**k-Means: Step-By-Step Example**

As a simple illustration of a k-means algorithm, consider the following data set consisting of the scores of two variables on each of seven individuals:

|  |  |  |
| --- | --- | --- |
| Subject | A | B |
| 1 | 1.0 | 1.0 |
| 2 | 1.5 | 2.0 |
| 3 | 3.0 | 4.0 |
| 4 | 5.0 | 7.0 |
| 5 | 3.5 | 5.0 |
| 6 | 4.5 | 5.0 |
| 7 | 3.5 | 4.5 |

This data set is to be grouped into two clusters.  As a first step in finding a sensible initial partition, let the A & B values of the two individuals furthest apart (using the Euclidean distance measure), define the initial cluster means, giving:

|  |  |  |
| --- | --- | --- |
|  | Individual | Mean Vector (centroid) |
| Group 1 | 1 | (1.0, 1.0) |
| Group 2 | 4 | (5.0, 7.0) |

The remaining individuals are now examined in sequence and allocated to the cluster to which they are closest, in terms of Euclidean distance to the cluster mean. The mean vector is recalculated each time a new member is added. This leads to the following series of steps:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Cluster 1 | | Cluster 2 | |
| Step | Individual | Mean Vector (centroid) | Individual | Mean Vector (centroid) |
| 1 | 1 | (1.0, 1.0) | 4 | (5.0, 7.0) |
| 2 | 1, 2 | (1.2, 1.5) | 4 | (5.0, 7.0) |
| 3 | 1, 2, 3 | (1.8, 2.3) | 4 | (5.0, 7.0) |
| 4 | 1, 2, 3 | (1.8, 2.3) | 4, 5 | (4.2, 6.0) |
| 5 | 1, 2, 3 | (1.8, 2.3) | 4, 5, 6 | (4.3, 5.7) |
| 6 | 1, 2, 3 | (1.8, 2.3) | 4, 5, 6, 7 | (4.1, 5.4) |

Now the initial partition has changed, and the two clusters at this stage having the following characteristics:

|  |  |  |
| --- | --- | --- |
|  | Individual | Mean Vector (centroid) |
| Cluster 1 | 1, 2, 3 | (1.8, 2.3) |
| Cluster 2 | 4, 5, 6, 7 | (4.1, 5.4) |

But we cannot yet be sure that each individual has been assigned to the right cluster.  So, we compare each individual’s distance to its own cluster mean and to  
that of the opposite cluster. And we find:

|  |  |  |
| --- | --- | --- |
| Individual | Distance to mean (centroid) of Cluster 1 | Distance to mean (centroid) of Cluster 2 |
| 1 | 1.5 | 5.4 |
| 2 | 0.4 | 4.3 |
| 3 | 2.1 | 1.8 |
| 4 | 5.7 | 1.8 |
| 5 | 3.2 | 0.7 |
| 6 | 3.8 | 0.6 |
| 7 | 2.8 | 1.1 |

Only individual 3 is nearer to the mean of the opposite cluster (Cluster 2) than its own (Cluster 1).  In other words, each individual's distance to its own cluster mean should be smaller that the distance to the other cluster's mean (which is not the case with individual 3).  Thus, individual 3 is relocated to Cluster 2 resulting in the new partition:

|  |  |  |
| --- | --- | --- |
|  | Individual | Mean Vector (centroid) |
| Cluster 1 | 1, 2 | (1.3, 1.5) |
| Cluster 2 | 3, 4, 5, 6, 7 | (3.9, 5.1) |

The iterative relocation would now continue from this new partition until no more relocations occur.  However, in this example each individual is now nearer its own cluster mean than that of the other cluster and the iteration stops, choosing the latest partitioning as the final cluster solution.

Also, it is possible that the k-means algorithm won't find a final solution.  In this case it would be a good idea to consider stopping the algorithm after a pre-chosen maximum of iterations.

# k-medoids

The **k-medoids algorithm** is a [clustering](http://en.wikipedia.org/wiki/Data_clustering) [algorithm](http://en.wikipedia.org/wiki/Algorithm) related to the [k-means](http://en.wikipedia.org/wiki/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](http://en.wikipedia.org/wiki/Medoids" \o "Medoids) or exemplars) and works with an arbitrary matrix of distances between datapoints instead of l_2. This method was proposed in 1987[[1]](http://en.wikipedia.org/wiki/K-medoids#cite_note-1) for the work with l_1norm 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](http://en.wikipedia.org/wiki/Silhouette_(clustering)).

It is more robust to noise and outliers as compared to [k-means](http://en.wikipedia.org/wiki/K-means) because it minimizes a sum of pairwise dissimilarities instead of a sum of squared Euclidean distances.

A [medoid](http://en.wikipedia.org/wiki/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]](http://en.wikipedia.org/wiki/K-medoids" \l "cite_note-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](http://en.wikipedia.org/wiki/Metric_space), most commonly [Euclidean distance](http://en.wikipedia.org/wiki/Euclidean_distance), [Manhattan distance](http://en.wikipedia.org/wiki/Manhattan_distance) or [Minkowski distance](http://en.wikipedia.org/wiki/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.

## Demonstration of PAM[[edit](http://en.wikipedia.org/w/index.php?title=K-medoids&action=edit&section=1" \o "Edit section: 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:

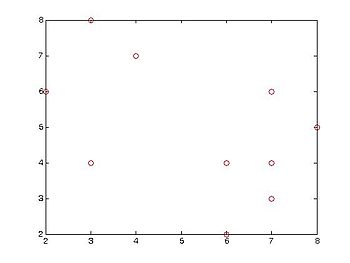
[](http://en.wikipedia.org/wiki/File:Kmedoid1.jpg)

Figure 1.1 – distribution of the data

|  |  |  |
| --- | --- | --- |
| X1 | 2 | 6 |
| X2 | 3 | 4 |
| X3 | 3 | 8 |
| X4 | 4 | 7 |
| X5 | 6 | 2 |
| X6 | 6 | 4 |
| X7 | 7 | 3 |
| X8 | 7 | 4 |
| X9 | 8 | 5 |
| X10 | 7 | 6 |

### Step 1[[edit](http://en.wikipedia.org/w/index.php?title=K-medoids&action=edit&section=2)]

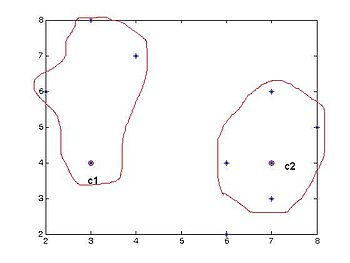
[](http://en.wikipedia.org/wiki/File:Kmedoid2.jpg)

Figure 1.2 – clusters after step 1

Initialize k centers.

Let us assume x2 and x8 are selected as medoids, so the centers are c1 = (3,4) and c2 = (7,4)

Calculate distances to each center so as to associate each data object to its nearest medoid. Cost is calculated using [Manhattan distance](http://en.wikipedia.org/wiki/Manhattan_distance) ([Minkowski distance](http://en.wikipedia.org/wiki/Minkowski_distance" \o "Minkowski distance) metric with r = 1). Costs to the nearest medoid are shown bold in the table.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Cost (distance) to c1 | | | | | |
| *i* | c1 | | 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 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Cost (distance) to c2 | | | | | |
| *i* | c2 | | 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:

Cluster1 = {(3,4)(2,6)(3,8)(4,7)}

Cluster2 = {(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 c1 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


\mbox{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{align}
\mbox{total cost} & = \{\mbox{cost}((3,4),(2,6)) + \mbox{cost}((3,4),(3,8))+ \mbox{cost}((3,4),(4,7))\} \\
 & ~+ \{\mbox{cost}((7,4),(6,2)) + \mbox{cost}((7,4),(6,4)) + \mbox{cost}((7,4),(7,3)) \\
 & ~+ \mbox{cost}((7,4),(8,5)) + \mbox{cost}((7,4),(7,6)) \} \\
 & = (3 + 4 + 4) + (3 + 1 + 1 + 2 + 2) \\
 & = 20 \\
\end{align}


### Step 2[[edit](http://en.wikipedia.org/w/index.php?title=K-medoids&action=edit&section=3)]

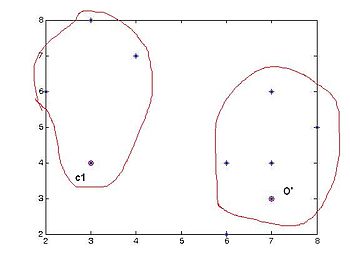
[](http://en.wikipedia.org/wiki/File:Kmedoid3.jpg)

Figure 1.3 – clusters after step 2

Select one of the nonmedoids O′

Let us assume O′ = (7,3)

So now the medoids are c1(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* | c1 | | 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 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *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** |

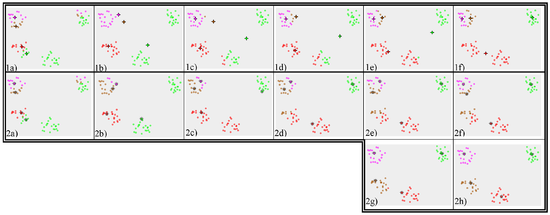
[](http://en.wikipedia.org/wiki/File:K-means_versus_k-medoids.png)

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]](http://en.wikipedia.org/wiki/K-medoids#cite_note-3)


\begin{align}
\mbox{total cost} & = 3 + 4 + 4 + 2 + 2 + 1 + 3 + 3 \\
 & = 22 \\
\end{align}


So cost of swapping medoid from c2 to O′ is


\begin{align}
 S & = \mbox{current total cost} - \mbox{past total cost} \\
 & = 22 - 20 \\
 & = 2 > 0.
\end{align}
=== 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.

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]](http://en.wikipedia.org/wiki/K-medoids#cite_note-4) where the authors also introduce a heuristic to choose the initial k medoids.