ROCK: A Robust Clustering Algorithm for Categorical Attributes

Authors: S. Guha, R. Rastogi, K. Shim
Presented by: Weinan Wang

What does the paper talk about?

- Traditional clustering algorithms that use distances between points for clustering are not appropriate for boolean and categorical attributes.
- A novel concept of *links* to measure the similarity between a pair of data points.
- A robust hierarchical clustering algorithm *ROCK* that employs links when merging clusters.

1. Shortcoming of Traditional Clustering Algorithm

What is clustering?

"Clustering is a technique for grouping data points such that points within a single group/ cluster have *similar* characteristics while points in different groups are *dissimilar*."

What is *similar/dissimilar*?

How to measure similarity/dissimilarity?

The most commonly used measurement for similarity in metric space is L_p metric.

The most commonly used criterion function for metric spaces is:

$$E = \sum_{i=1}^{k} \sum_{\vec{x} \in C_i} d(\vec{x}, \vec{m}_i)$$

The criterion function attempts to minimize the distance Of every point from the mean of the cluster to which the point belongs.

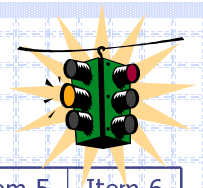
Example: market basket database

Trans.	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
# 1	1	1	1	0	1	0
# 2	0	1	1	1	1	0
# 3	1	0	0	1	0	0
# 4	0	0	0	0	0	1

Use Euclidean distance to measure the closeness between clusters:

Step1. distance between #1 and #2 is 1.414, which is the smallest distance between pairs of points. Merge #1 and #2.

Market Basket example cont'd:



Cluster	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
# 1	0.5	1	1	0.5	1	0
# 3	1	0	0	1	0	0
# 4	0	0	0	0	0	1

Step 2. Cluster #3 and cluster #4 are merged since the distance between them is 1.732 which is less than the distance between the centroid of the merged cluster from each of them.

Trans. #3 and trans. #4 have no item in common!

"Distances between points are not appropriate for boolean and categorical attributes."

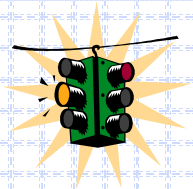
Another selection for similarity criterion

Instead of Euclidean distance, we can use set theoretic similarity measures such as the Jaccard coefficient to measure the similarity of two clusters.

The Jaccard coefficient for similarity between transactions T1 and T2 is :

$$\frac{|T_1 \cap T_2|}{|T_1 \cup T_2|}$$

Basket data example for Jaccard coefficient



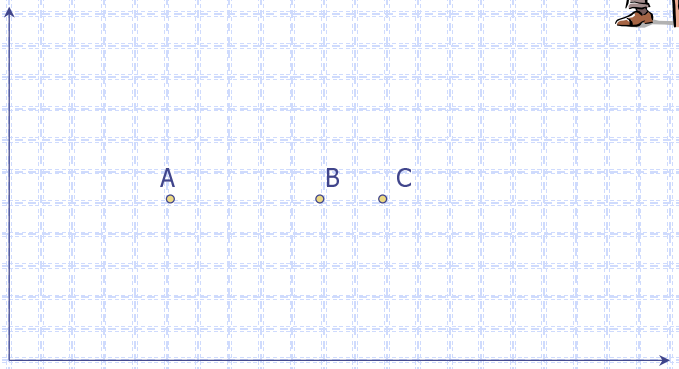
<1, 2, 3, 4, 5>	
{1, 2, 3}	{1, 4, 5}
{1, 2, 4}	{2, 3, 4}
{1, 2, 5}	{2, 3, 5}
{1, 3, 4}	{2, 4, 5}
{1, 3, 5}	{3, 4, 5}

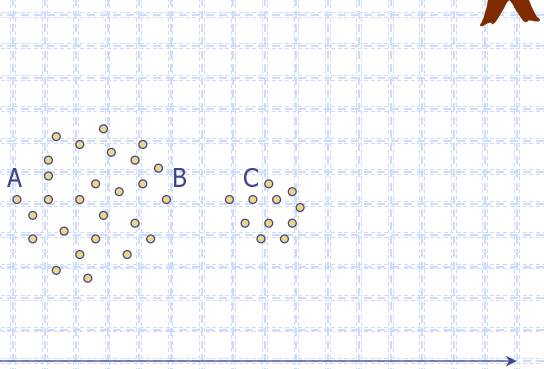
<1, 2, 6, 7>
{1, 2, 6}
{1, 2, 7}
{1, 6, 7}
{2, 6, 7}

Consider a market basket database over items 1, 2, …, 8, 9. Consider the two transaction clusters shown above.

Jaccard coefficient of {1, 2, 3} and {3, 4, 5} is 0.2 – they are in same cluster; Jaccard coefficient of {1, 2, 3} and {1, 2, 6} is 0.5, but they are not in a same cluster. Why?

Should B in the same cluster with C or with A?





B should be in the same cluster with A, but not in the same cluster with C, although B is more similar to C than to A.

2. Concepts of *neighbors* and *links*.

- A point's *neighbors* are those points that are considerably *similar* to it.
- Let $sim(p_i, p_j)$ be a similarity function that is normalized and captures the closeness between the pair of points p_i and p_j. Given a threshold θ , a pair of points p_i and p_j are defined to be *neighbors* if the following holds:

$$Sim(pi, pj) \geq \theta$$

- links(pi, pj) between two points p_i and p_j is the number of common neighbors between p_i and p_j .

Revisit the basket data example

<1, 2, 3, 4, 5>	
{1, 2, 3}	{1, 4, 5}
{1, 2, 4}	{2, 3, 4}
{1, 2, 5}	{2, 3, 5}
{1, 3, 4}	{2, 4, 5}
{1, 3, 5}	{3, 4, 5}

<1, 2, 6, 7>
{1, 2, 6}
{1, 2, 7}
{1, 6, 7}
{2, 6, 7}

Let's define the similarity function between two transactions as:

$$sim(T_1, T_2) = \frac{|T_1 \cap T_2|}{|T_1 \cup T_2|}$$

Revisit the basket data example cont'd

<1, 2, 3, 4, 5>	
{1, 2, 3}	{1, 4, 5}
{1, 2, 4}	{2, 3, 4}
{1, 2, 5}	{2, 3, 5}
{1, 3, 4}	{2, 4, 5}
{1, 3, 5}	{3, 4, 5}

<1, 2, 6, 7>
{1, 2, 6}
{1, 2, 7}
{1, 6, 7}
{2, 6, 7}

Let θ=0.5. We see:

- Transaction {1, 2, 6} has 5 links with transaction {1, 2, 7} and only 3 links with transaction {1, 2, 3}.
- Transaction {1, 6, 7} has 2 links with every transaction in the smaller cluster and 0 links with every other transaction in the bigger cluster

Criterion Function for clustering

Maximize following criterion function for the k clusters:

$$E_l = \sum_{i=1}^{k} n_i * \sum_{p_q, p_r \in C_i} \frac{link(p_q, p_r)}{n_i^{1+2f(\theta)}}$$

Where C_i denotes cluster i of size n_i

3. ROCK Clustering Algorithm

Figure 2: Overview of ROCK

- step1: draw a random sample from the database;
- step2: apply a hierarchical clustering algorithm that employs links to the sampled points.
- step3: The clusters involving only the sampled points are used to assign the remaining data points on disk to the appropriate clusters.

Goodness Measure for Merging two clusters

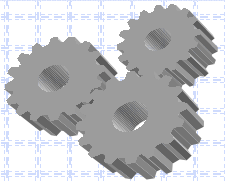
For a pair of clusters C_i and C_j, let links[C_i, C_j] store the number of cross links between clusters C_i and C_j, that is:

$$\sum_{p_q \in C_i, p_r \in C_j} links(p_q, p_r)$$

Define the goodness measure $g(C_i, C_j)$ for merging clusters C_i, C_j as follows:

$$g(C_i, C_j) = \frac{link[C_i, C_j]}{(n_i + n_j)^{1+2f(\theta)} - n_i^{1+2f(\theta)} - n_j^{1+2f(\theta)}}$$

Clustering Algorithm



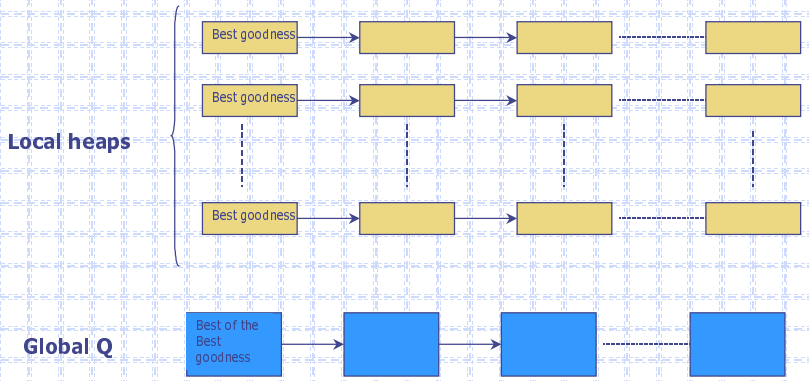
Input:

- set S of n sampled points to be clustered;
- the number of desired clusters k;
- threshold for similarity function.

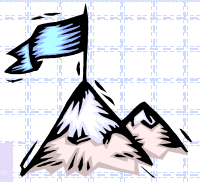
Data Strucrure:

- Each cluster *i* hold a local heap q[i]
- A global heap Q that contains all the clusters.

Clustering Algorithm cont'd



Pseudo code of the Clustering Algorithm



```
procedure cluster(S, k)
begin
1.  link := compute_links(S)
2.  for each s ∈ S do
3.      q[s] := build_local_heap(link, s)
4.  Q := build_global_heap(S, q)
5.  while size(Q) > k do {
6.      u := extract_max(Q)
7.      v := max(q[u])
8.      delete(Q, v)
9.      w := merge(u, v)
10.     for each x ∈ q[u] ∪ q[v] do {
11.         link[x, w] := link[x, u] + link[x, v]
12.         delete(q[x], u); delete(q[x], v)
13.         insert(q[x], w, g(x, w)); insert(q[w], x, g(x, w))
14.         update(Q, x, q[x])
15.     }
16.     insert(Q, w, q[w])
17.     deallocate(q[u]); deallocate(q[v])
18. }
end
```

Question ?