EAST WEST UNIVERSITY

CSE477: DATA MINING

SECTION: 1

PROJECT REPORT

SPRING 2021

SUBMITTED TO	SUBMITTED BY
Jesan Ahammed Ovi	Kafi Khan
Lecturer	ID: 2017-2-60-078
Department of CSE	Md. Shoumique Hasan
East West University	ID: 2017-2-60-152
	Nilofa Yeasmin
	ID: 2017-2-60-151
	Md. Ariful Islam Fahad
	ID: 2017-2-60-032
	Md. Mushfiqul Alam
	ID: 2016-1-60-040

DATE OF SUBMISSION

03 JUNE, 2021

We have two datasets, namely, Chess and Kosarak dataset. In this project, we have to compare the time required to generate frequent patterns from both of the datasets with both the Apriori and FP-Growth algorithms.

Methodology

We have used FP-Growth and Apriori module from mlextend library in Python. Furthermore, We have used transactionencoder module from mlextend to preprocess the datasets and Seaborn and Matplotlib to generate graphs. However, we have encountered an obstacle while trying to mine Kosarak dataset. This is because, after preprocessing the dataset with transactionencoder the dataset turns out to be 34 GB, which makes it improbable and inefficient to keep in the entire dataset in memory. As a result, we had to apply multiple methods to overcome this obstacle.

Method 1: We used Sparse dataframe from Python's Pandas library to overcome the aforementioned obstacle. Sparse dataframes do not hold any values in place of missing values and Pandas dataframe further treats those values as Boolean False, which in turn reduces the size of the dataset drastically. Upon using this dataframe, we could process the entire Kosarak dataset at once.

Method 2: In this method, we have pruned all the items that do not meet a certain support threshold. For our project, we have selected the support count as 5%. In doing so, most of the items get pruned right off the bat, as a result, there are fewer items for the transactionencoder to encode resulting in a lower number of columns in the dataframe. For example, without any preprocessing, the Kosarak dataset is about 990,002 * 35,000 in dimension after encoding. But, after pruning, the dataset is about 990,002 * 4 in dimension. However, it imposes a preprocessing overhead.

Furthermore, we have performed both of the algorithms on Chess dataset from the threshold of 65% to 95% and Kosarak dataset from the threshold of 50% to 95% incremented by 5% on each iteration. Moreover, we have performed each algorithm 10 times on each iteration to get a clearer picture of the threshold and reported the data in Box Plots.

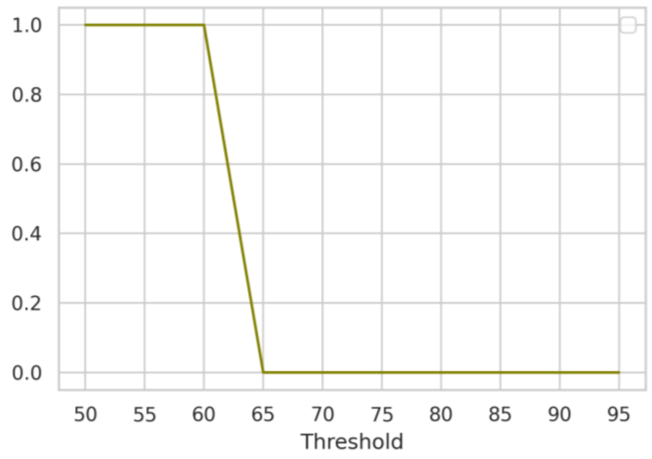
Datasets

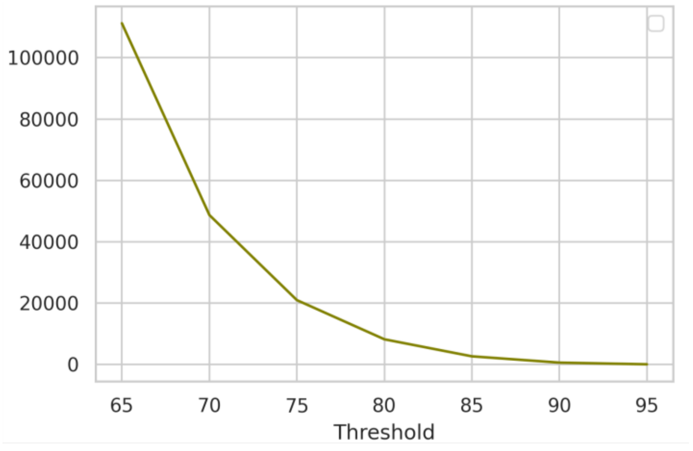
Chess Dataset: The chess dataset consists of 3196 transactions. Each transaction holding multiple values from 1 to 75. Initially, the dataset occupied around 335KB of memory. After using transactionecoder, the dataframe was (3196 * 75) in dimension and occupied around 235KB. Furthermore, after pruning infrequent values, the size of the dataframe was 207KB having 66 columns instead of 75.

Kosarak Dataset: Kosarak Dataset consists of 990,002 transactions. Each transaction holding multiple values from 1 to 35,000. The raw data occupied around 32MB of memory. After using transactionencoder, the dataframe was (990,002 * 35,000) in dimension and occupied around 34GB of memory. However, after using sparse dataframe the size of the entire dataframe was around 34MB. Lastly, after pruning infrequent values, the size of the dataset was 4MB having only 4 columns instead of 35,000.

Result Analysis

Upon performing the algorithms on both of the datasets, the number of frequent patterns can be seen in Figures 1 and 2.





Figures 1 and 2: Threshold vs Number of Frequent Patterns for Chess and Kosarak dataset respectively.

The difference between both of the datasets is quite apparent from the figure. While the Chess dataset contains considerably fewer instances of data than the Kosarak dataset, the Chess dataset contains substantially more frequent patterns than the Kosarak dataset. Later on, this discrepancy plays a huge role in the evaluation of the aforementioned algorithms.

Upon applying method 1 with both of the algorithms on Chess dataset, Fp-growth performs considerably well in terms of performance when compared to Apriori. This is to be expected, as Fp-Growth is a far more optimized and sophisticated algorithm. Figure 3 depicts a comparison between both of the algorithms using box plots.

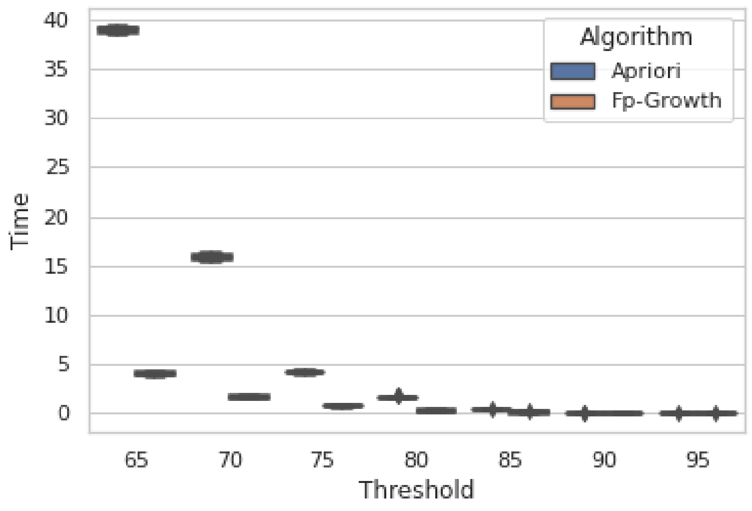


Figure 3: Comparison between algorithms on Chess dataset.

In Figure 3, for each threshold, the first boxplot is the time of Apriori and the second one is the time of Fp-Growth. As the threshold got larger and larger the difference between the performance of both of the algorithms becomes smaller and smaller.

Both algorithms perform unexpectedly while applied on the Kosarak dataset. Fp-Growth is an algorithm built to increase the efficiency of Apriori, so it is not expected to consume more time in mining for patterns than Apriori, but from Figure 4 we can witness it to be the case. This is because since on the 50% threshold there is only 1 frequent pattern (figure 2), Apriori has to check only one permutation of the itemsets, thus needing only one database scan. However, on Fp-Growth, the algorithm needs multiple database scans to build the tree, as a result, it performs worse in this particular case.

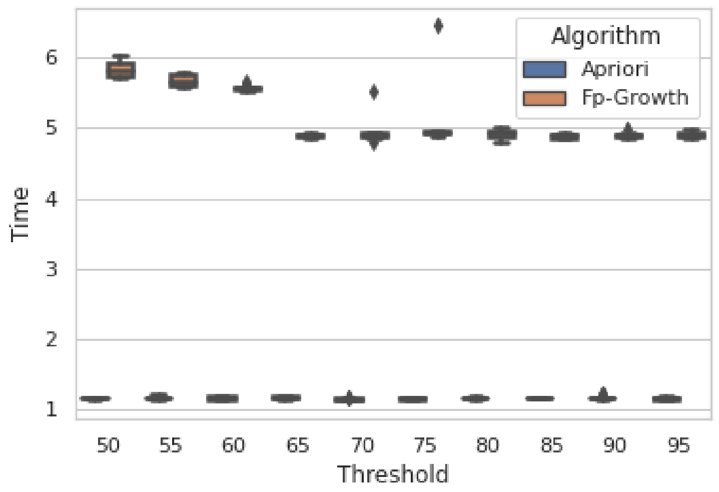


Figure 4: Comparison of the algorithms on the Kosarak Dataset.

Furthermore, the aforementioned approach 2 improves the performance of the algorithms significantly. We had expected that due to pruning all the records that do not meet a certain threshold, the algorithms will not have to try out different permutations, which will significantly improve their performance. However, it poses a computational overhead on both of the algorithms, because cleaning a huge dataset is incredibly time-consuming. For both Chess and Kosarak datasets, the required time for cleaning is around 20 minutes and 2 hours and 30 minutes respectively.

Upon applying method 2 with both of the algorithms, we have achieved significantly better results than the previous method. Figure 5 reports the time required for method 2 on the chess dataset on the different thresholds. Comparing Figures 3 and 5, we can see that on the 65% threshold, the previous method consumed around 40 seconds, and the current method takes only 8 seconds. Which persists for the rest of the thresholds

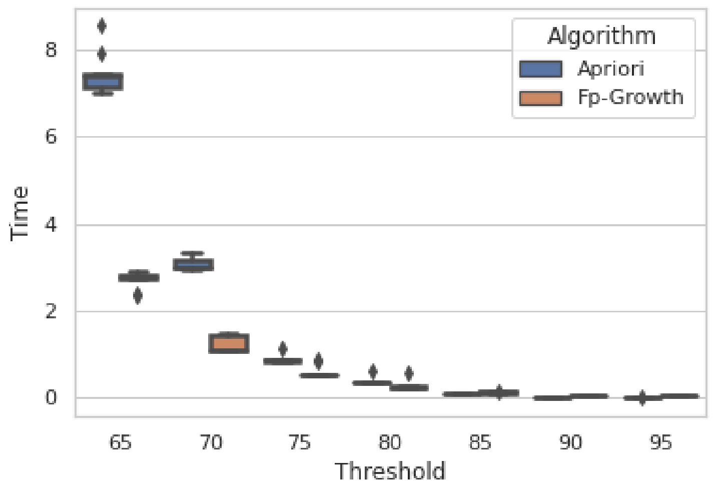


Figure 5: Comparison between algorithms on pruned Chess dataset.

Upon applying method 2 on Kosarak dataset, we have achieved a relatively better performance than before. Figure 6 reports the time required for method 2 on the Kosrak dataset on the different thresholds. Compared to Figure 4, it is evident that the performance increased significantly. Where on the previous method, Fp-growth took around 6 seconds to finish each iteration, in method 2 it takes around 4 seconds.

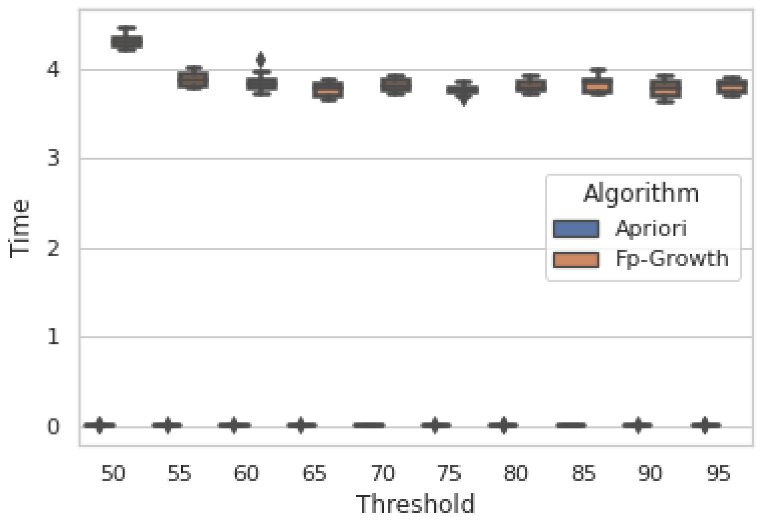


Figure 6: Comparison between algorithms on pruned Kosarak dataset.