FREQUENT PATTERN BASED DENSITY SUBSPACE CLUSTERING

Keerti Thakur, Prof. Abhishek Mathur
M. Tech, Department of IT, SATI Vidisha
Asst. Prof. Department of IT, SATI Vidisha,
keerti.thakur@gmail.com
Vidisha,(MP) India

ABSTRACT: Efficient algorithms to discover frequent patterns are crucial in data mining research. Several effective data structures, such as two-dimensional arrays, graphs, trees, and tries have been proposed to collect candidate and frequent itemsets. As the main disadvantage of FP-Growth is that it takes a lot of time to build and also needs more memory for storing the transactions. To overcome these disadvantages, in this paper we give a new association rule mining algorithm called MFP. MFP algorithm converts a transaction database to an MFP tree through scanning the transaction database only once, then prunes the tree and at last mine the tree. Because the MFP algorithm scans a transaction database one time less than the FP_growth algorithm, the MFP algorithm is more efficient under certain conditions.

Key Term: Data mining, Data clustering, Subspace clustering.

INTRODUCTION

Clustering technique, a procedure in which dataset is partitioned into clusters according to minimizing the intra cluster similarity and maximizing the inter-cluster similarity, has extensive applications. However, most existing algorithms are designed for low dimensional data, they can’t work well in high dimensional data, because in high dimensional data usually just a few dimensions are relevant to a cluster, and the data on irrelevant dimensions may produce much noise so that real clusters are hidden and can’t be discovered [14]. Another reason that many clustering algorithms struggle with high dimensional data is the curse of dimensionality [15][16]. As the number of dimensions in a dataset increases, distance measures become increasingly meaningless. Additional dimensions spread out the points until, in very high dimensions they are almost equidistance from each other.

Subspace clustering is an efficient approach to clustering high dimensional data. It has recently been developed rapidly. Subspace clustering assumes that different subspace may contain different meaningful cluster. A subspace cluster is a subset of points together with a subset of attributes, such that the cluster points project onto a small range of values in each of these attributes, and are uniformly distributed in the remaining attributes [5].

This paper proposes a subspace clustering algorithm which follows the bottom-up strategy, evaluating each dimension separately and then using only those dimensions with high density in further steps.

The rest of the paper is organized as follows, Section II describes related work. The proposed clustering technique is described in section III. The experimental results are provided in section IV. Finally, section V concludes the article.

II. RELATED WORK

Without loss of generality, the subspace clustering algorithms can be divided into two categories according to whether the grid structure is used or not. The CLIQUE [4] its successor PROCLUS, ORCLUS, which find an initial clustering in the full set of dimensions and evaluate the subspaces of each cluster, iteratively improving the results, or bottom-up approaches, such as CLIQUE, ENCLUS[9] and MAFIA, which use an APRIORI style algorithm to find dense regions in low dimensional spaces and combine them to form clusters. Many top-down algorithms require that the number of clusters and the size of the subspaces be input as parameters [8]. Bottom-up algorithms are able to find clusters of various shapes and sizes. The clusters can overlap each other. The main parameter required by many bottom-up algorithms is the density threshold. Beside of determining the number of clusters and finding high quality clusters, the other two difficult
for subspace clustering problem are how to determine the dimensionality of the subspaces and how to find out subset of attributes that are relevant for each cluster [12].

III. PROPOSED ALGORITHM

In this paper, we devise the DENGRID algorithm, standing for Density Grid subspace clustering , to discover the clusters with our proposed density thresholds. We model the problem of density conscious subspace clustering to a similar problem of frequent itemset mining [6][7]. We note that by regarding the intervals in all dimensions as a set of unique items in frequent itemset mining problem, any k-dimensional unit can be regarded as a k-itemset, i.e., an itemset of cardinality k. Thus, to identify the dense units satisfying the density thresholds in subspace clustering is similar to mine the frequent itemsets satisfying the minimum support in frequent itemset mining. However, our proposed density subspace clustering problem is significantly different from the frequent itemset problem since different density thresholds are utilized to discover the dense units in different subspace cardinalities, and thus, the frequent itemset mining techniques cannot be adopted here to discover the clusters.

In DENGRID, the dense unit discovery is performed by utilizing a novel data structure DMFP-tree (Density MFP-tree)[4][2], which is constructed on the data set to store the complete information of the dense units.

From the DMFP tree, we compute the lower bounds and upper bounds of the unit counts for accelerating the dense unit discovery, and these information are utilized in a divide and conquer scheme to mine the dense units. Proposed algorithm having two sub phases . In the first phase we consider to utilize the nodes satisfying the thresholds to discover the dense units, these nodes we called as ‘Generation of Inherent Dense unit’. In the second phase we consider, for the nodes whose node counts do not exceed Tk, We take the nodes carrying the same one dimensional unit together into consideration in discovering the k-dimensional dense units, these nodes we called as ‘Generation of Acquired Dense unit’.

In the previous paper [1] we use DFP tree (density FP tree ) for finding the dense point which is used later for finding the dense region. FP tree is highly efficient for finding the dense point and it take two scan of data point for finding the dense point. The first time, an L table is created. The second time, an FP tree is built. The cost of scanning a transaction database is usually very high. It take lots of time for generation of dense point .Therefore we use a new algorithm for finding the dense point which is more efficient than FP tree known as MFP tree

To control the increase in CPU cost which is incurred by large number of possible items combinations.

To avoid disk thrashing caused by the lack of main. Also, to control the increase in I/O cost, which is incurred by approaches such as database projection .

The Principle of the MFP algorithm

The mining process of MFP Algorithm can be divided into three steps. The first one is to construct MFP tree; the next is to prune MFP tree; and the last step is to mine MFP tree.

<table>
<thead>
<tr>
<th>TID</th>
<th>List of item_ID’s</th>
</tr>
</thead>
<tbody>
<tr>
<td>T001</td>
<td>I1,I2,I5</td>
</tr>
<tr>
<td>T002</td>
<td>I2,I4</td>
</tr>
<tr>
<td>T003</td>
<td>I2,I3</td>
</tr>
<tr>
<td>T004</td>
<td>I1,I2,I4</td>
</tr>
<tr>
<td>T005</td>
<td>I1,I3</td>
</tr>
<tr>
<td>T006</td>
<td>I1,I2,I3,I5</td>
</tr>
<tr>
<td>T007</td>
<td>I1,I2,I3</td>
</tr>
</tbody>
</table>

Figure 1. Transaction database

Constructing MFP tree

The root of an MFP tree labeled with ”null” is created first. Then, the first transaction record is picked up from the transaction database D being mined and inserted into the MFP tree in the following way.

The first item of the record is taken out. If the root of the tree has a sub-node whose label is the same as the label of item just taken out, we add 1 to the node’s count field. Otherwise, we create a new sub node of the root with the same label as the item. After that, we use the sub node as a sub tree’s root, take out the next item from the transaction record and insert it into the sub tree in the same way. The process repeats till all the items in the transaction record are put into the MFP tree. In this way, we can insert all the transaction records in D into the MFP tree and complete the process of converting the transaction database D to the MFP tree. In order to mine an MFP tree easier, the first column of the table is filled with
the item IDs which appear in the transaction data base. Then the corresponding chains of node links are built so that the occurrences of one item in the MFP tree are linked together. After that, each item support count is calculated through the node links and the results are put into the second column of table TL. MFP tree shown in Figure 2. The number in the bracket on the right side of a node label is the node’s count value.

Figure 2. The MFP tree from D

(1) Pruning MFP tree

In order to facilitate the MFP tree mining, we prune the tree according to the value of minimum support count given beforehand. Suppose that the minimum support count is 3 (i.e. min_sup 3/9 = 33%). Those items whose support counts are less than 3 are deleted from the table. All the tree nodes linked with the deleted item in the table are deleted also. If a deleted node has a sub tree linked, have its sub tree link to its mother node. In the example shown in Figure 3, I₄ and I₅ in TL are deleted because their support count is less than 3. Their occurrences in MFP tree are deleted afterward. After pruned, MFP tree is shown in Figure 3.

We take out the first leaf node I₃ from the left of MFP tree. All of combinations are made up from the nodes (except the root) on the branch to which the leaf connects. After that, we send all combinations into set CF, if there are not the same combination labels in set CF. Otherwise, we merge the two combinations with the same label by adding together the two count values. When all Combinations of the nodes on the branch are sent into set CF, we modify the nodes on the branch in following way. The leaf node’s count value is subtracted from every node’s count value (including the leaf node’s) on the branch. If a node’s count value on the branch is zero after subtracting, the node will be deleted. Then we take out the new first leaf node again from the left of MFP tree and process it in same way.

On finishing processing the MFP tree, we delete those combinations in CF, whose count value is less than minimum support count given. After that, all the combinations left in set CF are candidate frequent patterns we want. We can create all association rules from the combination in set CF, and use min conf given to find strong association rules. An example of mining MFP tree shown in Figure 4 is given below. We find leaf nodes I₃ which is the first leaf node of MFP tree from the left, and get the branch node I₃, I₁.
We combine the two nodes to get the combination of \(\{I_3, I_1\}\)\(1\). Because set CF is empty at this moment, the combination enters CF directly. Then we modify the nodes on the branch, by subtracting the number of the \(I_3\).\(\text{count}\) from the node.\(\text{counts}\) on the branch. After subtracting, the \(I_3\).\(\text{count}\) equals 0 and \(I_1\).\(\text{count}\) equals 4. Because \(I_3\).\(\text{count}\) is zero, the node is deleted from MFP tree. Now we take out the new first leaf node of the tree from the left, which is \(I_3\) and is the sub node of \(I_2\), and repeat the above process. This time, we get the branch from \(I_3\) to the root. The branch is composed of \(I_3\), \(I_2\), and \(I_1\). We can get all the combinations from the three nodes \(\{I_3,I_2,I_1\}\)\(2\), \(\{I_3,I_2\}\)\(2\), \(\{I_3,I_1\}\)\(2\), \(\{I_2,I_1\}\)\(2\) and put them into set CF. Then, we modify the count fields of the three nodes on the branch. At this time, we subtract the value of \(I_3\).\(\text{count}\) from the nodes\(\text{counts}\) on the branch. Because the value of \(I_3\).\(\text{count}\) is 0, the node is deleted and the node \(I_2\) becomes the first leaf node of MFP tree. We continue the process in the same way till the MFP tree is empty. After finishing processing MFP tree, we get set CF as below.

\[
\text{CF} = \{\{I_3,I_1\}\}(4), \{I_3,I_2,I_1\}\(2\), \{I_3,I_2\}\(4\), \{I_3,I_1\}\(2\),\{I_2,I_1\}\(4\) \}
\]

We delete those combinations in set CF, whose support count is less than the minimum support count given beforehand, (minimum support count = 3 in this example) and obtain set CF as below.

\[
\text{CF} = \{\{I_3,I_1\}\}(4), \{I_2,I_1\}\(4\), \{I_3,I_2\}\(4\) \}
\]

### IV . EXPERIMENT RESULTS

In this section, we compare the Dengrid approach for finding clusters with the algorithm of [1]. The proposed approach Den Grid and Previous approach [1] is executed on real dataset. The algorithm of the proposed approach is implemented in MATLAB version 7.6.0.324 (2008a)., we perform experiments on thyroid disease database contains 18,152 thyroid diagnoses with six numerical attributes from UCI machine learning repository. Now we show the bar graph of execution time and memory used between Dencos approach and Dengrid approach and finally we give the table that compare execution time and memory space differences.

As shown in bar graph represent the execution time and memory used in the process (1) is Dencos approach and (2) is Dengrid approach.

![Fig 4 Execution time comparison](image)

![Fig 5 Memory comparison](image)

Here we give the tabular comparison of execution time and memory used between Dencos and Dengrid approach at two different instances.
It is clear from the result shown above that our proposed Dengrid approach is faster and efficient than previous Dencos approach.

V. CONCLUSION

From the experimental result on the dataset shows that the Dengrid algorithm serves a good subspace clustering algorithm which identifies cluster in all subspaces quickly. It completes the mining process through scanning the transaction database only one time. It is concluded from the results that the algorithm is significantly outperforms previous works, thus demonstrating its practicability for subspace clustering.

REFERENCES


[6] J. Han, J. Pei, Y. W. Yin, R. Y. M. “Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach. Data Mining and Knowledge Discovery, 8(2004), 53–87.


[11] HKU 738002E from Hong Kong RGC, Frequent-Pattern based Iterative Projected Clustering Man Lung Yiu and Nikos Mamoulis Department of Computer Science University of Hong Kong Pokfulam.


