Implementation of SON Algorithm: Frequent Pattern mining

1.Introduction

The Frequent Itemset Mining problem consists of finding sets of frequent co-occurring elements in a given corpus collection of elements. In the following assignment the two pass Algorithm of Savasere, Omiecinski and Navathe is implemented. The implementation is written in scala and runs on the Apache Spark framework.

2.Implementation

In order to implement the SON algorithm we have to implement two different phases. In the first phase the frequent itemsets in a subsets of the file are found and then they summed to determine the candidate itemsets. In the second phase based on the candidate itemsets we specify which are the frequent itemsets are return them with their frequency.

In more detail, for the first phase we used the Apriori Algorithm provided by the teaching assistant. There was no need to split the file in subsets nor to sum the results of the subsets because spark does the splitting dynamically and sums the results without the need of extra code from the programmer (the mapPartition splits the RDD equally to the workers).

In the second phase we create sets containing the items of the whole file. Then initialize a data structure containing the candidate itemsets and a frequency initialized to zero for each of them. After that we compare each candidate itemset with all the itemsets in the initial file, if the candidate itemset is contained in an itemset of the initial file we increment its counter by one. Finally we can set our support threshold for the second phase in order to retrieve the frequent itemsets with their frequency. Again the RDD splitting is provided by the build in functions of spark.

3.Evaluation

In order to evaluate our algorithm we do two different kind of experiments. In the first experiment our goal is to measure the time needed for our algorithm to complete based on different support thresholds. In the second experiment we compare our implementation with the FP-Growth implementation of MLlib.

Our test machine is based on an Ubuntu 16.04 virtual machine with 2 cores running at 2.00 Ghz and 4 Gb of RAM. Because of the limited resources the dataset is cropped by 2/3 in order for the experiments to complete without crashing the machine.

THRESHOLD	**0.35**	**0.40**	**0.50**
FP-GROWTH ITEMS	57	41	21
OUR ALGORITHM ITEMS	57	41	21

As we measured from our experiments the support threshold does not affect the time needed for our algorithm to complete. In all the experiments the algorithm needed a median of about 10 minutes to complete. These are very slow times compared to the MLlib implementation of FPGrowth whose only need some seconds to complete.

In the other hand as presented by the table above, our algorithm had 100% precision compared again with the MLlib implementation of FPGrowth.

4.Future work

As future work we intend to make optimizations to the code in order to have similar execution time to the MLlib FP-Growth.

5.Conclusion

In this work we presented an implementation of parallel SON algorithm and shown that it retains 100% accuracy. We described in detail the steps to implement the algorithm. Finally we provide the code for everyone to use and experiment.