COL 761 : Data Mining Spring 2025

Assignment 1

Total Points : 100 Due Date: 11:59 pm, Monday, Feb 10 2025

Daksh Dhaker (2022CS51264)
Parth Verma (2022CS11936)
Umang Tripathi (2022CS51134)

1) Part-1: Frequent Itemset Mining (15 marks)

1.1) Analysis

Method Used:

- Support threshold is taken at five different values: 5%, 10%, 25%, 50%, and 90%.
- For every value, Apriori and FP-Growth algorithms are run on the given dataset webdocs.dat
- These values are then plotted against their respective threshold values.

Timeouts and Errors:

- Timeouts and Errors are handled separately. If no file is generated due to timeouts or errors, then an empty file is generated.
- Timeout is taken to be 60 mins.
- In case of Timeout the value of 60 min is used for the Plot.
- In case of an error, the time it took from the beginning till the halting due to error, that time is used for plotting (This case arose only in 5% FP-Growth due to Memory leak).

1.2) Observations

Time Taken is rounded off to the second decimal place.

Support Threshold	Time Taken by Apriori (seconds)	Time Taken by FP-Growth (seconds)
5%	3600 (Timeout)	888.38 (Memory leak)
10%	933.41	158.27
25%	58.68	61.47
50%	59.41	58.43
90%	58.95	58.74

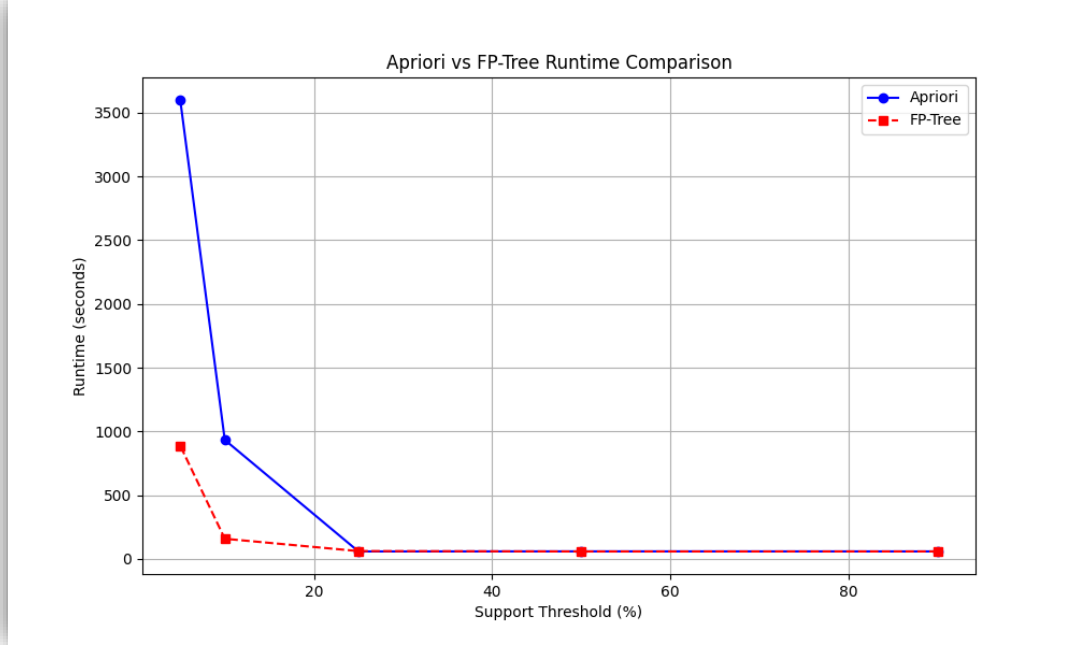
Note :

- In the case of Apriori with a 5% support threshold, the program exceeded the time limit (3600 seconds). Hence, an empty file is generated and its time taken is recorded as the timeout limit.

- In the case of FP-Growth with a 5% support threshold, the program terminated due to memory leak. Hence its time is also taken as the time it took from the beginning till its termination.

- In the case of the support threshold of 90%, there were no frequent itemset. So Apriori and FP-Growth both terminated with an error message “no (frequent) items found”. Hence an empty file is created for each algorithm.

Plot



1.3) Comments and Analysis:

- **General Performance Overview :**
    - Both algorithms take approximately 55 seconds to read data from webdocs.dat, so the actual frequent itemset mining time is the recorded runtime minus 55 seconds.
    - Apriori's runtime decreases as the support threshold increases, whereas FP-Growth remains fairly consistent across different thresholds.

- **Performance at Low Support Thresholds (5%, 10%)**
  (FP-Growth is much faster than Apriori)
    - In the case of 5%, Apriori Algorithm exceeded the time limit whereas FP-Growth started writing the frequent itemsets but before finishing it, it faced a memory leak.
    - FP-Growth is almost 6 times faster, proving its efficiency in handling large candidate itemsets.

- **Performance at Mid Support Thresholds (25%)**
  (FP-Growth is slightly slower than Apriori)
    - The performance difference is minor, with Apriori being slightly faster, but both algorithms show similar efficiency.

- **Performance at High Support Thresholds (50%, 90%)**
  (Both are nearly equal)
    - Both algorithms have nearly identical performance, meaning the choice of algorithm is less relevant for very high support levels.

- **Observed Trends and Efficiency**
    - Apriori performs poorly at low support thresholds (5%, 10%), where it becomes extremely slow due to candidate generation overhead.
    - FP-Growth is significantly better for low support thresholds because it builds a compact FP-tree instead of generating numerous candidate sets.
    - FP-Growth maintains a relatively stable runtime, whereas Apriori exhibits a steep runtime drop as support increases.

Explanation of Performance Differences

- Apriori suffers at low support thresholds due to two reasons.
- First one, due to its exponential candidate set generation.
- Second reason is that, if the maximum frequent itemset is of size K. Then it takes K iterations over the entire Transactions database. Since Transactions databases are so large that they generally don't reside in Main Memory but in Swap Spaces (like disks). Accessing these Swap spaces takes more times (1000 cycles more) than accessing Main Memory.
- Empirically, FP-growth is an order of magnitude faster than Apriori due to the following reasons.
- There is no explicit candidate generation and candidate testing. It involves only counting.
- Also since the FP-tree is assumed to be much more compact than the transaction database. It can reside in Main Memory. Hence to count the frequency of a frequent itemset. It does not have to repeatedly access the Swap Space (disk). It can directly access the FP-tree in Main Memory.
- The accesses of Swap Space are constant which is either 2 (in case of frequency based ordering) or 1 (pre-defined ordering eg: lexicographical ordering). Due to this, It is way faster than Apriori.

1.4) Conclusion

In conclusion, FP-Growth significantly outperforms Apriori at low support thresholds (5% and 10%), where Apriori suffers from exponential candidate generation, making it extremely slow. At moderate support (25%), both algorithms perform similarly, with Apriori being slightly faster. At higher support levels (50% and 90%), the performance gap disappears, and both algorithms exhibit nearly identical runtimes. Therefore, FP-Growth is the preferred choice for mining frequent itemsets at low support thresholds, while at high support levels, either algorithm can be used effectively.