

DATA ANALYTICS –1
ASSOCIATION RULE MINING PROJECT

Optimising and Experimenting with Classical Algorithms

SUBMITTED BY:-
Anjali Bhatnagar 2019201012

TASKS UNDER CONSIDERATION

The main aim of the project is experimenting with the known association rule mining algorithms and then improvising them further using the known techniques to reduce the processing time . From this we will come to see how different factors affect the association rule mining and when the improvisations are effective . The algorithms that we have considered for this project are Apriori and FP Growth. For the datasets used in this project we referred SPMF – An open source Data Mining Library. We have used the following five datasets majorly for this project:

- Sign Dataset
- Fifa Dataset
- Kosarak Dataset
- Bible Dataset
- Leviathan Dataset

Flow of the Project :

- Using inbuilt Apriori Algorithm
- Building Apriori Algorithm from scratch
- Using the Partitioning Method to optimize Apriori
- Using the Transaction Reduction as an optimization for Apriori
- Experimenting on Apriori with different datasets
- Using inbuilt FP Growth on the given datasets
- Implementing FP growth from scratch and optimizing it
- Comparison based on parameters and different datasets
- Conclusion

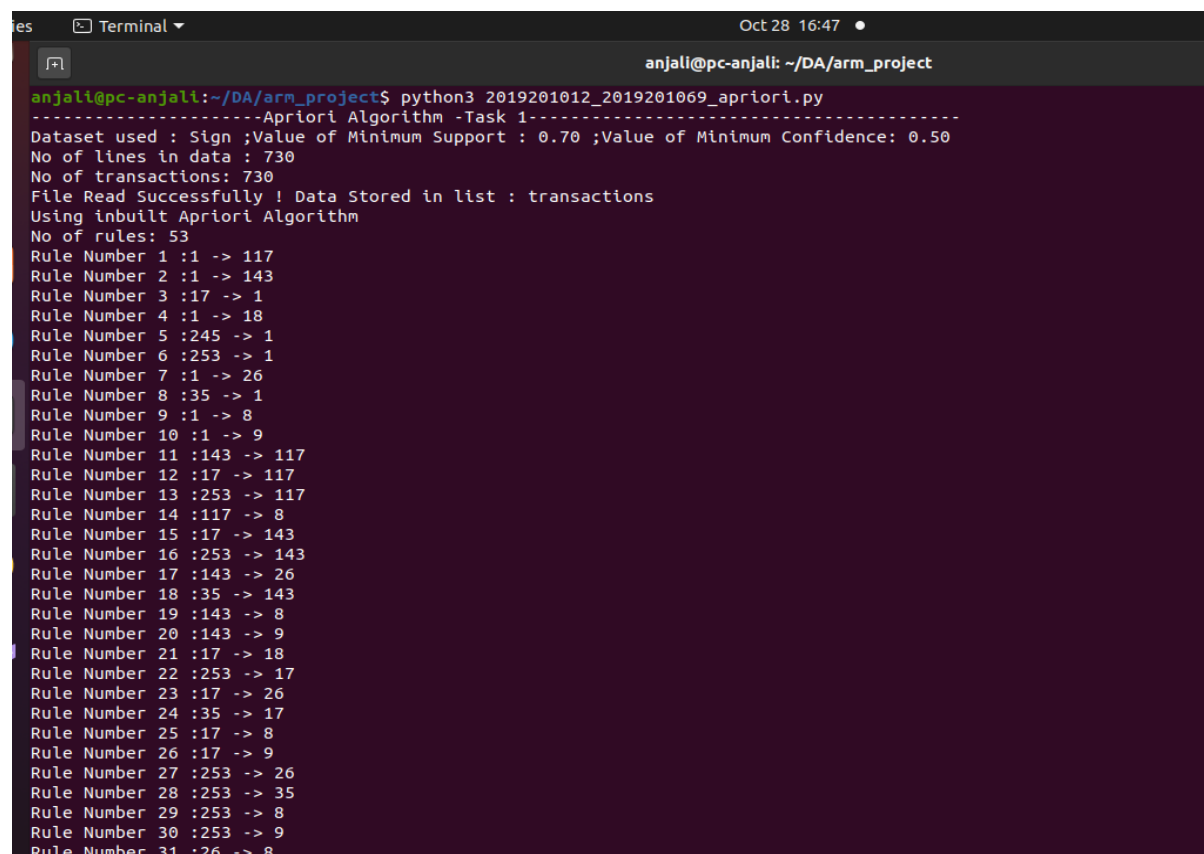
TASK 1 – Apriori Algorithm

The major focus of task1 is to implement the Apriori Algorithm .We have implemented it using the inbuilt library as well as from scratch and also improvised it using the partitioning method and the transaction reduction method.

1.1 Implementing Apriori Algorithm using Inbuilt Libraries:

We ran the inbuilt Apriori Algorithm and analysed on the basis of it as well . The major comparisons are though drawn on the basis of the Apriori algorithm from scratch and the Apriori Algorithm we optimized using the various optimization techniques.

So lets see the analysis of results on Sign Dataset for the inbuilt Apriori and our from scratch implementation of Apriori.



```
anjali@pc-anjali: ~/DA/arm_project
python3 2019201012_2019201069_apriori.py
-----Apriori Algorithm -Task 1-----
Dataset used : Sign ;Value of Minimum Support : 0.70 ;Value of Minimum Confidence: 0.50
No of lines in data : 730
No of transactions: 730
File Read Successfully ! Data Stored in list : transactions
Using inbuilt Apriori Algorithm
No of rules: 53
Rule Number 1 :1 -> 117
Rule Number 2 :1 -> 143
Rule Number 3 :17 -> 1
Rule Number 4 :1 -> 18
Rule Number 5 :245 -> 1
Rule Number 6 :253 -> 1
Rule Number 7 :1 -> 26
Rule Number 8 :35 -> 1
Rule Number 9 :1 -> 8
Rule Number 10 :1 -> 9
Rule Number 11 :143 -> 117
Rule Number 12 :17 -> 117
Rule Number 13 :253 -> 117
Rule Number 14 :117 -> 8
Rule Number 15 :17 -> 143
Rule Number 16 :253 -> 143
Rule Number 17 :143 -> 26
Rule Number 18 :35 -> 143
Rule Number 19 :143 -> 8
Rule Number 20 :143 -> 9
Rule Number 21 :17 -> 18
Rule Number 22 :253 -> 17
Rule Number 23 :17 -> 26
Rule Number 24 :35 -> 17
Rule Number 25 :17 -> 8
Rule Number 26 :17 -> 9
Rule Number 27 :253 -> 26
Rule Number 28 :253 -> 35
Rule Number 29 :253 -> 8
Rule Number 30 :253 -> 9
Rule Number 31 :26 -> 8
```

```
es Terminal ▾ Oct 28 16:47 •
anjali@pc-anjali: ~/DA/arm_project

Rule Number 31 :26 -> 8
Rule Number 32 :35 -> 8
Rule Number 33 :9 -> 8
Rule Number 34 :17 -> 1 & 143
Rule Number 35 :253 -> 1 & 143
Rule Number 36 :1 -> 143 & 8
Rule Number 37 :17 -> 1 & 253
Rule Number 38 :17 -> 1 & 8
Rule Number 39 :253 -> 1 & 8
Rule Number 40 :253 -> 17 & 8
No of valid rules: 40
Overall time for inbuilt apriori: 0.022993087768554688
Using Apriori Developed from scratch
87
Overall time for apriori implementation from scratch: 0.1045525074005127
Rules we got
Rule Number 1 : ['1'] -> ['117']
Rule Number 2 : ['18'] -> ['1']
Rule Number 3 : ['35'] -> ['17']
Rule Number 4 : ['253'] -> ['8']
Rule Number 5 : ['143'] -> ['26']
Rule Number 6 : ['26'] -> ['253']
Rule Number 7 : ['35'] -> ['1']
Rule Number 8 : ['1'] -> ['35']
Rule Number 9 : ['17'] -> ['1']
Rule Number 10 : ['8'] -> ['26']
Rule Number 11 : ['253'] -> ['143']
Rule Number 12 : ['143'] -> ['253']
Rule Number 13 : ['35'] -> ['8']
Rule Number 14 : ['8'] -> ['35']
Rule Number 15 : ['253'] -> ['17']
Rule Number 16 : ['17'] -> ['253']
Rule Number 17 : ['1'] -> ['143']
Rule Number 18 : ['143'] -> ['1']
Rule Number 19 : ['1'] -> ['26']
Rule Number 20 : ['26'] -> ['1']
Rule Number 21 : ['253'] -> ['35']
Rule Number 22 : ['35'] -> ['253']
Rule Number 23 : ['245'] -> ['1']
Rule Number 24 : ['1'] -> ['8']
Rule Number 25 : ['8'] -> ['1']
Rule Number 26 : ['17'] -> ['26']
Rule Number 27 : ['26'] -> ['17']
Rule Number 28 : ['35'] -> ['143']
Rule Number 29 : ['143'] -> ['35']
Rule Number 30 : ['117'] -> ['253']
Rule Number 31 : ['117'] -> ['8']
Rule Number 32 : ['8'] -> ['117']
Rule Number 33 : ['17'] -> ['8']
Rule Number 34 : ['8'] -> ['17']
Rule Number 35 : ['17'] -> ['117']
Rule Number 36 : ['117'] -> ['17']
Rule Number 37 : ['9'] -> ['17']
Rule Number 38 : ['9'] -> ['8']
Rule Number 39 : ['8'] -> ['9']
Rule Number 40 : ['253'] -> ['9']
Rule Number 41 : ['9'] -> ['253']
Rule Number 42 : ['17'] -> ['18']
Rule Number 43 : ['18'] -> ['17']
Rule Number 44 : ['1'] -> ['9']
Rule Number 45 : ['9'] -> ['1']
Rule Number 46 : ['253'] -> ['1']
Rule Number 47 : ['1'] -> ['253']
Rule Number 48 : ['17'] -> ['143']
Rule Number 49 : ['143'] -> ['17']
Rule Number 50 : ['9'] -> ['143']
Rule Number 51 : ['143'] -> ['9']
Rule Number 52 : ['143'] -> ['117']
Rule Number 53 : ['117'] -> ['143']
Rule Number 54 : ['143'] -> ['8']
Rule Number 55 : ['18'] -> ['143']
```

```
es Terminal ▾ Oct 28 16:47 •
anjali@pc-anjali: ~/DA/arm_project

Rule Number 16 : ['2'] -> ['253']
Rule Number 17 : ['1'] -> ['143']
Rule Number 18 : ['143'] -> ['1']
Rule Number 19 : ['1'] -> ['26']
Rule Number 20 : ['26'] -> ['1']
Rule Number 21 : ['253'] -> ['35']
Rule Number 22 : ['35'] -> ['253']
Rule Number 23 : ['245'] -> ['1']
Rule Number 24 : ['1'] -> ['8']
Rule Number 25 : ['8'] -> ['1']
Rule Number 26 : ['17'] -> ['26']
Rule Number 27 : ['26'] -> ['17']
Rule Number 28 : ['35'] -> ['143']
Rule Number 29 : ['143'] -> ['35']
Rule Number 30 : ['117'] -> ['253']
Rule Number 31 : ['117'] -> ['8']
Rule Number 32 : ['8'] -> ['117']
Rule Number 33 : ['17'] -> ['8']
Rule Number 34 : ['8'] -> ['17']
Rule Number 35 : ['17'] -> ['117']
Rule Number 36 : ['117'] -> ['17']
Rule Number 37 : ['9'] -> ['17']
Rule Number 38 : ['9'] -> ['8']
Rule Number 39 : ['8'] -> ['9']
Rule Number 40 : ['253'] -> ['9']
Rule Number 41 : ['9'] -> ['253']
Rule Number 42 : ['17'] -> ['18']
Rule Number 43 : ['18'] -> ['17']
Rule Number 44 : ['1'] -> ['9']
Rule Number 45 : ['9'] -> ['1']
Rule Number 46 : ['253'] -> ['1']
Rule Number 47 : ['1'] -> ['253']
Rule Number 48 : ['17'] -> ['143']
Rule Number 49 : ['143'] -> ['17']
Rule Number 50 : ['9'] -> ['143']
Rule Number 51 : ['143'] -> ['9']
Rule Number 52 : ['143'] -> ['117']
Rule Number 53 : ['117'] -> ['143']
Rule Number 54 : ['143'] -> ['8']
Rule Number 55 : ['18'] -> ['143']
```

```
Terminal
Oct 28 16:47
anjali@pc-anjali: ~/DA/arm_project

Rule Number 70 : ['253', '1'] -> ['17']
Rule Number 71 : ['17'] -> ['1', '8']
Rule Number 72 : ['1'] -> ['17', '8']
Rule Number 73 : ['8'] -> ['1', '17']
Rule Number 74 : ['1', '17'] -> ['8']
Rule Number 75 : ['17', '8'] -> ['1']
Rule Number 76 : ['1', '8'] -> ['17']
Rule Number 77 : ['17'] -> ['1', '143']
Rule Number 78 : ['143'] -> ['1', '17']
Rule Number 79 : ['1', '17'] -> ['143']
Rule Number 80 : ['17', '143'] -> ['1']
Rule Number 81 : ['1', '143'] -> ['17']
Rule Number 82 : ['253'] -> ['1', '8']
Rule Number 83 : ['1'] -> ['253', '8']
Rule Number 84 : ['8'] -> ['253', '1']
Rule Number 85 : ['253', '1'] -> ['8']
Rule Number 86 : ['253', '8'] -> ['1']
Rule Number 87 : ['1', '8'] -> ['253']
Now lets improvise the Apriori Algorithm built from scratch:
-----Improvisation Number : 1-----
Partitioning
Time Required for partitioning : 1.8358230590820312e-05
Length of partition 1 : 146
Length of partition 2 : 146
Length of partition 3 : 146
Length of partition 4 : 146
Length of partition 5 : 146
87
Total time: 0.0809483528137207
-----Improvisation Number : 2-----
Transaction Reduction:
No of frequent items present in the data of length 1 : 13
No of frequent items present in the data of length 2 : 33
No of frequent items present in the data of length 3 : 7
No of frequent items present in the data of length 4 : 0
87
0.15403485298156738
anjali@pc-anjali:~/DA/arm_project$
```

So the major difference between the inbuilt Apriori and the Apriori that we built from scratch is in the number of rules. We had a slightly higher number of rules as compared to the inbuilt Apriori the reason being their left hand side of the rule i.e. the antecedent comprises of only single items where as there is no such constraint in our rules that we obtained from our Apriori Algorithm from scratch .

So as we can see from the screenshots attached lets write down the time taken by each apriori algorithm run :

DATASET : SIGN.TXT

| Algorithm | Support | Confidence | Time |
|-------------------------------------|---------|------------|-------|
| Inbuilt Apriori | 0.70 | 0.50 | 0.022 |
| Apriori from Scratch | 0.70 | 0.50 | 0.104 |
| Apriori using Partitioning Method | 0.70 | 0.50 | 0.080 |
| Apriori Using Transaction Reduction | 0.70 | 0.50 | 0.154 |

1.2 Implementing Apriori Algorithm from scratch and optimizing it:

We have implemented the original Apriori algorithm from scratch by first finding frequent itemsets of size 1 which follow the minimum support constraint and then from these of size 2 and so on following the naive Apriori Approach. After finding the frequent itemsets we then found the association rules for these. We observed that it requires a lot of time going over the entire database and all the transactions . So inorder to reduce the time required by our Apriori algorithm we used the following 2 optimization approaches :

- **Partitioning Method**

Concept Used : If the database D is divided into n partitions and frequent itemset mining is performed in each of them individually, any itemset that is potentially frequent with respect to D occur as a frequent itemset in at least one of those n partitions. Now the run of F IM on each of those partitions can be done paralelly and all the local frequent itemsets generated can be checked for being globally frequent by a scan of the database again. So we used this approach for partitioning our database i.e. the transactions into partitions and then mined them . However as we did not use any parallelizing softwares or GPU's to run the code paralelly on different nodes , to find the overall time of this method we have considered that for a single partition as other partitions can also be done simultaneously.

For some datasets we got a reduction in time but for some there was not much reduction . This is majorly because of different sizes of data that we have used and the varied values of support and confidence.

We will be seeing further in the analysis regarding how these algorithms perform under different conditions.

- **Transaction Reduction Method**

Concept Used : A transaction that does not contain any frequent k-itemsets cannot contain any frequent $(k + 1)$ -itemsets. Therefore, such a transaction can be marked or removed from further consideration. We have also used this as an optimization approach and analysed the data accordingly .

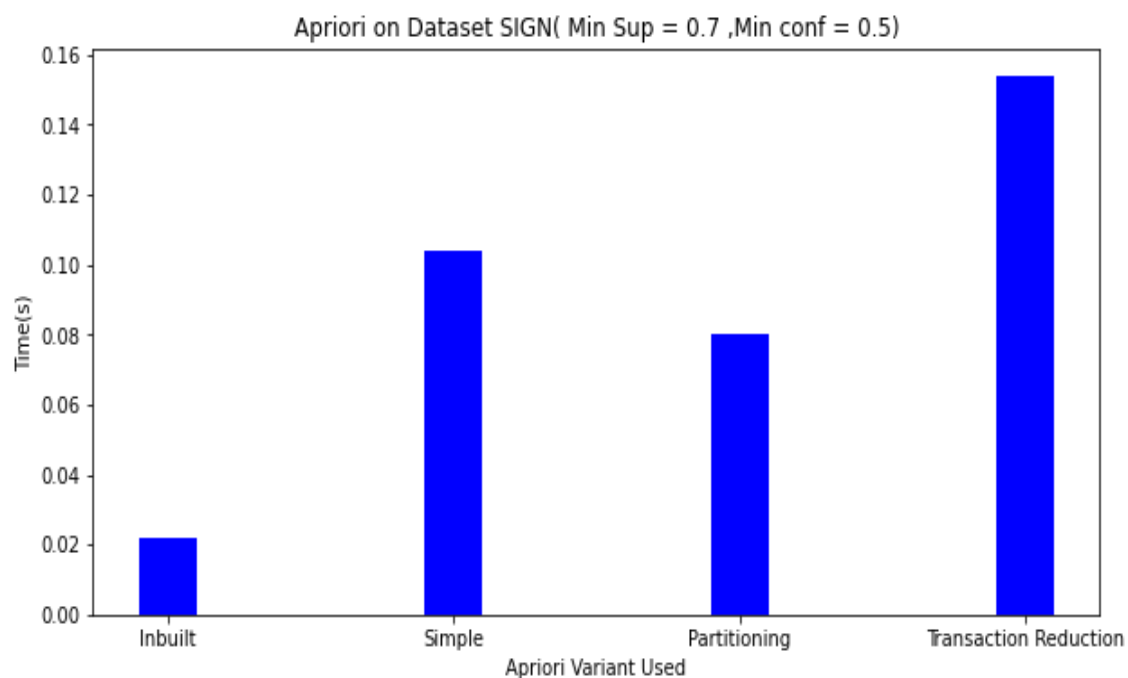
1.3 Analysis of Apriori Algorithm on different Datasets :

Smaller values of support and confidence were taking higher amount of time so for the purpose of these experiments we have considered bigger values of support and confidence which are also in correlation with the mining concepts that we should mine only the useful rules and patterns.

Data Set Used : SIGN.txt

Number of transactions : 730

| Algorithm | Support | Confidence | Time |
|-------------------------------------|---------|------------|-------|
| Inbuilt Apriori | 0.70 | 0.50 | 0.022 |
| Apriori from Scratch | 0.70 | 0.50 | 0.104 |
| Apriori using Partitioning Method | 0.70 | 0.50 | 0.080 |
| Apriori Using Transaction Reduction | 0.70 | 0.50 | 0.154 |



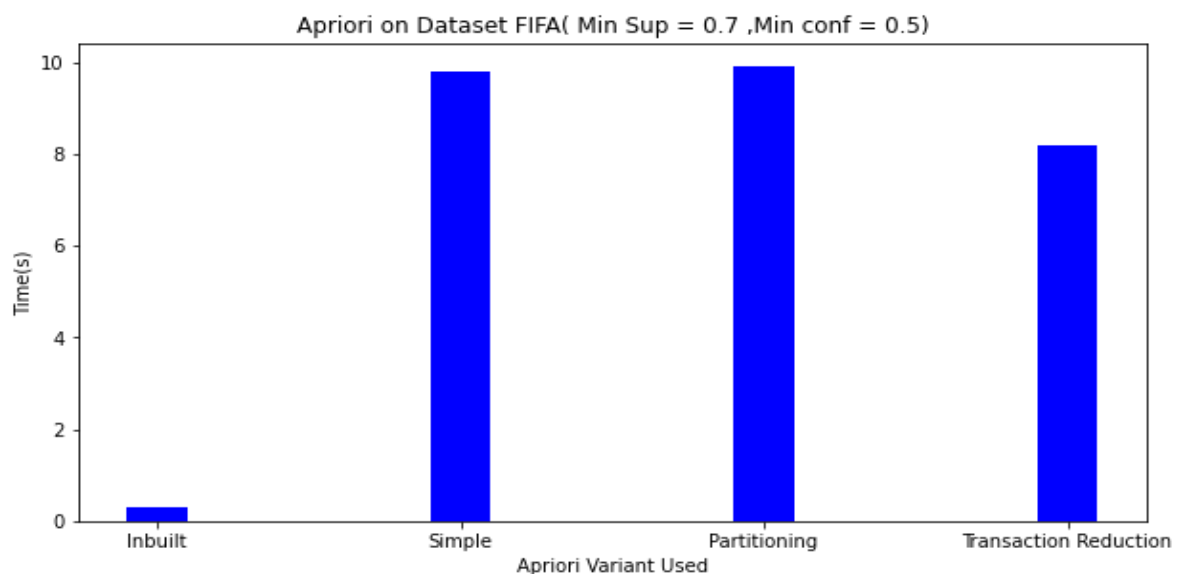
From the algorithm we developed from scratch the Partitioning algorithm worked best for this dataset . Number of partitions that we used were 5.

Data Set Used : Fifa

Number of transactions : 20450

```
8.194854497909546
Overall Time taken by different Approaches for fifa.txt
Overall time for inbuilt apriori: 0.2851130962371826
Overall time for apriori implementation from scratch: 9.79949140548706
Total time taken by partitioning: 9.926949262619019
Total time taken by Transaction Reduction: 8.194854497909546
anjali@pc-anjali:~/DA/arm_project$
```

| Algorithm | Support | Confidence | Time |
|-------------------------------------|---------|------------|------|
| Inbuilt Apriori | 0.70 | 0.50 | 0.28 |
| Apriori from Scratch | 0.70 | 0.50 | 9.79 |
| Apriori using Partitioning Method | 0.70 | 0.50 | 9.92 |
| Apriori Using Transaction Reduction | 0.70 | 0.50 | 8.19 |

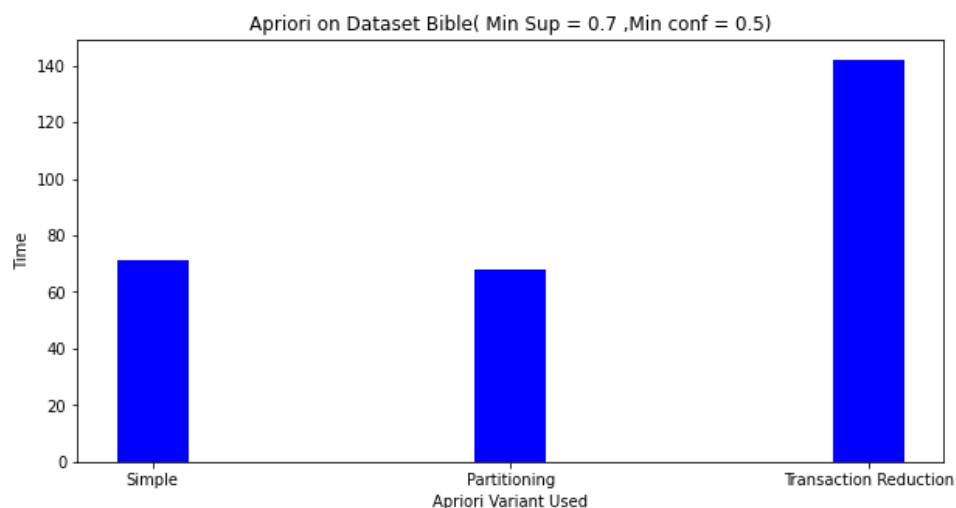


Dataset Used : Bible

No of transactions: 36369

```
142.89940094947815
Overall Time taken by different Approaches for bible.txt
Overall time for inbuilt apriori: 0.3195974826812744
Overall time for apriori implementation from scratch: 71.44380950927734
Total time taken by partitioning: 68.37608289718628
Total time taken by Transaction Reduction: 142.89940094947815
```

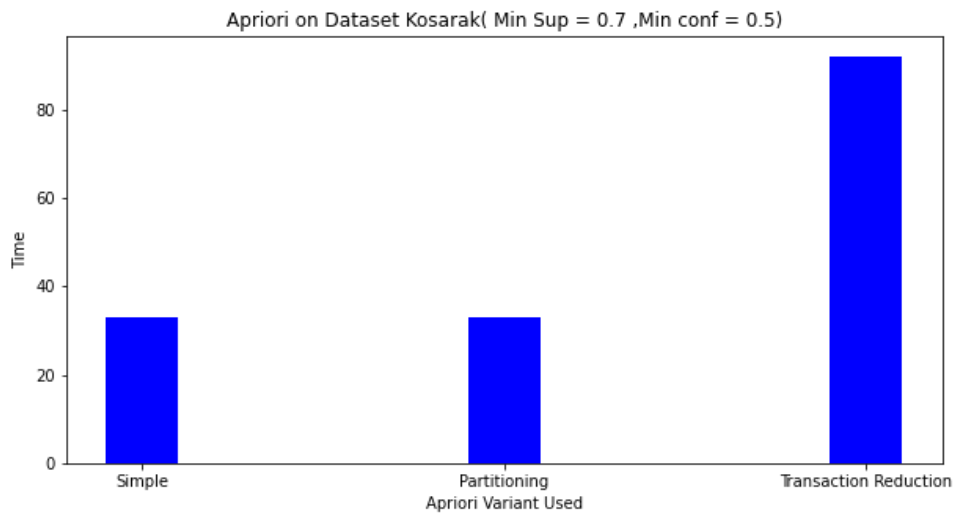
For this dataset we could not find any frequent itemsets for the given minimum support. Important observation that we had was that partitioning took the least time as expected as we could see from each partition which will be frequent and which will not be frequent and further scanning of the entire database could be avoided .



A similar result was seen for Kosarak Dataset where there werent much rules to be formed given the minimum support value and also as the data set was very big it gave us slight idea regarding the time parameters as well.

Dataset Kosarak :

| Algorithm | Support | Confidence | Time |
|-------------------------------------|---------|------------|-------|
| Apriori from Scratch | 0.70 | 0.50 | 32.09 |
| Apriori using Partitioning Method | 0.70 | 0.50 | 33.01 |
| Apriori Using Transaction Reduction | 0.70 | 0.50 | 96.59 |



Dataset : Leviathan

| Algorithm | Support | Confidence | Time |
|-------------------------------------|---------|------------|-------|
| Apriori from Scratch | 0.70 | 0.50 | 7.66 |
| Apriori using Partitioning Method | 0.70 | 0.50 | 7.42 |
| Apriori Using Transaction Reduction | 0.70 | 0.50 | 15.91 |

```

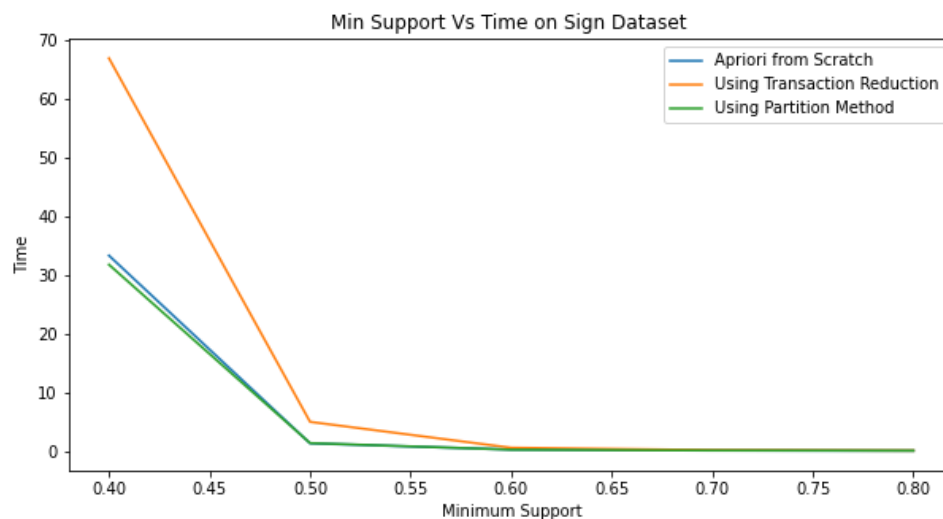
15.9132250787354
Overall Time taken by different Approaches for leviathan.txt
Overall time for inbuilt apriori: 0.08872628211975098
Overall time for apriori implementation from scratch: 7.668760538101196
Total time taken by partitioning: 7.47244930267334
Total time taken by Transaction Reduction: 15.9132250787354
anjali@pc-anjali:~/DA/arm_project$ 
(n.txt",0.5,0.5)"""

```

This was with respect to the different implementations of Apriori Used . Now we will compare these on the basis of support . We ran the Apriori Algorithm (scratch as well as the optimized one) for different values of support .

The table below shows the observations that we recorded in terms of time :

| Support | 0.8 | 0.7 | 0.6 | 0.5 | 0.4 |
|--------------------------------------|------------|------------|------------|------------|------------|
| Apriori from Scratch | 0.048 | 0.104 | 0.23 | 1.30 | 33.3 |
| Apriori Partitioning | 0.047 | 0.08 | 0.22 | 1.34 | 31.75 |
| Apriori Transaction Reduction | 0.064 | 0.159 | 0.55 | 4.96 | 66.93 |



Conclusion of Task 1:

This task clearly helped us to learn about the Apriori Algorithm and how optimizing it helps us. Different datasets showed different outcomes for different values of minimum support and minimum confidence. Also we saw the relation between increasing the support and decrease in the time and decreasing the support and increase in time as the number of frequent item sets would increase and thus will the time needed to process them .

TASK 2 – FP Growth Algorithm

The major focus of task2 is to implement the FP Growth Algorithm .We have implemented it using the inbuilt library as well as from scratch and also improvised it using the merging technique.

Approach Used : Bottom Up Technique and Merging

In the Bottom UP Technique the database representing the frequent items is compressed into a frequent pattern tree i.e. FP-tree, which retains the itemset association information. It then divides the compressed database into a set of conditional databases each associated with one frequent item .

Each database is mined separately . It is usually less time consuming then Apriori Algorithm.

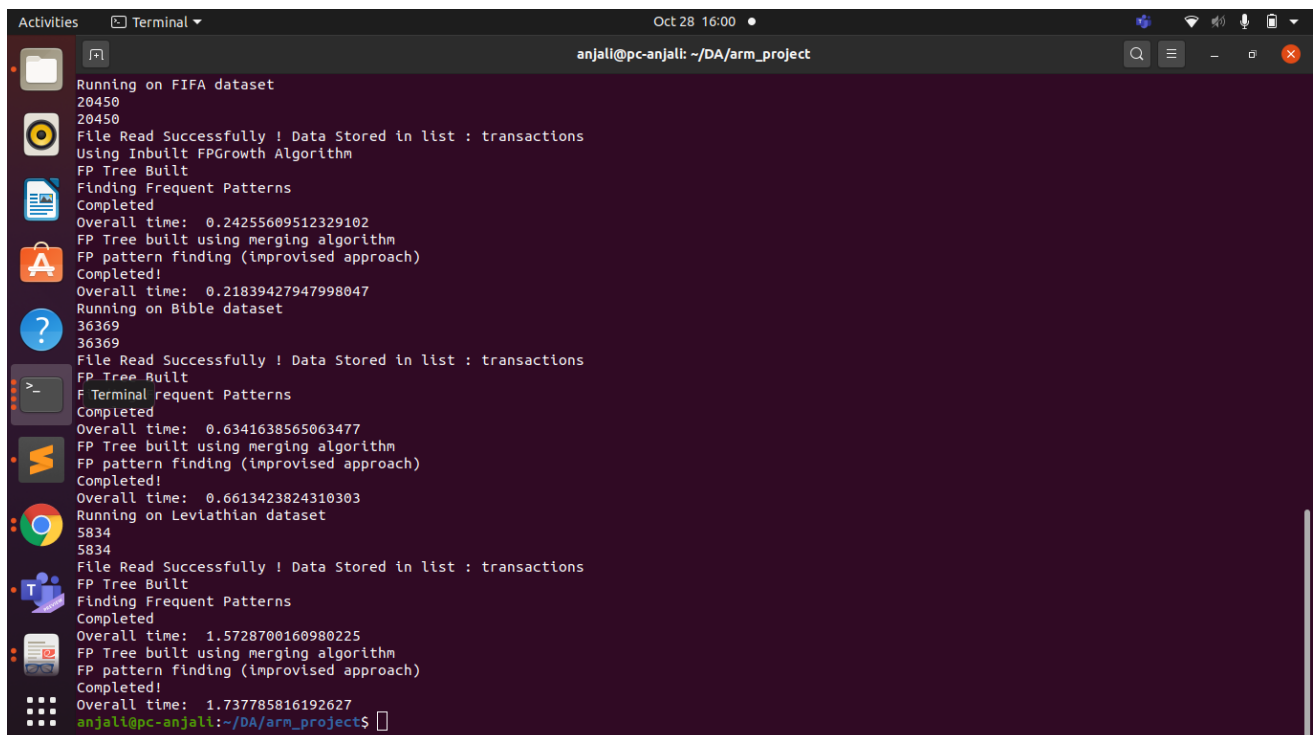
Construction Steps:

- Construct the FP Tree – Create the root of FP Tree as null , then scan the database and sort the list of frequent items and their counts after the scanning is done . After that take each transaction and sort it according to each item count on it. Then we invoke the insert transaction function after the node table and linkage table is made. If item has node containing the same item which transaction has at the level of insertion then increase its count or if not present insert it . We first check if its not present then we insert it with count as 1.
- The tree is traversed in bottom up manner . The link cells contain the linkages and we also have a header table to traverse the tree in bottom up manner.
- Mining the FP Tree by traversing it in bottom up manner – starting with the item with the lowest count in header table then we traverse the nodes with that item using the link cell at each node we will traverse bottom up and then generate the frequent pattern base for that item and then from these make the conditional fp tree ad thus doing this procedure for each item we generate the frequent patterns. Then these patterns can be mined to form rules.

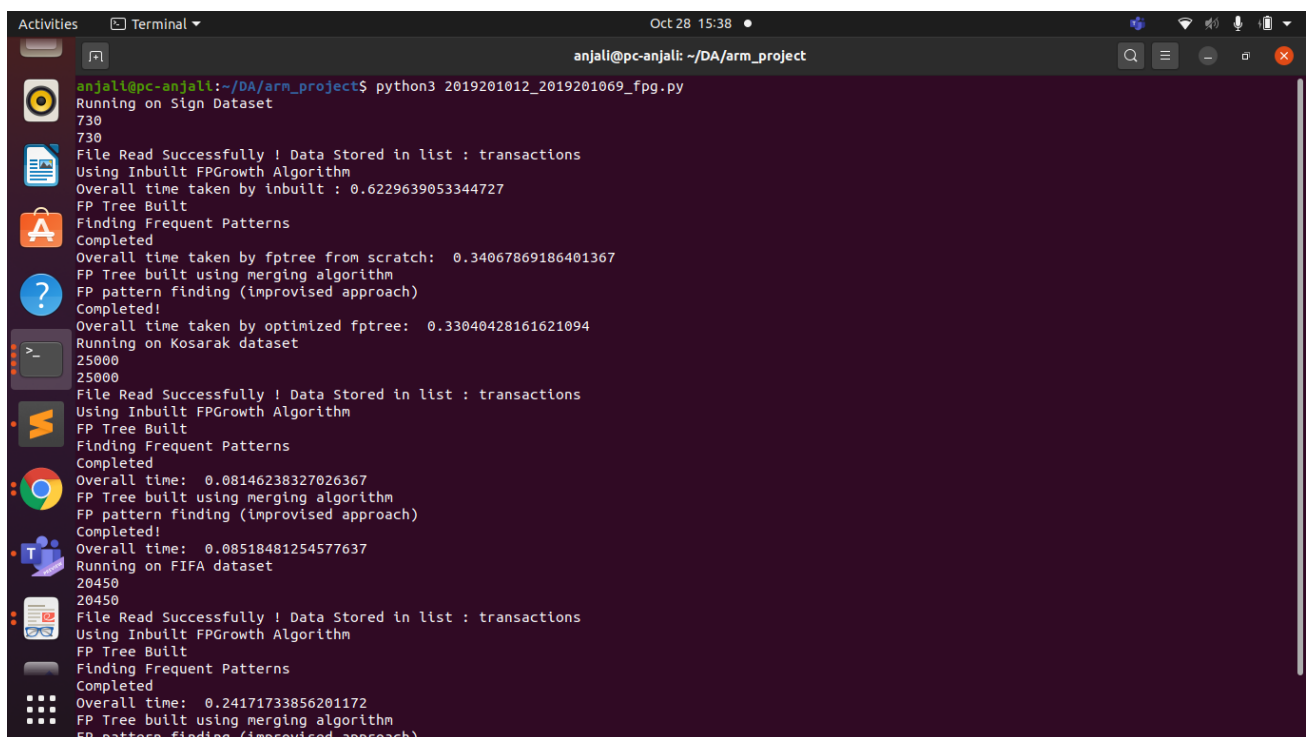
Optimizations Done:

- For each node we added a visited parameter . This is done to exploit the property that while mining the path in bottom up manner ,we will also go through subpaths which will be again later traversed when the item stored in that node is visited in the header table and thus can be optimized to improve the run time.
- All other approach is same as the bottom up approach . So if we know that a path is traversed we can use this fact for improving the run time as explained in the example mentioned in the doc.

Screenshots of the Analysis Done :



```
anjali@pc-anjali: ~/DA/arm_project
Running on FIFA dataset
20450
20450
File Read Successfully ! Data Stored in list : transactions
Using Inbuilt FPGrowth Algorithm
FP Tree Built
Finding Frequent Patterns
Completed
Overall time: 0.24255609512329102
FP Tree built using merging algorithm
FP pattern finding (improvised approach)
Completed!
Overall time: 0.21839427947998047
Running on Bible dataset
36369
36369
File Read Successfully ! Data Stored in list : transactions
FP Tree Built
Finding Frequent Patterns
Completed
Overall time: 0.6341638565063477
FP Tree built using merging algorithm
FP pattern finding (improvised approach)
Completed!
Overall time: 0.6613423824310303
Running on Leviathan dataset
5834
5834
File Read Successfully ! Data Stored in list : transactions
FP Tree Built
Finding Frequent Patterns
Completed
Overall time: 1.5728700160980225
FP Tree built using merging algorithm
FP pattern finding (improvised approach)
Completed!
Overall time: 1.737785816192627
anjali@pc-anjali:~/DA/arm_project$
```

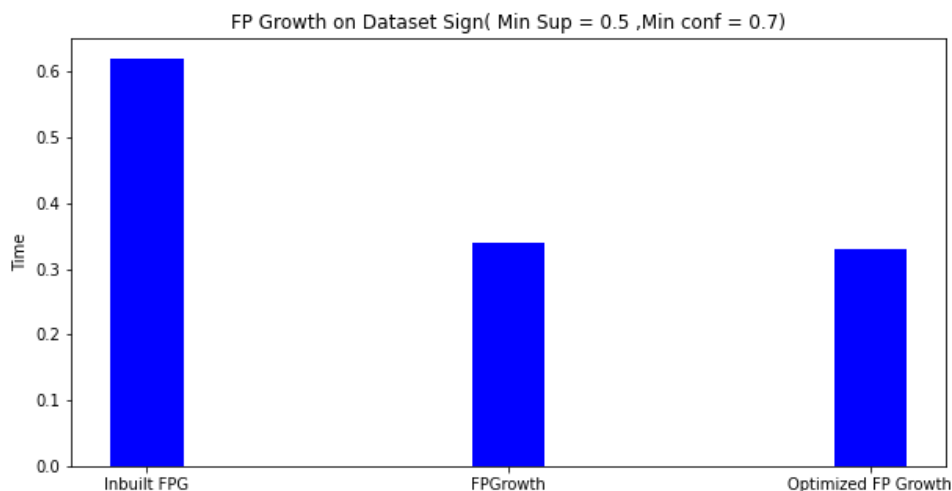


```
anjali@pc-anjali:~/DA/arm_project$ python3 2019201012_2019201069_fpg.py
Running on Sign Dataset
730
730
File Read Successfully ! Data Stored in list : transactions
Using Inbuilt FPGrowth Algorithm
Overall time taken by inbuilt : 0.6229639053344727
FP Tree Built
Finding Frequent Patterns
Completed
Overall time taken by fptree from scratch: 0.34067869186401367
FP Tree built using merging algorithm
FP pattern finding (improvised approach)
Completed!
Overall time taken by optimized fptree: 0.33040428161621094
Running on Kosarak dataset
25000
25000
File Read Successfully ! Data Stored in list : transactions
Using Inbuilt FPGrowth Algorithm
FP Tree Built
Finding Frequent Patterns
Completed
Overall time: 0.08146238327026367
FP Tree built using merging algorithm
FP pattern finding (improvised approach)
Completed!
Overall time: 0.08518481254577637
Running on FIFA dataset
20450
20450
File Read Successfully ! Data Stored in list : transactions
Using Inbuilt FPGrowth Algorithm
FP Tree Built
Finding Frequent Patterns
Completed
Overall time: 0.24171733856201172
FP Tree built using merging algorithm
FP pattern finding (improvised approach)
```

Value of min_support and min_confidence : 0.5 and 0.7 respectively . We experimented with many datasets and this is the result that we obtained.

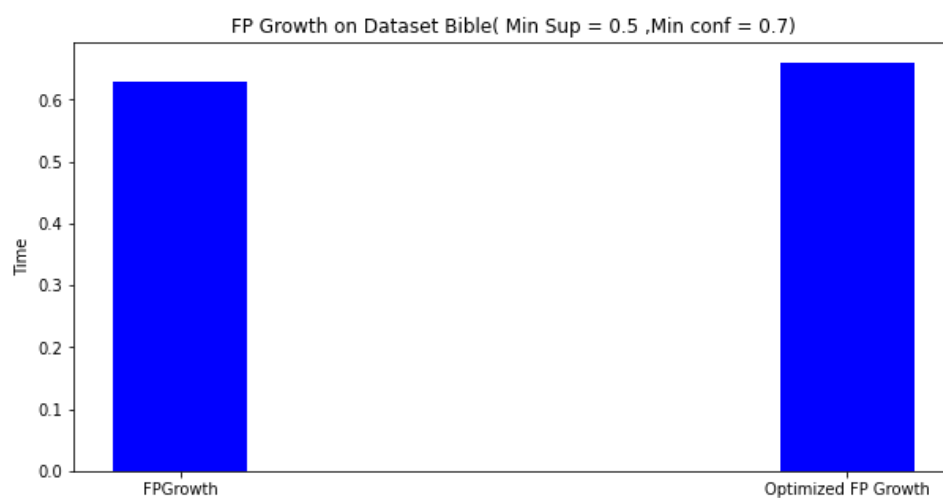
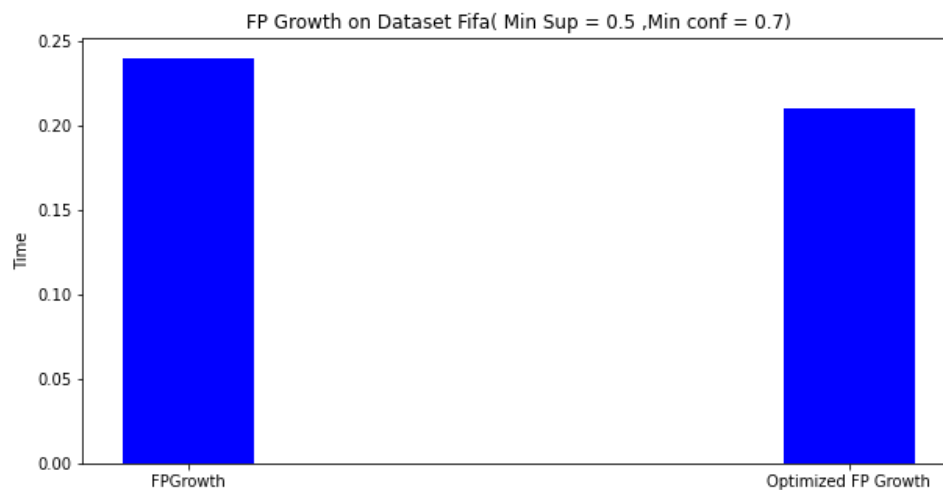
| Dataset Used | No of transactions | Time taken by FP Growth from scratch | Time taken by optimised FP Growth |
|--------------|--------------------|--------------------------------------|-----------------------------------|
| Sign | 730 | 0.34 | 0.33 |
| Kosarak | 25000 | 0.081 | 0.085 |
| Fifa | 20450 | 0.24 | 0.21 |
| Bible | 36369 | 0.63 | 0.66 |
| Leviathan | 5834 | 1.57 | 1.73 |

As as a part of task 2 we also had to use the inbuilt fp growth we have used it as well and this is a comparison for these 3 , that we obtained for the dataset sign. We have though ran the inbuilt fpgrowth for other datasets as well but we have shown the comparison just for sign dataset as this is the main dataset we have used for all the major analysis owing to its small size and as we could easily visualize the results for it as well as we have attached the screenshots for apriori.



For the other datasets we have drawn the comparison between the fp growth algorithm (basic implementation) and the optimized implementation.

The following 2 graphs show the comparison for bible and fifa dataset where as the other 2 kosarak and leviathan follow the same pattern and hence we have just represented their values in the table which clearly show us that optimized fp growth performs better !



Conclusion of Task 2:

This task helped us to have a deeper understanding of the FP Growth algorithm and how it works. By running it on different datasets for both the normal and optimized version it was very clear that the optimized algorithm performs better though in some cases it was not the case as it depends on the value of the various parameters and the data as well. Now the results that we got from here can be used for the comparisons of Task 3 as well.

TASK 3: Comparative Case Study

As we could see from task 1 and task 2 the optimized version works better for both the cases generally though there are some exceptions due to the data set size , the minimum support , the minimum confidence , the type of the transactions etc. Though one thing which we could conclude from all this is that generally optimization works whether it be Apriori or FP Growth . As in Apriori we tried 2 different optimization techniques we saw that different techniques work differently for different datasets .

We have individually analysed the tasks 1 and 2 efficiently . Now lets see the major conclusions from both of the tasks for various datasets:

For FP Growth :

| Dataset Used | Time taken by FP Growth from scratch | Time taken by optimised FP Growth |
|--------------|--------------------------------------|-----------------------------------|
| Sign | 0.34 | 0.33 |
| Kosarak | 0.081 | 0.085 |
| Fifa | 0.24 | 0.21 |
| Bible | 0.63 | 0.66 |
| Leviathan | 1.57 | 1.73 |

Conclusion : Optimized FP Growth Approach performs better !

For Apriori Algorithm :

| Dataset Used | Apriori (without optimization) | Apriori using Partitioning Method | Apriori using Transaction Reduction |
|--------------|--------------------------------|-----------------------------------|-------------------------------------|
| Sign | 0.104 | 0.080 | 0.154 |
| Kosarak | 32.09 | 33.01 | 96.59 |
| Fifa | 9.79 | 9.92 | 8.19 |
| Bible | 71.44 | 68.37 | 142.89 |
| Leviathan | 7.66 | 7.47 | 15.9 |

So from this table for Apriori we could conclude that the optimized version of Apriori performs better in some case partitioning works well in some Transaction Reduction technique works well .

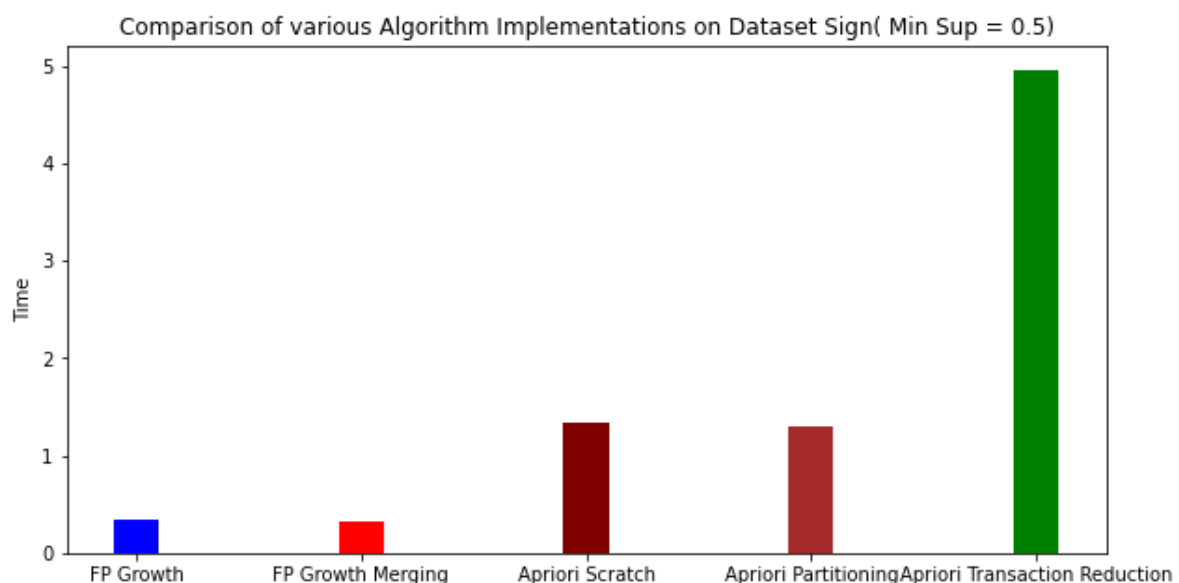
Now lets compare all these 5 models : Apriori without any optimization, Apriori using Partitioning , Apriori using Transaction Reduction , FP Growth and FP Growth with optimization.

| Dataset Used | Apriori (without optimization) | Apriori using Partitioning Method | Apriori using Transaction Reduction | Time taken by FP Growth from scratch | Time taken by optimised FP Growth |
|--------------|--------------------------------|-----------------------------------|-------------------------------------|--------------------------------------|-----------------------------------|
| Sign | 1.30 | 1.34 | 4.96 | 0.34 | 0.33 |
| Kosarak | 32.09 | 33.01 | 96.59 | 0.081 | 0.085 |
| Fifa | 9.79 | 9.92 | 8.19 | 0.24 | 0.21 |
| Bible | 71.44 | 68.37 | 142.89 | 0.63 | 0.66 |
| Leviathan | 7.66 | 7.47 | 15.9 | 1.57 | 1.73 |

From the above table the major conclusions we draw are :

- FP Growth takes less time than Apriori .
- FP Growth optimized version performs better than FP Growth Simple.
- Optimized versions of Apriori perform better than the naïve version of Apriori.

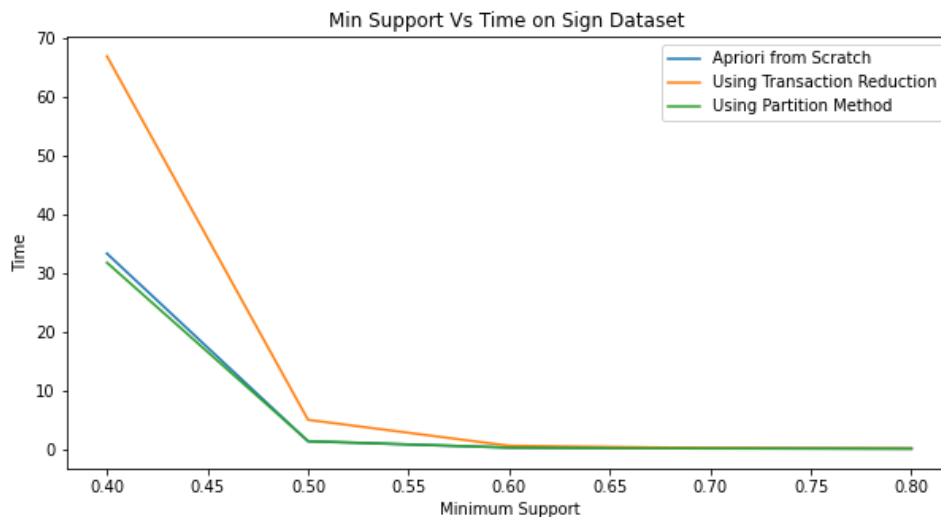
Now we will fix the value of minimum support to 0.5 and see which algorithm works well under these constraints for sign data set:



So from the above graph it is evident that FP Growth works the best (optimized version).

Now as this is the task of comparison between various parameters lets compare for support too :

| Support | 0.8 | 0.7 | 0.6 | 0.5 | 0.4 |
|-------------------------------|-------|-------|------|------|-------|
| Apriori from Scratch | 0.048 | 0.104 | 0.23 | 1.30 | 33.3 |
| Apriori Partitioning | 0.047 | 0.08 | 0.22 | 1.34 | 31.75 |
| Apriori Transaction Reduction | 0.064 | 0.159 | 0.55 | 4.96 | 66.93 |



Conclusion Drawn regarding support :

As we decrease the support more items can meet the minimum threshold parameter and thus qualify to become frequent. As the number of frequent itemsets increase the time also increases as the further steps which are dependent on this increases. Also more frequent item means that we will have to mine for more rules and thus it will require more time .Also if many items meet a particular support bar then the interesting patterns might not have that high reliability as the higher the support and the confidence the more reliable are the rules.

Conclusion of the Project

This project focussed majorly on the association mining rules algorithm and we focussed on the 2 majorly used algorithms APRIORI and FP Growth. We experimented with many datasets of different sizes and saw how different algorithms behaved differently usually FP growth performed well for the data sets under consideration .

Concluding the project we can say that Optimized FP Growth Algorithm performs the best !

-----Thank You-----