k-medoids

From Wikipedia, the free encyclopedia

The ***k*-medoids algorithm** is a clustering algorithm related to the k-means algorithm and the medoidshift algorithm. Both the k-means and k-medoids algorithms are partitional (breaking the dataset up into groups) and both attempt to minimize the distance between points labeled to be in a cluster and a point designated as the center of that cluster. In contrast to the k-means algorithm, k-medoids chooses datapoints as centers (medoids or exemplars) and works with an arbitrary matrix of distances between datapoints instead of l_2. This method was proposed in 1987[1] for the work with l_1 norm and other distances.

k-medoid is a classical partitioning technique of clustering that clusters the data set of n objects into k clusters known *a priori*. A useful tool for determining k is the silhouette.

It is more robust to noise and outliers as compared to k-means because it minimizes a sum of pairwise dissimilarities instead of a sum of squared Euclidean distances.

A medoid can be defined as the object of a cluster whose average dissimilarity to all the objects in the cluster is minimal. i.e. it is a most centrally located point in the cluster.

The most common realisation of k-medoid clustering is the **Partitioning Around Medoids (PAM)** algorithm and is as follows:[2]

1. Initialize: randomly select (without replacement) k of the n data points as the medoids
2. Associate each data point to the closest medoid. ("closest" here is defined using any valid distance metric, most commonly Euclidean distance, Manhattan distance or Minkowski distance)
3. For each medoid m
   1. For each non-medoid data point o
      1. Swap m and o and compute the total cost of the configuration
4. Select the configuration with the lowest cost.
5. Repeat steps 2 to 4 until there is no change in the medoid.

Contents

Demonstration of PAM

Cluster the following data set of ten objects into two clusters i.e. $k = 2$.

Consider a data set of ten objects as follows:

X_1	2	6
X_2	3	4
X_3	3	8
X_4	4	7
X_5	6	2
X_6	6	4
X_7	7	3
X_8	7	4
X_9	8	5
X_{10}	7	6

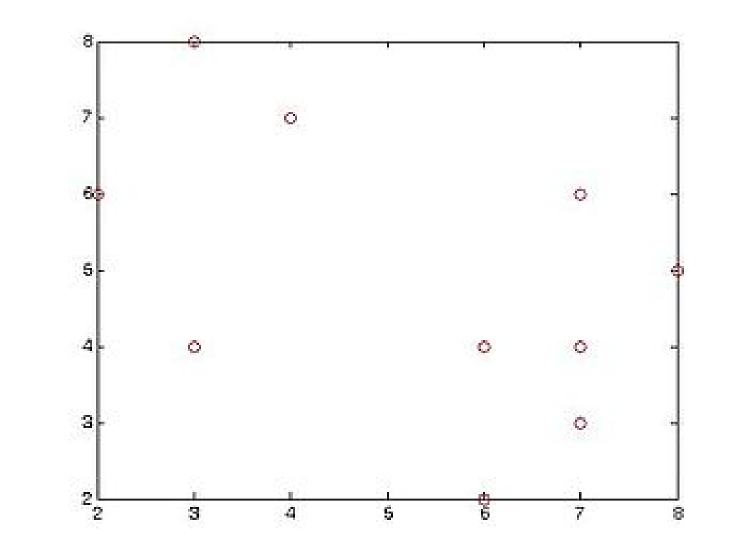

Figure 1.1 – distribution of the data

Step 1

Initialize k centers.

Let us assume x_2 and x_8 are selected as medoids, so the centers are $c_1 = (3,4)$ and $c_2 = (7,4)$

Calculate distances to each center so as to associate each data object to its nearest medoid. Cost is calculated using Manhattan distance (Minkowski distance metric with $r = 1$). Costs to the nearest medoid are shown bold in the table.

Cost (distance) to c_1					
i	c_1		Data objects (X_i)		Cost (distance)
1	3	4	2	6	**3**
3	3	4	3	8	**4**
4	3	4	4	7	**4**
5	3	4	6	2	5
6	3	4	6	4	3
7	3	4	7	3	5
9	3	4	8	5	6
10	3	4	7	6	6

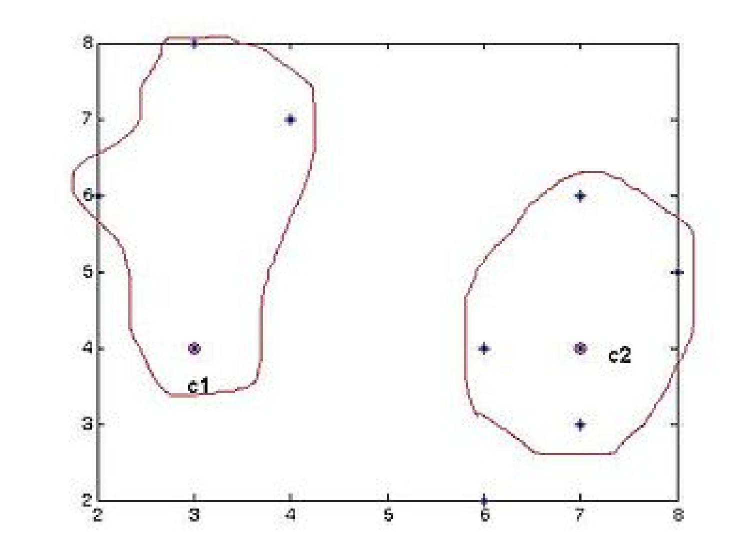

Figure 1.2 – clusters after step 1

Cost (distance) to c_2					
i	c_2		Data objects (X_i)		Cost (distance)
1	7	4	2	6	7
3	7	4	3	8	8
4	7	4	4	7	6
5	7	4	6	2	**3**
6	7	4	6	4	**1**
7	7	4	7	3	**1**
9	7	4	8	5	**2**
10	7	4	7	6	**2**

Then the clusters become:

Cluster_1 = {(3,4)(2,6)(3,8)(4,7)}

Cluster_2 = {(7,4)(6,2)(6,4)(7,3)(8,5)(7,6)}

Since the points (2,6) (3,8) and (4,7) are closer to c_1 hence they form one cluster whilst remaining points form another cluster.

So the total cost involved is 20.

Where cost between any two points is found using formula

$$\text{cost}(x, c) = \sum_{i=1}^{d} |x_i - c_i|$$

where x is any data object, c is the medoid, and d is the dimension of the object which in this case is 2.

Total cost is the summation of the cost of data object from its medoid in its cluster so here:

$$\begin{aligned}
\text{total cost} = &\{\text{cost}((3,4),(2,6)) + \text{cost}((3,4),(3,8)) + \text{cost}((3,4),(4,7))\} \\
&+ \{\text{cost}((7,4),(6,2)) + \text{cost}((7,4),(6,4)) + \text{cost}((7,4),(7,3)) \\
&+ \text{cost}((7,4),(8,5)) + \text{cost}((7,4),(7,6))\} \\
= &(3+4+4) + (3+1+1+2+2) \\
= &20
\end{aligned}$$

Step 2

Select one of the nonmedoids O′

Let us assume O′ = (7,3)

So now the medoids are $c_1(3,4)$ and $O'(7,3)$

If $c1$ and O' are new medoids, calculate the total cost involved

By using the formula in the step 1

i	c_1		Data objects (X_i)		Cost (distance)
1	3	4	2	6	**3**
3	3	4	3	8	**4**
4	3	4	4	7	**4**
5	3	4	6	2	5
6	3	4	6	4	3
8	3	4	7	4	4
9	3	4	8	5	6
10	3	4	7	6	6

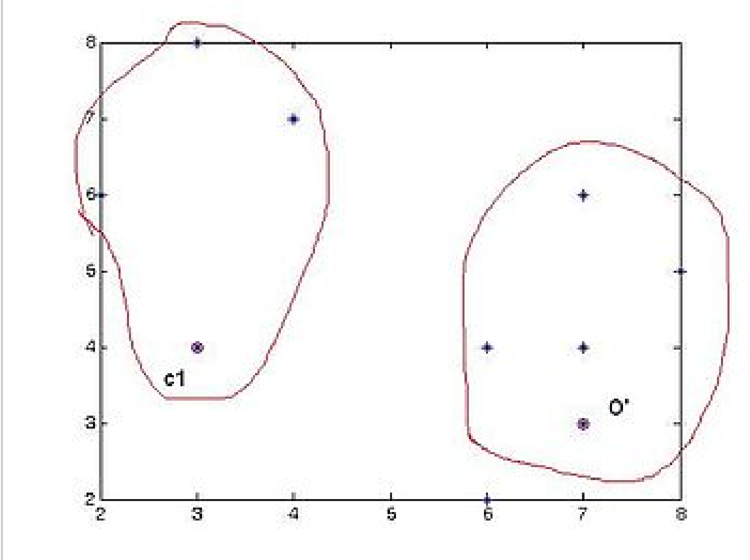

Figure 1.3 – clusters after step 2

i	O'		Data objects (X_i)		Cost (distance)
1	7	3	2	6	8
3	7	3	3	8	9
4	7	3	4	7	7
5	7	3	6	2	**2**
6	7	3	6	4	**2**
8	7	3	7	4	**1**
9	7	3	8	5	**3**
10	7	3	7	6	**3**

$$\begin{aligned}\text{total cost} &= 3+4+4+2+2+1+3+3\\ &= 22\end{aligned}$$

So cost of swapping medoid from c_2 to O' is

$$\begin{aligned}S &= \text{current total cost} - \text{past total cost}\\ &= 22-20\\ &= 2>0.\end{aligned}$$

=== program:

So moving to O' would be a bad idea, so the previous choice was good. So we try other nonmedoids and found that our first choice was the best. So the configuration does not change and algorithm terminates here (i.e. there is no change in the medoids).

It may happen some data points may shift from one cluster to another cluster depending upon their closeness to medoid.

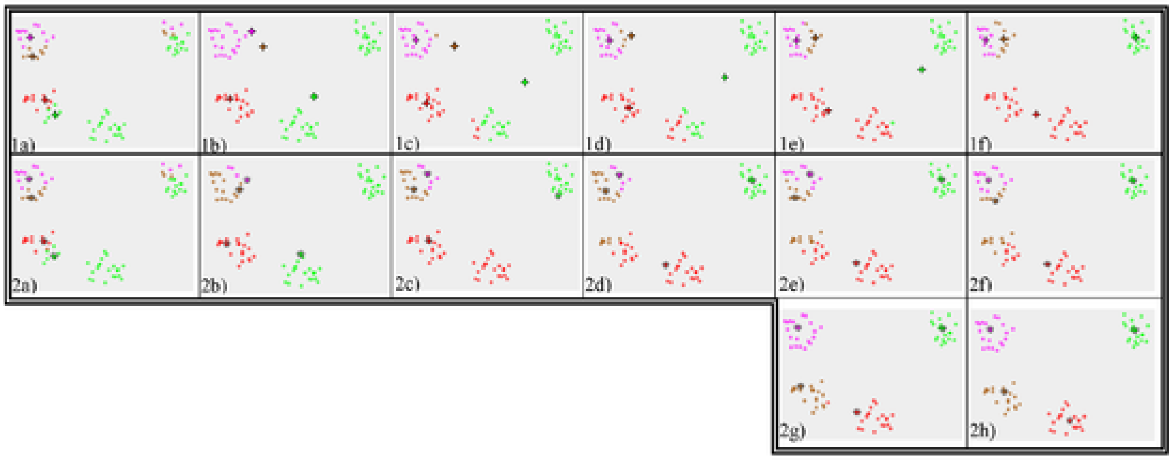

Figure 2. K-medoids versus k-means. Figs 2.1a-2.1f present a typical example of the k-means convergence to a local minimum. This result of k-means clustering contradicts the obvious cluster structure of data set. In this example, k-medoids algorithm (Figs 2.2a-2.2h) with the same initial position of medoids (Fig. 2.2a) converges to the obvious cluster structure. The small circles are data points, the four ray stars are centroids (means), the nine ray stars are medoids.[3]

In some standard situations, k-medoids demonstrate better performance than k-means. An example is presented in Fig. 2. The most time-consuming part of the k-medoids algorithm is the calculation of the distances between objects. If a quadratic preprocessing and storage is applicable, the distances matrix can be precomputed to achieve consequent speed-up. See for example,[4] where the authors also introduce a heuristic to choose the initial k medoids.

Software

- ELKI includes several *k*-means variants, including an EM-based *k*-medoids and the original PAM algorithm.
- Julia contains a *k*-medoid implementation in the JuliaStats clustering package.[5]
- R includes in the "flexclust" package variants of *k*-means and in the "cluster" package.
- RapidMiner has an operator named KMedoids, but it does *not* implement the KMedoids algorithm correctly. Instead, it is a k-means variant, that substitutes the mean with the closes data point (which is not the medoid).
- Java-ML (http://java-ml.sourceforge.net/). Includes a *k*-medoid implementation that is incorrect in the same way as the RapidMiner version.

External links

E.M. Mirkes, K-means and K-medoids (http://www.math.le.ac.uk/people/ag153/homepage/KmeansKmedoids/Kmeans_Kmedoids.html) (Applet), University of Leicester, 2011.

References

1. Kaufman, L. and Rousseeuw, P.J. (1987), Clustering by means of Medoids, in Statistical Data Analysis Based on the L_1–Norm and Related Methods, edited by Y. Dodge, North-Holland, 405–416.
2. Sergios Theodoridis & Konstantinos Koutroumbas (2006). *Pattern Recognition 3rd ed.* p. 635.
3. The illustration was prepared with the Java applet, E.M. Mirkes, K-means and K-medoids: applet (http://www.math.le.ac.uk/people/ag153/homepage/KmeansKmedoids/Kmeans_Kmedoids.html). University of

Leicester, 2011.
4. H.S. Park , C.H. Jun, A simple and fast algorithm for K-medoids clustering, Expert Systems with Applications, 36, (2) (2009), 3336–3341.
5. Clustering.jl (https://github.com/JuliaStats/Clustering.jl)

Categories: Statistical algorithms | Data clustering algorithms

---