Question #1 Report

Implementation and results of common association rule mining algorithms

Abhinav Khanna(2017CSB1061)

CS524: Data mining

INTRODUCTION

In the area of association rule mining, 3 major algorithms were discussed: Apriori, FP tree and ECLAT. The purpose of this project is to implement these algorithms on various datasets, analyze their space and time complexity, and conclude which algorithm is better for which dataset. The scope of this project is limited by the hardware capabilities, such as RAM, number of cores/processors, and CPU clock cycles, availability and usage of GPUs et al.

THEORETICAL FOUNDATION

Of the three algorithms implemented, Apriori and FP tree deal with horizontal datasets, whereas ECLAT deals with vertical datasets.

Apriori is the most naive and closest to a brute force algorithm of the three. Without going into the full details of the algorithm, it is sufficient to say that it sequentially generates itemsets at each level i.e. firstly all frequent 1-itemsets, then all frequent 2-itemsets, and so on. This generation is done by combining 2 itemsets of the same size, which share the same prefix, i.e. to generate a (k+1)-itemset, any two k-itemsets which share a prefix of (k-1) length.

FP tree works by first sorting items within a transaction in decreasing order of frequency, and getting rid of infrequent items altogether. Then, the full FP tree is built. Also, a list of pointers is maintained, in which each pointer acts as the head of a linked list connecting all other nodes containing the same item. Conditional FP-trees are generated for each itemset. If an itemset is frequent enough, other supersets of this itemset are checked. This process goes on recursively. At any point, if an itemset is not frequent, it is discarded, and none of its supersets are generated(as can be verified by the support monotonicity property).

ECLAT works on vertical datasets. The simple idea behind ECLAT is that we first count all frequent 1-itemsets, simply by counting the number of transactions associated with it. Then all other frequent itemsets are generated, following a similar pattern as that in Apriori. There exists a variation of ECLAT, which is dECLAT, which creates diffsets, instead of tidsets. However, in my implementation, I have used ECLAT(i.e. tidsets).

IMPLEMENTATION

- ### Apriori Algorithm

  Data Preparation

  This step involves reading the dataset file, one transaction at a time. I used a dictionary, which acted as a map from an item to it's support. Once all the transactions were read, the items whose support was less than the minimum support, were removed from the dictionary. This gave me the set

of all frequent 1-itemsets.

Mining frequent k-itemsets

For generating 2-itemsets, 3-itemsets and so on, I adopted the method of support counting using hashing. Thus, first all the candidate itemsets were generated at a level k by following the principle of combining 2 itemsets of length (k-1) with the same prefix of length (k-2) .

Then,to perform support counting using hashing, for each transaction, all subsets of length k were generated using python's itertools library. Then, of all these subsets, whichever were present in the set of candidate itemsets, the support of those candidate itemsets was incremented. This process was repeated for all transactions. Then, the candidate itemsets whose support count was less than minimum support were deleted.

The value of k is then incremented, and all the steps mentioned previously are repeated until we get an empty set, i.e. no frequent itemset of that size k can be generated.

- ## FP tree growth algorithm

Data Preparation

This algorithm requires that the items in a transaction be sorted in decreasing order of support. To do this, I have first read the files and stored the frequent items in a dictionary. Then I have again read the file, one transaction at a time, and sorted the items in the transaction accordingly. Then I have written these transactions to a new file, whose name is "<original dataset name> " + "_sorted". E.g, after reading the file "retail.dat", the new file containing sorted transactions is stored in "retail_sorted.txt".

Frequent Itemset mining

Firstly, an FP tree is constructed, following the steps discussed in the lectures. A node in the tree contains the following parameters: support(integer denoting actual support), key(denoting which item the node denotes), children(list of nodes), neighbor(pointer to the next node containing the same item), conditional_support(initially 0, incremented

while building conditional FP tree). Apart from this, a pointer_map is also maintained, which is a list of pointers, where each pointer points to a different item in the FP tree, essentially serving as the head of the linked list of neighbors containing the same items.

Once the FP tree is built, conditional FP trees are built. Starting from any 1-itemset, it's conditional FP tree is built. If the conditional support is greater than the minimum support, then all possible 2-itemsets are tried recursively. This process goes on recursively, until the k-itemset is not frequent. It is important to note that at each level of the recursive call, a copy of the existing conditional FP tree has to be stored, as Python by default passes the reference of the tree root.

- ## ECLAT Algorithm

Data Preparation

Firstly, all the infrequent items are removed. But in ECLAT, data preprocessing step also involves converting the horizontal transactions, given in all the datasets, and converting them to vertical transactions. I created a dictionary which maps each item to a list of transactions that the item was present in.

Mining frequent k-itemsets

Once we have vertical transactions, the mining part is quite similar to that of Apriori. At level k, we take intersection of the tidsets of 2 itemsets at (k-1)th level. If the number of transactions(i.e. support) is greater than minimum support, the itemset is added to the frequent itemset list. At any level, if no new itemset is frequent, then the process is terminated.

TIME AND SPACE COMPLEXITY RESULTS

Dataset	Apriori	FP tree	ECLAT
T10I4D100K.dat	20 min 31 KB Min_sup = 0.01	Not completed 33.8 MB Min_sup = 0.01	22 sec 40 KB Min_sup = 0.01
T40I10D100K.dat	29 min 35 KB Min_sup = 0.05	Not completed 63 MB Min_sup = 0.05	1 min 36.5 KB Min_sup = 0.05
retail.dat	50 min 627 KB Min_sup = 0.01	13 min 20 MB Min_sup = 0.01	1 sec 3.3 MB Min_sup = 0.01
groceries.csv	8 sec 33 KB Min_sup = 0.01	45 min 5.5 MB Min_sup = 0.01	0.8 sec 970 KB Min_sup = 0.01

Table 1

Note: Space mentioned is the memory occupied by the respective jupyter notebook, which is indicative of the actual space complexity.

- ### Dataset Information

Properties	T10I4D100K.dat	T40I10D100K.dat	retail.dat	groceries.csv
# of transactions	100000	100000	88162	9835
Avg. width of transaction	10.1	39.6	10.3	4.4
Maximal frequent itemset size	3	2	4	3
# of maximal frequent	1	15	6	31

itemsets				
Total # of items	871	943	16471	169

CONCLUSION

After running all 3 algorithms on various datasets, my conclusion(s) is that the Apriori algorithm is the most brute force algorithm of the three. It is also evident that FP tree growth algorithm has very high space requirements, which can be attributed to the fact that each conditional FP tree has to be stored on the stack memory. Due to the **low average width of transactions** provided in all the datasets, it turns out that **ECLAT** is the most efficient algorithm in all of the 4 datasets, as is evident from Table 1.

REFERENCES

1. Data Mining and Analysis: Fundamental Concepts and Algorithms: Textbook by Mohammed J. Zaki and Wagner Meira
2. Introduction to Data Mining: Book by Michael Steinbach, Pang-Ning Tan, and Vipin Kumar
3. Class Notes