Association Rule Mining Project

For extra details on optmisations please refer to the optimisations pdf

Only one team member has to submit

Deadline : 9 Nov 11:55 PM

1 FP Growth

Implement the FP Growth algorithm to generate closed frequent itemsets at runtime with the merging strategy optimisation as given below:

Merging Strategy : Push right the branches that have been mined for a particular item say p, i.e. push these branches to the remaining branch(es) of the FP-tree as it can be time and space consuming to generate numerous conditional pattern bases

2 Apriori

Implement the Apriori algorithm to generate closed frequent itemsets at runtime using both of the optimisations strategies given below:

- **Hash Based Technique**: When scanning each transaction in the database to generate the frequent 1-itemsets , we can generate all the 2-itemsets for each transaction, hash (i.e., map) them into the different buckets of a hash table structure, and increase the corresponding bucket counts. A 2-itemset with a corresponding bucket count in the hash table that is below the support threshold cannot be frequent and thus should be removed from the candidate set. Refer to figure 6.5 of Han, Kimber, Pei for better understanding .

  Note: This can be generalised to other k's than just 2, but for the sake of simplicity, let's just stick to 2 for this project .

- **Partitioning** : Distributed and Parallel algorithms for apriori based frequent itemset mining obey a rule : 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 must occur as a frequent itemset in at least one of those n partitions. Now the

algorithm can be on each of those partitions in parallel and all the local frequent itemsets generated can be checked for being globally frequent by a scan of the database again.

Just try out what can be a suitable partitioning, and perform sequentially the apriori generation in each of them, look at what was the maximum time needed among the individual runs to get an idea of how much parallelising this would have helped.

Note: Make note of the possible improvement in performance if the algorithm were parallelised by following the last paragraph above .

3 Tasks and Submission

- Keep in mind the fact that for both of the implementations you have to find ***closed frequent itemsets*** in runtime.

  An itemset X is closed in a data set D if there exists no proper super-itemset Y such that Y has the same support count as X in D. An itemset X is a closed frequent itemset in set D if X is both closed and frequent in D. It has to be a running algorithm that takes a lower time to generate the closed frequents not generating all frequent itemsets and then deciding on them to get closed frequent itemsets.

- **Allowed languages** : C++, Java , Python, note that whichever language you choose you are not allowed to use any specific libraries for the direct implementation of the algorithms, however, standard library is allowed.

- **Dataset**: SPMF Open Dataset Library. Please put due reference for the same in your report. Try to stick to Real-life datasets section. The library contains various datasets ranging from small to large. So use accordingly as per the need of your algorithm.

- **Format of input** : Each line corresponds to a transaction, ending of line deonted by a -2. In a line the items are separated by a -1. Mostly datasets have numeric format only, so use that to your convenience.

- **Naming convention**: <RollNumber1> <RollNumber2> ARM.zip, e.g., 20171117 20171185 ARM.zip. Submit a zip file containing your code files and the report pdf for the tasks mentioned in the project.
  Note: Python notebooks are not allowed.

- There should be 2 code files and the report . The codes should be named as <RollNumber1> <RollNumber2> apriori.cpp (or java or py extension required) for the apriori code and <RollNumber1> <RollNumber2> fpg.cpp for fp-growth.

- Don't plagiarise. ofc!