Association Rule Mining Project

Optimising and Experimenting with Classical Algorithms

Deadline : 23rd October 2020, 11:55 PM

1 Apriori Algorithm

As per the standard technique, in order to find L_k, the frequent itemsets of size k from L_{k-1}, the frequent itemsets of size $k-1$, a two-step process is followed, consisting of join and prune actions for the candidate generation, i.e., to find C_k and hence starting from $1-$frequent itemsets, we get to reach the k-frequent itemsets . Now a brute force approach is somewhat expensive and call for certain optimisations .

- **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 run of FIM 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 . **Note** : You do not need to significantly parallelise the algorithm . 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 .

- **Transaction Reduction** : 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 . This is an easy task and hence recommended as a to be tried optimisation.

- **Hash based Technique** : When scanning each transaction in the database to generate the frequent 1-itemsets, L_1 , 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 clear 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 .

Suggestion : Please go through section 6.2.1 and 6.2.3 of Han, Kimber, Pei to get better understanding of the algorithm and its optimisations .

Tasks :

- Implement the entire apriori algorithm in any of the allowed languages using any 2 of the above strategies . Run the algorithm on few datasets from the given dataset library .
- State your partitioning strategy or the hash mapping whichever of the 2 you choose respectively.
- Compare the performance of your new implementation with the standard version. Do not worry the optimisation strategies are generally to be applied on very large databases, so it might not show sufficient improvement on datasets we use, so try to optimise as far as possible and state the improvements you have got .

2 FP-growth

Many techniques have been proposed to further improve the performance of frequent itemset mining algorithms. Taking FP-tree– based frequent pattern growth algorithms (e.g., FP-growth) as an example, implement one of the following optimization techniques.

- **Top-Down Projection Technique** : The conventional approach involves using a bottom-up projection technique to generate conditional pattern bases . However, one can develop a top-down projection technique, that is, project onto the suffix path of an item p in the generation of a conditional pattern base. This may or may not improve the performance and hence in improved implementations, a case handling is done to use either of the projection techniques .
- **Merging strategy** : It is time and space consuming to generate numerous conditional pattern bases during pattern-growth mining. An interesting alternative is to push right the branches that have been mined for a particular item p, that is, to push them to the remaining branch(es) of the FP-tree. This is done so that fewer conditional pattern bases have to be generated and additional sharing can be explored when mining the remaining FP-tree branches.

Suggestion : Please go through section 6.2.4 of Han, Kimber, Pei to get better understanding of the algorithm and its optimisations .

Tasks :

- Implement the entire FP-growth algorithm in any of the allowed languages using any 1 of the above strategies . Run the algorithm on few datasets from the given dataset library .
- Explain clearly how have you used any of the optimisation in your algorithm.
- Compare the performance of your new implementation with the unoptimised version.

Bonus :
Since the number of itemsets in general are huge in number, few pattern mining algorithms stick to just finding closed frequent itemsets or maximal frequent itemsets . 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. An itemset X is a maximal frequent itemset (or max-itemset) in a data set D if X is frequent, and there exists no super-itemset Y such that X is a proper subset of Y and Y is frequent in D. Can you modify the FP-growth algorithm to generate only closed frequent itemsets or maximal frequent itemsets ? We do not expect generating all frequent itemsets and then deciding on them to get closed frequent or maximal frequent ones, we want a running algorithm that takes lower time to generate the closed frequent or the maximal frequent itemsets than the conventional FP-growth algorithm to generate the frequent itemsets .

3 Comparative Case Study

Compare the performance of each algorithm with various kinds of data sets. In your report, analyze the situations (e.g., data size, data distribution, minimal support threshold setting, size of desired frequent itemsets) where one algorithm may perform better than the others, and try to reason argumentatively .

4 Necessities

- 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 library : 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 .

- General Format of dataset : 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 .
- Tasks in section 1,2 and 3 are compulsory. Only the closed frequent itemset mining in section 2 is considered for bonus. Implementing more than the required optimisations wherever there is a choice fetches no bonus .

5 Submission

- 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 . Sorry, **python notebooks are not allowed** this time .
- 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 .
- Submission from one member is sufficient .
- Please refrain from any kind of plagiarism, else cases involved will be dealt with strict penalty .