Umang Srivastava (2019101090)

Avlok Gupta (2018111017)

ARM Project Report

Course : Data Analytics - I

Datasets:

1. *BMSWebView1 (Gazelle) (KDD CUP 2000)* (BMS1_spmf.txt)
2. BMSWebView2 (Gazelle) (KDD CUP 2000) (BMS2_spmf.txt)
3. MSNBC (MSNBC.txt)

Dataset	Seq. Count	Item Count	Avg. Seq. Length	Density
BMS1_spmf.txt	59601	497	2.42	287.58
BMS2_spmf.txt	77512	3340	4.62	107.22
MSNBC.txt	31790	17	13.23	24740.10

* Density = (Seq. Count)*(Avg. Seq. Length) / (Item Count)

Implementation

A. FP Growth

1. Two data structures used : t_NODE for tree nodes of FP tree and l_NODE for linked list pointing to tree node and next node.
2. Generate transactions and corresponding transaction count(tr_cnt). This is a list of all transaction in dataset. Top-level FP-tree will have all tr_cnt as 1 but conditional FP-tree can have greater value for shared branch. FP tree is also initialized with a pre_defined which stores the pattern it is conditioned on.
3. Next, we generate the list of frequency of all single items(freqA). Then, we sort and filter the list of pairs <transactions, frequency> based on frequency such that items with frequency < minimum support are eliminated.

4. Headers table of linked list is initiated for remaining items. Conditional Patterns for each item is also initiated to generate corresponding conditional FPTree. FreqA list is sorted in ascending order.
5. FPTree is generated by calling function generateFPTree() function. It adds each transaction. It generates a suffix tree of t_Node with item and and its count.
6. We call the mining method mineFrequentItemsets() recursively with itemsets. The base case is that FPTree has a single path, then all possible combination of its nodes are returned. For recursion, if there are more than one branches, conditional patterns for each frequent items are generated.
7. As conditional patterns(prefix of each tree node) are generated, the conditional FP Tree is mined beginning from the step 2 again.

B. Apriori

1. For **partitioning**, Find frequent 1-itemsets and create candidate itemset for size k+1 given k frequent itemsets.
2. Check count of these candidates and select as (k+1) - itemsets.
3. Repeat the steps again until (k+1) - itemsets return empty set.
4. For **Hash mapping**, when we get frequent 1-itemsets we also get the frequent 2-itemsets by putting the 2-itemsets in hash buckets(Python handles directly for tuples) and selecting the 2-itemsets which have minimum support to form a set of 2-frequent itemsets.

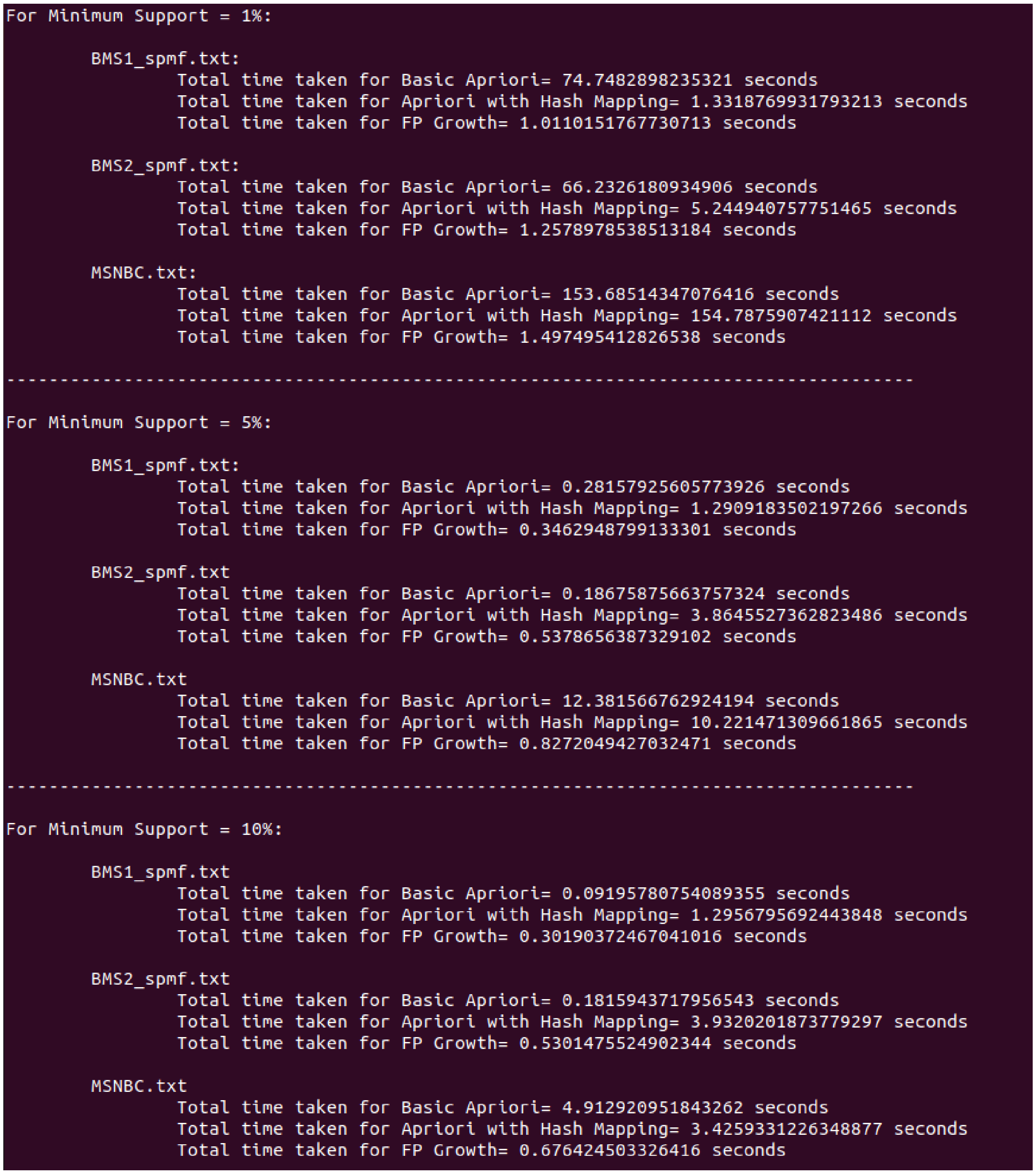
Execution

```
with open('MSNBC.txt', 'r') as file: # enter the file name here
      D = file.readlines()
MINSUPPORT = int(0.1*ln) # enter the minimum support value
```

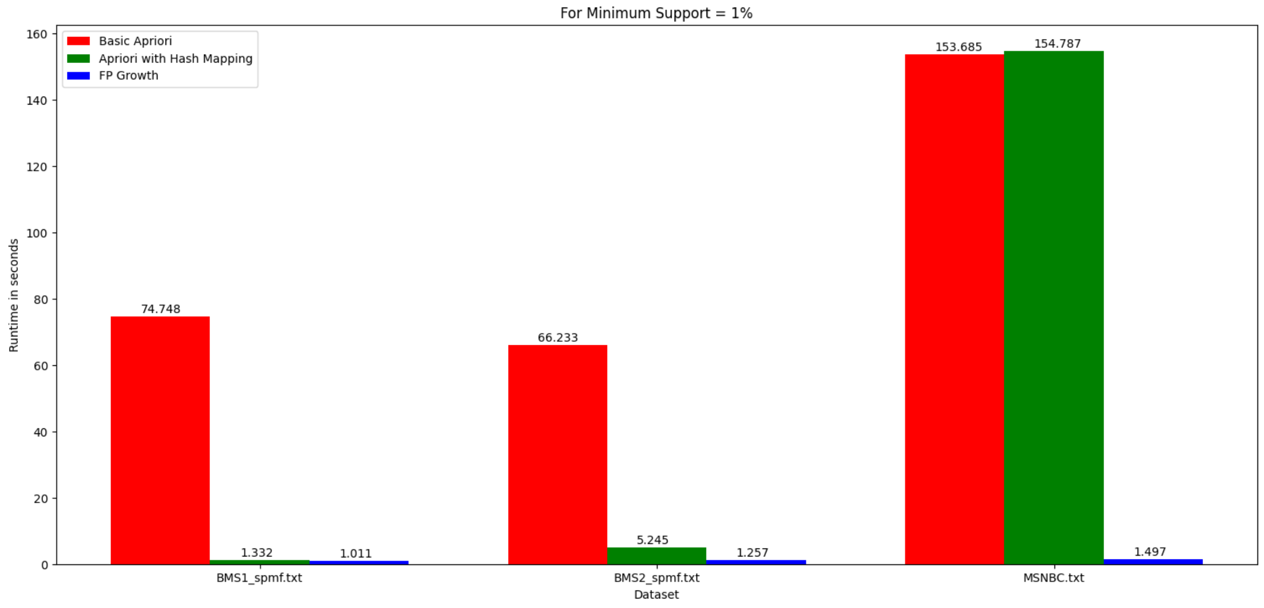
Then to execute files,

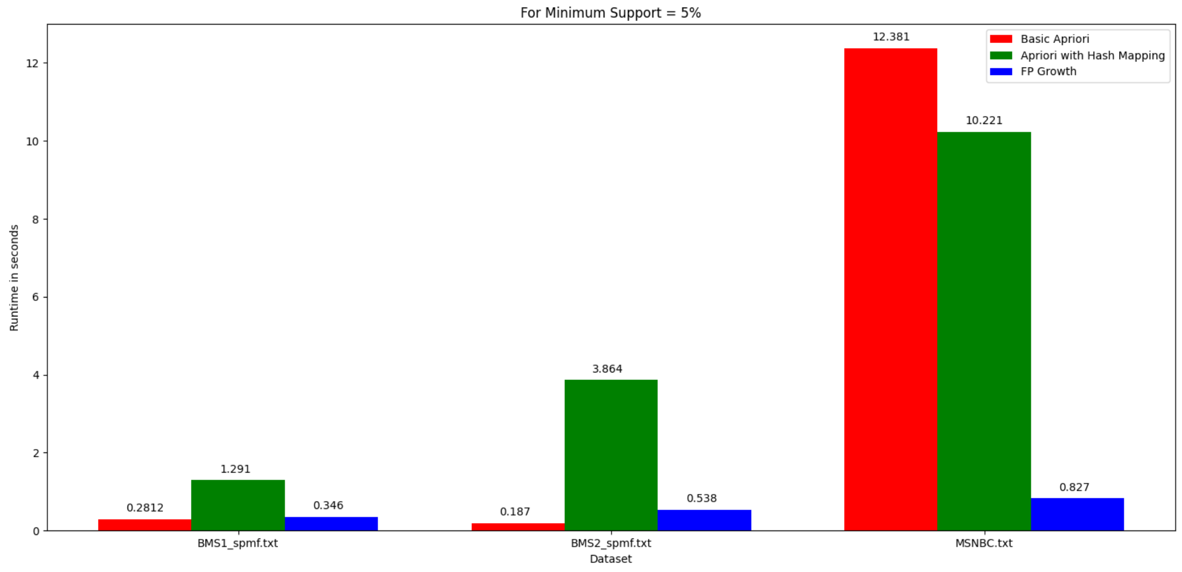
```
$ python3 2019101090_2018111017_apriori.py
$ python3 2019101090_2018111017_fpg.py
```

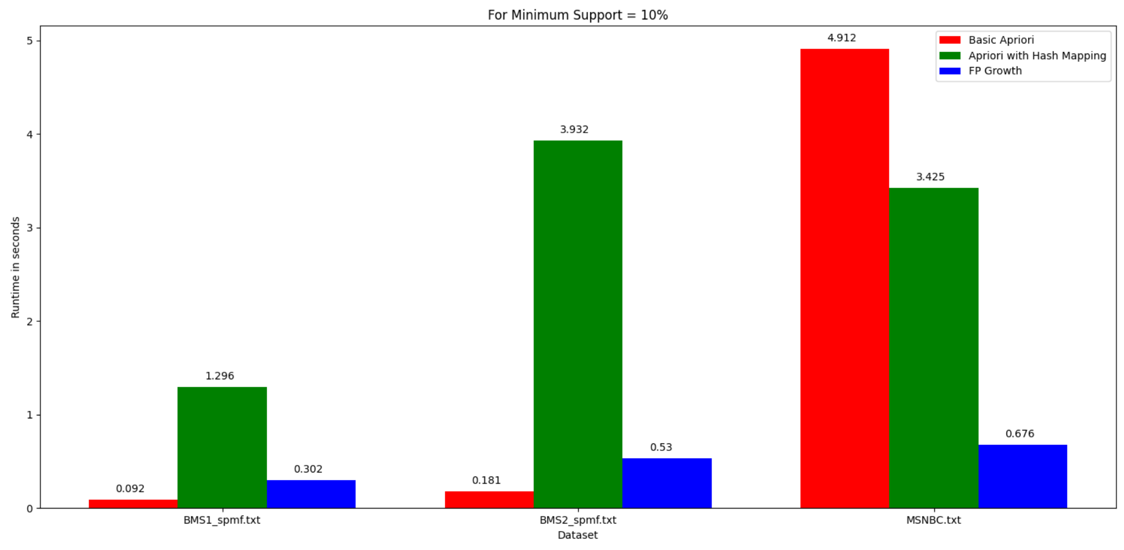
Performance Comparison



```
For Minimum Support = 1%:

        BMS1_spmf.txt:
                Total time taken for Basic Apriori= 74.7482898235321 seconds
                Total time taken for Apriori with Hash Mapping= 1.3318769931793213 seconds
                Total time taken for FP Growth= 1.0110151767730713 seconds

        BMS2_spmf.txt:
                Total time taken for Basic Apriori= 66.2326180934906 seconds
                Total time taken for Apriori with Hash Mapping= 5.244940757751465 seconds
                Total time taken for FP Growth= 1.2578978538513184 seconds

        MSNBC.txt:
                Total time taken for Basic Apriori= 153.68514347076416 seconds
                Total time taken for Apriori with Hash Mapping= 154.7875907421112 seconds
                Total time taken for FP Growth= 1.497495412826538 seconds

------------------------------------------------------------------------------------

For Minimum Support = 5%:

        BMS1_spmf.txt:
                Total time taken for Basic Apriori= 0.28157925605773926 seconds
                Total time taken for Apriori with Hash Mapping= 1.2909183502197266 seconds
                Total time taken for FP Growth= 0.3462948799133301 seconds

        BMS2_spmf.txt
                Total time taken for Basic Apriori= 0.18675875663757324 seconds
                Total time taken for Apriori with Hash Mapping= 3.8645527362823486 seconds
                Total time taken for FP Growth= 0.5378656387329102 seconds

        MSNBC.txt
                Total time taken for Basic Apriori= 12.381566762924194 seconds
                Total time taken for Apriori with Hash Mapping= 10.221471309661865 seconds
                Total time taken for FP Growth= 0.8272049427032471 seconds

------------------------------------------------------------------------------------

For Minimum Support = 10%:

        BMS1_spmf.txt
                Total time taken for Basic Apriori= 0.09195780754089355 seconds
                Total time taken for Apriori with Hash Mapping= 1.2956795692443848 seconds
                Total time taken for FP Growth= 0.30190372467041016 seconds

        BMS2_spmf.txt
                Total time taken for Basic Apriori= 0.1815943717956543 seconds
                Total time taken for Apriori with Hash Mapping= 3.9320201873779297 seconds
                Total time taken for FP Growth= 0.5301475524902344 seconds

        MSNBC.txt
                Total time taken for Basic Apriori= 4.912920951843262 seconds
                Total time taken for Apriori with Hash Mapping= 3.4259331226348877 seconds
                Total time taken for FP Growth= 0.676424503326416 seconds
```


For Minimum Support = 1%
Basic Apriori
Apriori with Hash Mapping
FP Growth
Runtime in seconds
Dataset
BMS1_spmf.txt
BMS2_spmf.txt
MSNBC.txt
74.748
1.332
1.011
66.233
5.245
1.257
153.685
154.787
1.497
0
20
40
60
80
100
120
140
160


For Minimum Support = 5%
Basic Apriori
Apriori with Hash Mapping
FP Growth
Runtime in seconds
Dataset
BMS1_spmf.txt
BMS2_spmf.txt
MSNBC.txt
0.2812
1.291
0.346
0.187
3.864
0.538
12.381
10.221
0.827
0
2
4
6
8
10
12



Observations

1. For higher value of minimum support, i.e., lesser number of frequent itemsets above threshold, partitioning apriori and FP growth work well but hashmapping optimisation works worse as complexity of hasmapping is more.
2. For BMS datasets, for higher value of minimum support, FP growth is slower compared to Apriori. This is most probably because of recursion that occurs in FP growth. Partitioning Apriori performs better because it works on candidate generations and number of candidates is much less for higher minimum support, so lesser candidates for k+1 size.
3. For BMS datasets, for less value of minimum support, i.e., more number of frequent itemsets, FP growth works exponentially better than partitioning apriori as there are large number of candidates generated at each step of apriori, but is comparable to hashmapping optimisation.
4. Performance of Apriori depends on the dataset size and number of individual items. It is much more dependent on the number of individual dataset items as it is required for pruning step which is major problem. The dataset size is used in starting, during scan so it is not a big deal.
5. For higher value of minimum support, size of dataset decides the runtime of algorithm. The number of candidates will be much less so the pruning step will not take much time therefore MSNBC initially takes more time.

6. With decreasing value of minimum support value, importance of pruning time increase so number of candidates become important. So, MSNBC dataset uses less time.
7. For BMS1 and BMS2 datasets, for lower value of minimum support, hashmapping takes lesser time than partitoning but this is not true for MSNBC dataset. MSNBC dataset has much larger transaction length and lesser number of items so it is dense.
8. The hashmapping algorithm is implemented for 2 itemsets so while it reduces time taken for 2 itemsets, it does not affect the performance of other k itemsets. But again, despite of this, there is a significant improvement in runtime of code for all datasets.