Report:- ARM Project

By:- Kartik Garg(2019101060) and Shreyash Rai(2019101096)

DATASETS:

http://www.philippe-fournier-viger.com/spmf/datasets/SIGN.txt
http://www.philippe-fournier-viger.com/spmf/datasets/MSNBC.txt
http://www.philippe-fournier-viger.com/spmf/datasets/MT745584_SPMF.txt
http://www.philippe-fournier-viger.com/spmf/dataset_proof_sequence.zip

So, i have picked 4 datasets to work with
The last link makes the browser download a zip file, in the zip file we have a ".dat" file and I have taken just the data from line 17 onwards.

Precise files that I used can be found in the link below. The link is of a github repository which will be made public past the deadline.

https://github.com/krtikgrg/AssociationPrerequisite/tree/master/data

Minimum Supports chosen are:
For proof_sequence dataset, it is kept at 10
For sign dataset, it is 450
For covid dataset, it is 480
For msnbc dataset, it is 400

NOTE: The values here are not percentages but the exact count of the transactions required to be frequent.

FP-Growth:

Firstly, I read the whole dataset and make it compatible that is removing the non valid integers like -1 and -2. Following this, I then remove duplicates from each transaction along with maintaining a counter for each item that I get in the transactions.

Then I go on to sort the items in each of the transactions in such a way that the most frequent item appears at index 0, followed by the second most frequent and then by third most frequent and so on. In the process of doing so I remove those items whose counters are below the minimum support chosen.

Above mentioned part is represented until the line 70 of the code.

From now onwards, I will work on making the FP-Tree. For that i design a node class which will provide me the instances for each of the nodes in the tree. Along with that I define an insert method which inserts my nodes into the existing tree.

So, I have the prerequisites ready and then I start inserting each of the itemsets into the tree.

Now I define a method to make pattern bases on the go, like for a path a->b->c->d for some key d. What I am gonna do is I will push the respective pattern bases for each a,b,c,d in this path itself instead of revisiting this path for each node separately. In this way I will use the merging strategy so that I can get all the pattern bases in a single DFS of the graph, which will otherwise require much more time since I will need to revisit each path for each node in it.

Now I define a generate method which will generate the conditional FP-Tree for each of the keys which is nothing but the non empty subsets of the pattern base along with the key concerned with that pattern base. I also maintain a counter for each set made which I further use to determine the final frequent itemsets.

The final frequent itemsets are then subjected to a routine so as to find the closed frequent itemsets.

Then the final closed frequent itemsets are given as an output on the screen.

Apriori:

The code flow starts from line 172 with the dataset being read, followed by the removal of duplicates from each transaction.

Then I divide the dataset into some partitions which is a predefined number set at the very beginning of the code.

Then from line 202, our actual data mining starts. We then iterate over all the partitions and mine each partition separately using apriori algo with hashing technique used.

From the results of each partition we then make a union of all these results. This union contains all the possible sets that can be frequent but to be fully sure we need to check. So we then read the database again and check which itemsets are frequent in this tentative set.

We now have all the frequent itemsets, which are then served to a routine to get final closed frequent itemsets. These itemsets are further given as an output on the screen.

In the apriori function, we first basically decide a mod value which will be used to hash the 2-frequent itemsets. Basically mod-value is the number of buckets in which i am going to divide the 2-itemsets into.

Then I read each transaction in that partition and maintain a counter for each item along with using the hash based approach for 2-itemsets. This part is represented from line 81 in the code.

So, now from the calculated counters for 1-itemsets i get the frequent ones using a modified min support. For 2-frequent itemsets I use the hash to first get the buckets that have a number

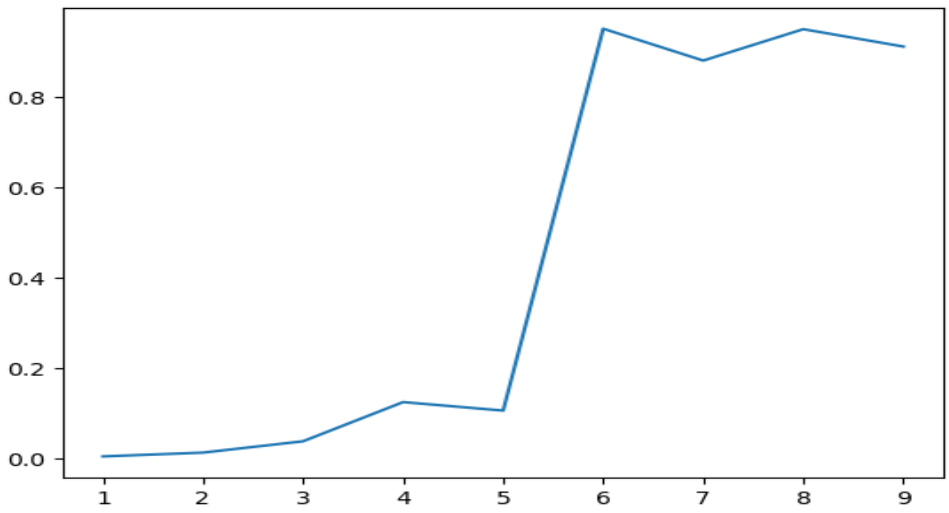
of itemsets greater than the min support and then I further check in the chosen buckets and see what all itemsets I need to incorporate.

So I now have 1-itemsets and 2-itemsets, the frequent ones. Now I go on to use the regular apriori approach to further calculate the k-frequent itemsets .

Parallelising the partitions:

I tried both sequential execution and parallel execution for all the datasets mentioned above.
And I have found that if I sequentially execute each of the partitions and make note of the time required to do all the stuff then making partitions actually slows down my execution. Whereas when parallelism is deployed I start to see the improvements in my time requirements. Like a significant improvement was observed in some cases. One more thing of note was that for small datasets like the proof dataset mentioned above was not performing well on parallelising so i think that parallelising only helps when we have a decently large dataset. Like it won't be useful to use it over small datasets(in this case proof dataset which contains approximately 35 transactions)

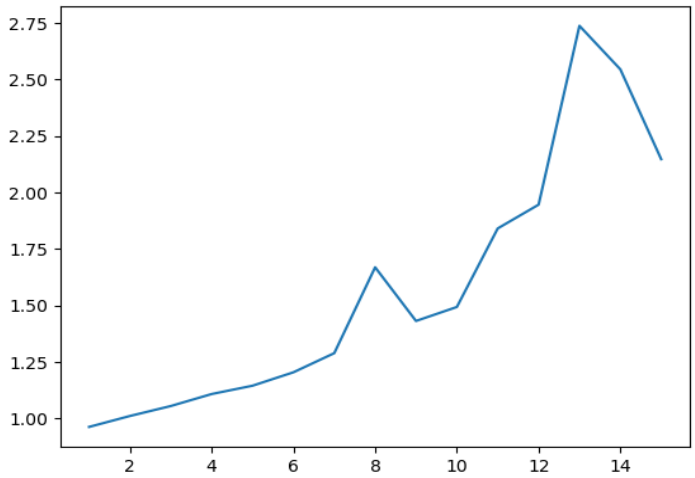
For "Proof", dataset



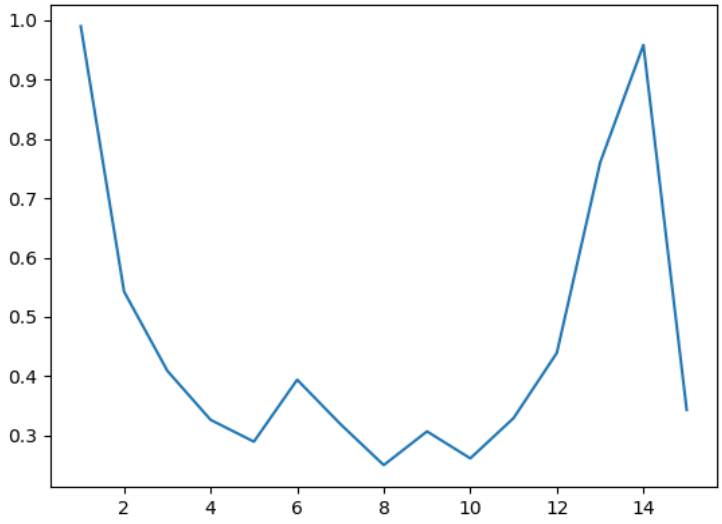
This is the graph for sequential execution, and we see our algorithm takes more time than usual. It was a small dataset so parallelising for this one didn't help and I got a similar graph.

For sign dataset:

Sequential Execution:



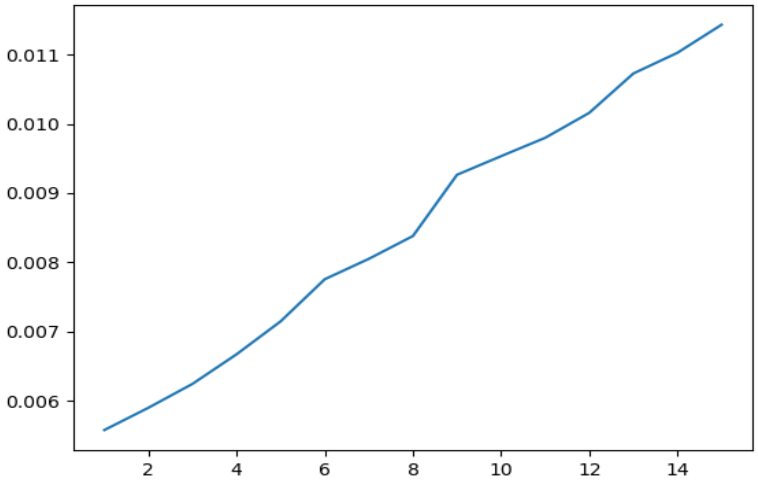
Parallel Execution:



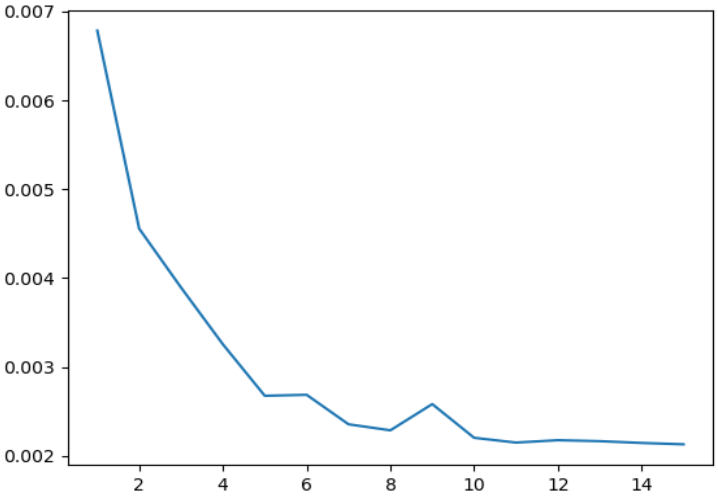
We see that on parallelising we got an improvement of approximately 5 times when we had around 8 partitions compared with a single partition.

For Covid dataset:

Sequential Execution:



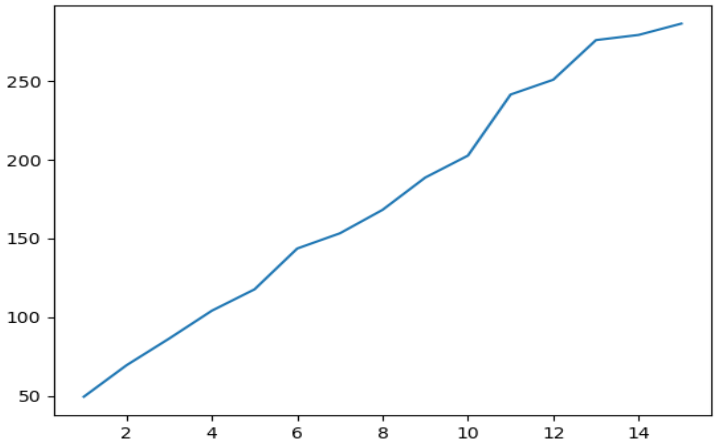
Parallel Execution:



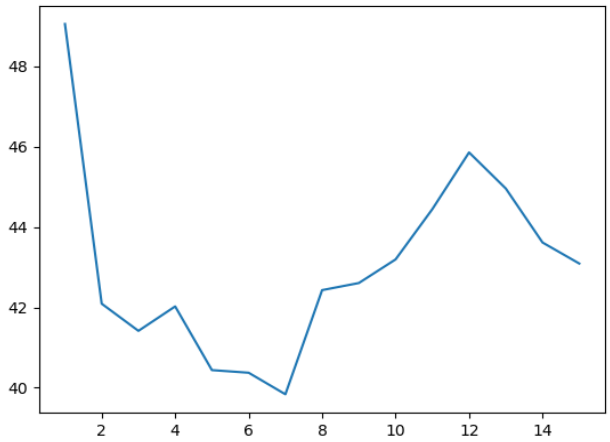
We again witnessed a 5 fold improvement on increasing the number of partitions.

For msnbc dataset:

Sequential Execution:



Parallel Execution:



We again see a similar graph with a global minima at N=7 partitions and we see that we see an improvement of approximately 10 seconds when compared to N=1 partition.

So what we observe is upto some certain level of parallelising helps us in improving the time standards but after that limit our process is burdened more by merging the individual partition results which makes our algo to consume more computational power along with more time.