Line plot on Chess Dataset

Discussion:

We can see that the chess dataset has lower transactions and items than mushroom dataset. But from the bar charts and line plots of the two datasets, the chess dataset generates more frequent patters than mushroom dataset. The table 1 is displaying the same thing also.

Table 1: Summary of each dataset.

Threshold (%)	**Chess Dataset**		**Mushroom Dataset**	
	Total Frequent patterns	**Time (second)**	**Total Frequent patterns**	**Time (second)**
60%	254944	1601.52	51	0.13
65%	111239	341.266	39	0.1195
70%	48731	78.74	31	0.1175
80%	8227	6.02	23	0.0922
90%	622	0.32	9	0.0875

As the chess dataset generates more frequents patterns, it takes a lot computational time. From the table we see that at 60% threshold there is almost **5000** times difference in the total number of Frequent patterns. As a result, there is also a huge time difference.

So, we can say that despite the differences in the number of frequent patterns and computational time, both line charts demonstrate the efficiency of the Apriori algorithm. As the minimum support threshold increases, the algorithm consistently shows reduced execution time, highlighting its scalability and efficiency in mining frequent patterns. In summary, the implemented Apriori algorithm proves to be an effective approach for extracting valuable insights from datasets, despite the limitation of scanning the dataset multiple times.