Journal of University of Science and Technology Beijing
Vol.7(2000), No.4, p.318

Information

A New Clustering Algorithm for Categorical Attributes

Songfeng Lu, Zhengding Lu

College of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China
(Received 1999-12-20)

Abstract: In traditional data clustering, similarity of a cluster of objects is measured by distance between objects. Such measures are not appropriate for categorical data. A new clustering criterion to determine the similarity between points with categorical attributes is presented. Furthermore, a new clustering algorithm for categorical attributes is addressed. A single scan of the dataset yields a good clustering, and more additional passes can be used to improve the quality further.

Key words: data mining; clustering; similarity

1 Introduction

Data mining can yield exciting results for almost every organization that collects data on its customers, markets, products or processes. By discovering hidden patterns and relationships in the data, data mining enables users to extract a greater value from their data than simple query and analysis approaches. Among discovering many kinds of knowledge in large databases, clustering has attracted great attention in database research communities in recent years [1–4].

Data clustering is a useful technique for grouping data points such that points within a single group have similar characteristics, while points in different groups are dissimilar. Clustering has been extensively studied in statistics, machine learning, information retrieval and so on. More recently, clustering algorithms for mining large databases have been proposed in references [1–3]. Most of the previous approaches are based on either probability or distance measure. If n is the number of different items, then each transaction is represented by a point in an n-dimensional space. Points that are nearby in this n-dimensional space are clustered together. These clustering algorithms are able to effectively cluster transactions when dimensionality of the space is relatively small and most of the items are presented in each transaction. However, these algorithms fail to produce meaningful clusters if the number of items is large and the fraction of the items present in each transaction is small. This type of datasets are quite common in many data mining domains, in which the number of different items is very large but each transaction has only few of these items. For example, consider a standard cluster C defined by the following items: liqueur, white wine, red wine, beer, champagne, orangeade, lemonade and mineral water. Transaction T_1 ={liqueur, white wine, red wine}, T_2={red wine, beer, champagne}, T_3={champagne, orangeade, lemonade}, T_4={orangeade, mineral water, liqueur}, T_5={white wine, champagne}. All transactions share no more than one item in common; therefore, pairwise similarity in this example is weak. However, we know these transactions should belong to same cluster. The above situation is further exacerbated by the fact that the set of items that define clusters may not have uniform sizes.

In addition, these algorithms are suitable for data with numeric attributes rather than Boolean and categorical attribute. Consider another situation that database contains the following 4 transactions: T_1={a, b, c}, T_2={b, c, d}, T_3={a} and T_4={d}. If the set of items in database is I={a, b, c, d}. The transactions can be viewed as points with Boolean attributes corresponding to the items a, b, c and d. Then the 4 transactions become to T_1={1, 1, 1, 0}, T_2={0, 1, 1, 1}, T_3={1, 0, 0, 0} and T_4={0, 0, 0, 1}. Using Euclidean distance or the distance of reference [2] measure these transactions, then T_1 and T_2 will be arranged into one cluster, T_3 and T_4 will be merged into another cluster. However, T_3 and T_4 do not have a single item in common. Thus, the algorithms using distance cluster for categorical data are not effective [5].

In reference [4], the authors address the problem of clustering related customer transactions in a market basket database. They assume that itemsets that define clusters are disjoint and have no overlap among them. This may not be true in practice since transactions in different clusters may have a few common items.

Reference [5] addresses the limitation of distance

metric between transactions. It presents the concept of links to measure the similarity between a pair of data points, where links are the common neighbors. It is not effective using this method to deal the first example mentioned above. In addition, it needs scan data set one more time.

In this paper, we present a new algorithm CACA (clustering algorithm for categorical attributes) to cluster the categorical data. A single scan of the dataset yields a good clustering, and more additional passes can be used to improve the quality further.

2 Clustering

2.1 Clustering criterion

We define a new clustering criterion instead of the distance measure.

Let $I=\{i_1, i_2, \cdots, i_m\}$ be a set of items. Let D be a set of transactions, where each transaction T is a set of items such that $T \subseteq I$. The support of an item i in cluster C is:

$$\sup(C(i))=\frac{\text{the number of transactions in } C \text{ that contain } i}{\text{the number of transactions in } C} \quad (1)$$

An item i is a popular item in cluster C if most of transactions in C contain i, thus, item i contributes to similarity of cluster C. Whereas a non-popular item i contributes to dissimilarity of cluster C. So, we should increase the influence of those popular items to a cluster. The support of an item in a cluster can do it, because the more popular of an item, the higher its support in a cluster.

Let $|C|$ be the number of transactions in C. If $T_a \in C_x$, $T_b \in C_y$, we define the similarity between transaction T_a and T_b as

$$\text{sim}(T_a,T_b)=\frac{|C_x|\cdot\sum_{i\in T_a}\sup(C_y(i))+|C_y|\cdot\sum_{i\in T_b}\sup(C_x(i))}{2|C_x|\cdot|C_y|} \quad (2)$$

We define the similarity between transaction T and cluster C as

$$\text{sim}(T,C)=\frac{\sum_{T\in C}\text{sim}(T,T_i)}{|C|} \quad (3)$$

If $T_a \in C_i$, $T_b \in C_i$, $T_c \in C_t$, $i \neq j$, we have $\text{sim}(T_a,T_b) \geq \text{sim}(T_a,T_c)$, $\text{sim}(T_a,T_b) \geq \text{sim}(T_b,T_c)$, $\text{sim}(T_a,C_i) \geq \text{sim}(T_c,C_i)$, $\text{sim}(T_b,C_i) \geq \text{sim}(T_c,C_i)$ and so on.

Let Cent_C to be the centroid of cluster C. Each itemset i of Cent_C stores the following two items: (1) i and (2) $n(i)$, where $n(i)$ is the number of transactions in cluster C that contain i.

And then, from equation (2), we can get the similarity of transaction T to centroid of cluster C:

$$\text{sim}(T,\text{Cent}_C)=\frac{\sum_{i\in\text{Cent}_C}\sup T(i)+\sum_{i\in T}n(i)\sup(\text{Cent}_C(i))}{2} \quad (4)$$

In equation (4), we suppose that the cluster only includes T.

2.2 Clustering tree

In the process of clustering, when we read a new transaction from database, we must identify which cluster can include the transaction. The simplest manner is calculating all the similarity of each cluster to this transaction, and the transaction should be assign to the most similarity cluster. It is suitable for using above approach when the number of clusters is less. It will be take much time when the number of clusters is high. To solve this problem, we present a notation of Clustering Tree (CT).

A clustering tree is a binary tree. The height of clustering tree H is decided by the number of clusters N. H satisfies the formula: $2^H \leq N \leq 2^{H+1}$. Each leaf node is the centroid of a cluster. Each nonleaf node contains two items of the form $[\text{Cent}_i, \text{child}_i]$, where $i=1, 2$; child_i is a pointer to its ith child node, and Cent_i is the centroid of its ith child node. So, a nonleaf node represents a subcluster made up of all its child nodes.

We give a threshold value of similarity, say MS (Minimum Similarity). And then all transactions in a leaf node must satisfy the MS; *i.e.* the similarity between any two transactions in a cluster must not less than MS. In order to prevent the CT is too large or too small, we give another limitation that the number of clusters must not less than N. Once minimum similarity and the number of clustering are given, the CT can be determined.

Such a CT will be modified dynamically as new transactions are inserted. When a new transaction T is read from database, we should insert T to CT. The process of insertion is as follows:

(1) Identifying the appropriate cluster.

Starting from the root, we select a child node that is the most similar child node to T in the root, repeating above process until we reach a leaf node.

(2) Modifying the leaf node.

In the leaf node, we find the most similar cluster, say C_t, if $\text{sim}(T,\text{Cent}_t)$ is not less than MS, then we assign T to C_t. If $\text{sim}(T,\text{Cent}_t)$ is less than MS, a new cluster will be added. We will describes "adding a new cluster" in next paragraph.

(3) Adding a new cluster.

We build the CT according to the breadth first strategy, *i.e.* we will never create the (H+1)th level of CT unless the Hth level is full. If the current number of clusters is equal to 2^H, *i.e.* the Hth level is full, then we create the (H+1)th level. For example, in **figure 1**(a), among the clusters in the 3th level, C_2 is more similar to the new transaction T than the other clusters. And then we should create the 4th level of CT, move C_2 to D_1, and let T be D_2, creating new C_2 by merging D_1 and D_2 (see figure 1(b)). If we reach a leaf node whose level is less than H, we just spilt this node to two child nodes, one child is itself and another is the new transaction.

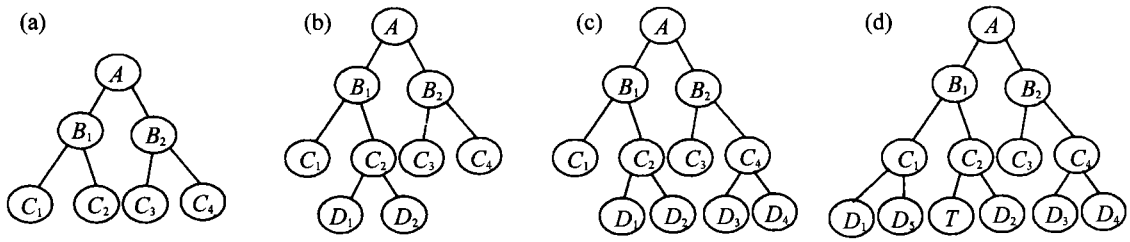

Figure 1 Four kinds of clustering trees.

When we reach a leaf node whose level is H, if at that time the number of clusters is less than 2^H, *i.e.* the Hth level is not full, we can't create a new level. We will rearrange some clusters. For example, in figure 1(c), if D_2 is most similar cluster to T among all child nodes of C_2, we will rearrange all clusters in the Hth level: D_1, D_2, D_3, D_4 and T. Because D_2 is more similar to T than D_1, we will merge T and D_2 to C_2, thus T will replace D_1. And then we find the most similar cluster to D_1 in the other leaf node of the ($H-1$)th. In figure 1(c), if C_1 is the most similar cluster to D_1, we will merge C_1 and D_1. Move C_1 to the Hth level, suppose C_1 becomes to D_5, we will get a new C_1 by merging D_5 and D_1 (see figure 1(d)).

(4) Updating the CT.

After inserting T to CT, we should update the information of CT and recalculate the value of centroid of each changed cluster and subcluster. For example, C_2, B_1 and A should be recalculated in figure 1(b), C_1, C_2, B_1 and A should be recalculated in figure 1(d).

If the number of clusters is larger than N, we will rebuild the tree. In order to limit the number of clusters, we must reduce the MS. The process of rebuilding the clustering tree is as follows: (1) Merging the old clusters to a new cluster if these clusters satisfy the new MS. (2) Splitting the old clusters that couldn't be merged. If the similarity of a transaction T to a new cluster C_{new} is satisfied the new MS, we merge T to C_{new}. Repeating the above process until no more one transaction can be merged. (3) Merging the left transaction to create new clusters according to the new MS. (4) Updating the CT.

We identify the algorithm of building the CT with build_ct (transaction T).

2.3 Clustering algorithm

After each transaction is read from the database, the preprocess of clustering is completed. If the user is satisfied with the result of clustering, and then we can end the clustering, otherwise we can execute the refine algorithm to make the result more precise. We read each transaction T, and move T to an existing non-singleton cluster that is the most similar to T. After each move, the cluster identifier is updated and any empty cluster is eliminated immediately. If no transaction is moved in one pass of all transactions, refinement algorithm terminates; otherwise, a new pass begins.

From above clustering criterion and theories, we have the algorithm for clustering transactions, say CACA, as follows:

(1) CT={}, i=1;
(2) while $i \le n$ do
(3) build_ct (T_i); // see 2.2
(4) i++;
(5) loop
(6) do
(7) clustering_over = true;
(8) for (i=1; $i \le$|CS|; i++)
(9) for (j=1; $j \le$|C_i|; j++)
(10) if exist $C_j \in$CS make sim(T_i, C_j) minimum then
(11) $C_j \leftarrow T_i$;
(12) clustering_over = false;
(13) delete C from CS where C={};
(14) until clustering_over

In CACA, n is the number of transactions in database, N is the number of clusters that the user want to obtain. The result of clustering is kept in CS, |CS| is the actual number of clusters. Clustering algorithm consists of two cycles, the first is from 1 to 5 lines, and it pre-clusters the transactions to N clusters. In order to obtain more precise result, we calculate the similarity of each transaction to each cluster in the second cycle from 6 to 14 lines. If no transaction is moved in one pass of all transactions and all clusters, the second phase terminates.

3 Experimental

In this section, we study the performance of our algorithm for clustering categorical data. Our experiment is performed on a Legend Luan-2000 machine running Windows NT4.0 with 450M Pentium III Processor and 64 MB RAM.

To get a better feel for how CACA performs in practice, we ran CACA, ROCK [5] and traditional algorithms on real-life data sets mushroom. The mushroom dataset is obtained from http://www.ics.uci.edu/~mlearn/MLSummary.html. The mushroom database has 8 124 records for 4 208 edible mushrooms and 3916 poisonous mushrooms; it records 23 attributes (edible or poisonous, cap-shape, cap-surface, cap-color, bruises, odor, population, habitat and so on) of mushroom. All attributes are categorical attributes, for example, the values of the population attribute are abundant, clustered, numerous, scattered, several or solitary.

We set the number of expected classes to be 18. CLARANS algorithm produces 18 clusters as follow as **table 1**, ROCK algorithm produces 20 clusters as follow as **table 2**, and **table 3** shows the clusters and their distribution produced by our algorithm. From tables 1–3, we can find that CACA and ROCK are more effectively and precisely than the traditional clustering algorithms.

Table 1 Clustering results of CLARANS algorithm

No.	Number of Edible	Number of Poisonous	No.	Number of Edible	Number of Poisonous	No.	Number of Edible	Number of Poisonous
1	886	4	7	253	258	13	164	183
2	302	738	8	200	160	14	216	180
3	0	508	9	125	98	15	150	240
4	402	50	10	192	236	16	220	318
5	14	144	11	138	164	17	168	0
6	225	168	12	148	160	18	405	307

Table 2 Clustering results of ROCK algorithm

No.	Number of Edible	Number of Poisonous	No.	Number of Edible	Number of Poisonous	No.	Number of Edible	Number of Poisonous
1	96	0	8	0	32	15	32	72
2	0	292	9	0	1 296	16	0	1 728
3	704	0	10	0	8	17	288	0
4	96	0	11	48	0	18	0	8
5	768	0	12	48	0	19	192	0
6	0	168	13	0	288	20	16	0
7	1 728	0	14	192	0			

Table 3 Clustering results of CACA algorithm

No.	Number of Edible	Number of Poisonous	No.	Number of Edible	Number of Poisonous	No.	Number of Edible	Number of Poisonous
1	96	0	7	1 782	0	13	0	297
2	0	263	8	0	38	14	286	0
3	42	0	9	0	1 296	15	46	72
4	140	0	10	0	22	16	0	1 728
5	1 620	0	11	102	0			
6	2	200	12	92	0			

All of the clusters (except cluster 15) discovered by our algorithm are pure clusters, *i.e.* mushrooms in each cluster are either all edible or all poisonous. We can say our clustering result is ideal because the algorithm CACA can distinguish the edible and poisonous mushrooms. All items in cluster 15 share more common attributes thus it is not well-separated. We can further refine Cluster 15 to get more purity by adjusting the threshold MS.

The clusters produced by CACA is more concentrate than other algorithms, 79% data lies in the clusters which contain more than 1 000 transactions. ROCK achieves the purity of clusters by creating more clusters. However, it could not capture the structure of data in a concise representation because only 58% data lies in clusters whose number is great than 1 000. So, CACA get few clusters without losing precise.

4 Conclusions

Traditional clustering algorithms rely on a measure of distance of points, and these algorithms are unsuitable for clustering the categorical data. In this paper, we present a new clustering criterion to determine the similarity between a pair of points with categorical attributes. We also present a new algorithm for clustering categorical data. Experimental results show that our approach is effective. In the future, we will research how to choose a much better minimum similarity MS in the process of clustering.

References

[1] R. T. Ng, J. Han: Efficient and Effective Clustering Methods for Spatial Data Mining. [In] Bocca J B, Jarke M, Zaniolo C eds.: *Proceedings of the 20th International Conference on Very Large Databases*. Morgan Kaufmann, San Francisco, 1994, pp.144-155.

[2] T. Zhang, R. Ramakrishnan, M. Livny: *Data Mining and Knowledge*, 1997, No.1, p.141.

[3] M. Ester, H. P. Kriegel, X. Xu: A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. [In] E. Simoudis, J. Han, U. M. Fayyad eds.: *Proceedings of the 2nd International Conference on Knowledge Discovery in Databases and Data Mining*. AAAI Press, Menlo Park, 1996, pp.226-231.

[4] E. Han, G. Karypis, V. Kumar, *et al.*: Clustering Based on Association Rule Hypergraphs. [In] J. Peckham ed.: *1997 SIGMIOD Workshop on Research Issues on Data Mining and Knowledge Discovery*. ACM Press, New York, 1997.

[5] S. Guha, R. Rastogi, K. Shim: ROCK: A Robust Clustering Algorithm for Categorical Attributes. [In] *Proceedings of the 15th International Conference on Data Engineering*. IEEE Computer Society, Los Alamitos, 1999, pp.521-521.