Clustering

Hung Le

University of Victoria

March 1, 2019

Clustering problem

Given a dataset $\mathcal{D}$, find a way to split $\mathcal{D}$ into *clusters* such that data points in the same clusters have high similarity while data points in different clusters have low similarity.

Sometimes high similarity means low distance and vice versa.

Clustering problem

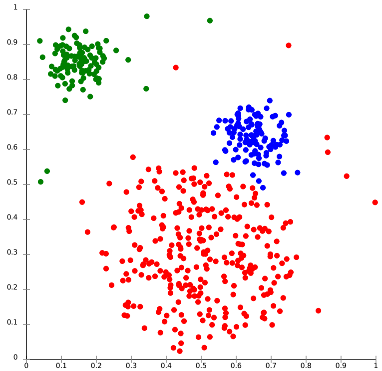


Figure: An example of clustering problem[1]

[1] https://en.wikipedia.org/wiki/Cluster_analysis#/media/File:OPTICS-Gaussian-data.svg

Distance and Similarity Measures

Given two points (vectors) $\mathbf{p}, \mathbf{q} \in \mathbb{R}^d$, we can measure:

- The Euclidean distance between **p** and **q** is:

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{d} (\mathbf{p}[i] - \mathbf{q}[i])^2} \tag{1}$$

- The cosine similarity between **p** and **q** is:

$$\cos(\mathbf{p}, \mathbf{q}) = \frac{\sum_{i=1}^{n} \mathbf{p}[i]\mathbf{q}[i]}{||\mathbf{p}||_2 ||\mathbf{q}||_2} \tag{2}$$

The Curse of Dimensionality

High dimensional Euclidean spaces are very weird:

- Choosing n random points on the unit cube, i.e, choosing $\mathbf{x}[i]$ randomly from $[0, 1]$ for each point $\mathbf{x}$, almost all points will have a distance close to the average distance.
- The cosine between two random vectors is almost close to 0 w.h.p, which means the angle is close to 90 degrees.
- Choosing random points in a hypersphere, most of them would close to the surface of the sphere.
- Many algorithms have running time of the form 2^d and in many cases, this is the best we can do.
- Many more.

Hierarchical Clustering

$\textsc{HCClustering}(\mathcal{D})$

$\mathcal{C} \leftarrow \emptyset$

for each p in $\mathcal{D}$

$\mathcal{C} \leftarrow \mathcal{C} \cup \{p\}$

repeat

Pick the best two clusters C_1, C_2 in $\mathcal{C}$

$C \leftarrow C_1 \cup C_2$

$\mathcal{C} \leftarrow \mathcal{C} \setminus \{C_1, C_2\} \cup C$

until stop

return $\mathcal{C}$

Hierarchical Clustering

$\textsc{HCClustering}(\mathcal{D})$
$\quad \mathcal{C} \leftarrow \emptyset$
$\quad$ **for each** p in $\mathcal{D}$
$\quad\quad \mathcal{C} \leftarrow \mathcal{C} \cup \{p\}$
$\quad$ **repeat**
$\quad\quad$ Pick the best two clusters C_1, C_2 in $\mathcal{C}$
$\quad\quad C \leftarrow C_1 \cup C_2$
$\quad\quad \mathcal{C} \leftarrow \mathcal{C} \setminus \{C_1, C_2\} \cup C$
$\quad$ **until** stop
$\quad$ return $\mathcal{C}$

- Which cluster pair is the best to merge?
- When to stop?

Stopping Conditions

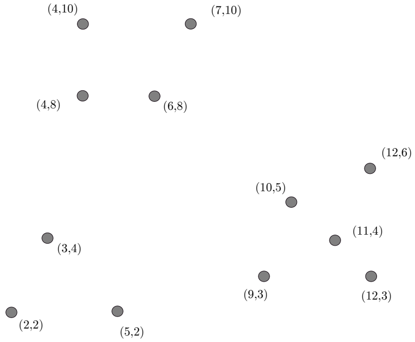
There are so many possible ways to determine the stopping condition. Here are a few examples:

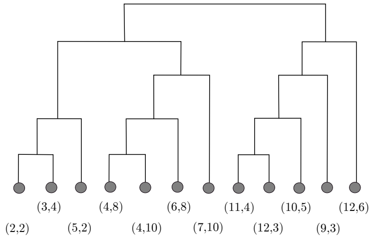
- When the number of clusters reach a predetermined threshold K.
- When the combination of two best clusters produce an unsatisfactory results:
  - If the *diameter* of the resulting cluster is big. Diameter is the maximum pairwise distance of points in a cluster.
  - If the *density* of the resulting cluster is low. Density is roughly the number of points per unit volume of the cluster.

Stopping Conditions (Cont.)

There are so many possible ways to determine the stopping condition. Here are a few examples:

- There is only one cluster left. Typically, in this case, we are interested in the *cluster tree*: the tree representing the merging process.





Best Cluster Pair

Two clusters are the best for merging if:

- Their *centroids* are closest among all pairs of clusters. A centroid of the point set X is the average point $\mathbf{c}$:

$$\mathbf{c} = \frac{\sum_{\mathbf{x} \in X} \mathbf{x}}{|X|} \tag{3}$$

  Centroids are only well-defined in Euclidean spaces.

- Or the *minimum pairwise distance* of points between two clusters is minimum (single-linkage).
- Or the *average distance* between two clusters is minimum (average-linkage).

Best Cluster Pair (Cont.)

Two clusters are the best for merging if:

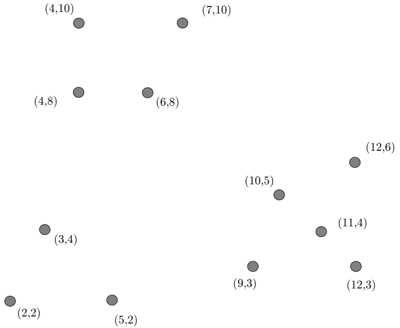
- Or combining two clusters produces the cluster with *lowest radius*. Radius of a cluster X with centroid $\mathbf{c}$ is:

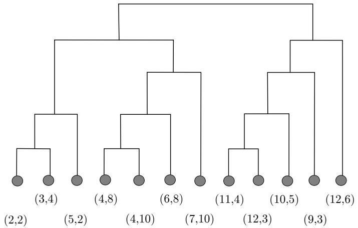
$$\mathrm{Rad}(X) = \max_{\mathbf{x} \in X} d(\mathbf{x}, \mathbf{c}) \tag{4}$$

- Or combining two clusters produces the cluster with *smallest diameter* (complete-linkage). The diameter of a cluster X is:

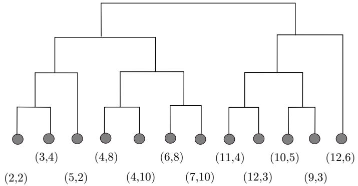
$$\mathrm{Diam}(X) = \ max_{\mathbf{x},\mathbf{y} \in X} d(\mathbf{x}, \mathbf{y}) \tag{5}$$

Effect of Different Merging Criteria





Closest Centroid Rule



Minimum Diameter Rule

Computational Complexity of HC

Let's focus on a specific setting:

- Two clusters are the best for merging if their centroids are closest among all pairs of clusters in $\mathcal{C}$.
- Stop merging when there is only one cluster left.

Hierarchical Clustering

$\textsc{HCClustering}(\mathcal{D})$

$\mathcal{C} \leftarrow \emptyset$

for each p in $\mathcal{D}$

$\mathcal{C} \leftarrow \mathcal{C} \cup \{p\}$

repeat

Pick the clusters C_1, C_2 in $\mathcal{C}$ with minimum centroid distance.

$C \leftarrow C_1 \cup C_2$

$\mathcal{C} \leftarrow \mathcal{C} \setminus \{C_1, C_2\} \cup C$

until $|\mathcal{C}| = 1$

return $\mathcal{C}$

Hierarchical Clustering

HCClustering($\mathcal{D}$)
$\quad\mathcal{C} \leftarrow \emptyset$
$\quad$**for each** p in $\mathcal{D}$
$\quad\quad\mathcal{C} \leftarrow \mathcal{C} \cup \{p\}$
$\quad$**repeat**
$\quad\quad$Pick the clusters C_1, C_2 in $\mathcal{C}$ with minimum centroid distance.
$\quad\quad C \leftarrow C_1 \cup C_2$
$\quad\quad\mathcal{C} \leftarrow \mathcal{C} \setminus \{C_1, C_2\} \cup C$
$\quad$**until** $|\mathcal{C}| = 1$
$\quad$return $\mathcal{C}$

Time complexity:

- $O(n)$ iterations.
- $O(n^2)$ time to find the two clusters with minimum centroid distance.

Worst case time complexity is $O(n^3)$.

Speeding up HC

Ideas: using a Priority Queue to keep track of cluster distances, so that each iteration can be done in $O(n \log n)$ time.

- The total running time is $O(n^2 \log n)$. Much better than $O(n^3)$ but still far from the ideal $O(n)$ running time.

Speeding up HC

Ideas: using a Priority Queue to keep track of cluster distances, so that each iteration can be done in $O(n \log n)$ time.

- The total running time is $O(n^2 \log n)$. Much better than $O(n^3)$ but still far from the ideal $O(n)$ running time.

Details for implementing HC with priority queue Q:

- Initially, compute distances between every pair of points, put them all in the queue Q. This takes $O(n^2)$ time.

Speeding up HC

Ideas: using a Priority Queue to keep track of cluster distances, so that each iteration can be done in $O(n \log n)$ time.

- The total running time is $O(n^2 \log n)$. Much better than $O(n^3)$ but still far from the ideal $O(n)$ running time.

Details for implementing HC with priority queue Q:

- Initially, compute distances between every pair of points, put them all in the queue Q. This takes $O(n^2)$ time.
- In each iteration:
  - Fetch the smallest distance from Q, along with two corresponding clusters, say C_1, C_2. This takes $O(\log n)$ time.

Speeding up HC

Ideas: using a Priority Queue to keep track of cluster distances, so that each iteration can be done in $O(n \log n)$ time.

- The total running time is $O(n^2 \log n)$. Much better than $O(n^3)$ but still far from the ideal $O(n)$ running time.

Details for implementing HC with priority queue Q:

- Initially, compute distances between every pair of points, put them all in the queue Q. This takes $O(n^2)$ time.
- In each iteration:
  - Fetch the smallest distance from Q, along with two corresponding clusters, say C_1, C_2. This takes $O(\log n)$ time.
  - Delete all distances associated with C_1, C_2 from Q. There are at most $O(n)$ such distances. Thus, this takes $O(n \log n)$ time total.

Speeding up HC

Ideas: using a Priority Queue to keep track of cluster distances, so that each iteration can be done in $O(n \log n)$ time.

- The total running time is $O(n^2 \log n)$. Much better than $O(n^3)$ but still far from the ideal $O(n)$ running time.

Details for implementing HC with priority queue Q:

- Initially, compute distances between every pair of points, put them all in the queue Q. This takes $O(n^2)$ time.
- In each iteration:
  - Fetch the smallest distance from Q, along with two corresponding clusters, say C_1, C_2. This takes $O(\log n)$ time.
  - Delete all distances associated with C_1, C_2 from Q. There are at most $O(n)$ such distances. Thus, this takes $O(n \log n)$ time total.
  - Compute the distances from the new cluster to other clusters in $\mathcal{C}$, and put all such distances to Q. There are at most $O(n)$ new distances. Thus, this takes $O(n \log n)$ time total.

HC for non-Euclidean spaces

In non-Euclidean spaces, the notion of centroids may not be well-defined. But we can define *clustroids* instead. A clustroid of a cluster X is a point $x \in X$ that minimizes:

- *Sum of distances* to other points in X.
- Or *maximum distance* to other points in X.
- Or *sum of square of distances* to other points in X.

K-means[2]

$\text{K-MEANS}(\mathcal{D}, k)$

Choose k points $\{\mathbf{c}_1, \ldots, \mathbf{c}_k\}$ in $\mathcal{D}$ to be initial centroids

repeat

  for each point $p \in \mathcal{D}$

    Assign p to the cluster of the closest centroid.

  Let $C_1, \ldots, C_k$ be the clusters after the assignment.

  Recompute centroids $\mathbf{c}_i$ for each C_i, $1 \leq i \leq k$.

. **until** no assignment change

return $\{C_1, \ldots, C_k\}$

[2] This is slightly different from the algorithm presented in the MMDS book.

K-means[2]

$\text{K-MEANS}(\mathcal{D}, k)$

Choose k points $\{\mathbf{c}_1, \ldots, \mathbf{c}_k\}$ in $\mathcal{D}$ to be initial centroids
repeat
  for each point $p \in \mathcal{D}$
    Assign p to the cluster of the closest centroid.
  Let $C_1, \ldots, C_k$ be the clusters after the assignment.
  Recompute centroids $\mathbf{c}_i$ for each C_i, $1 \leq i \leq k$.
. **until** no assignment change
return $\{C_1, \ldots, C_k\}$

- How to initialize centroids?
- How to choose k?

[2] This is slightly different from the algorithm presented in the MMDS book.

Choosing Initial Centroids

Ideas: We want to pick centroids that likely belong to different clusters. There are two approaches:

- Pick points that are as far away from each other as possible.
- Sample a small set of points, apply (may be expensive) clustering algorithm, such as Hierarchical Clustering, to form k clusters on the sample set. Then choose centroids of the clusters.

Choosing Initial Centroids (Cont.)

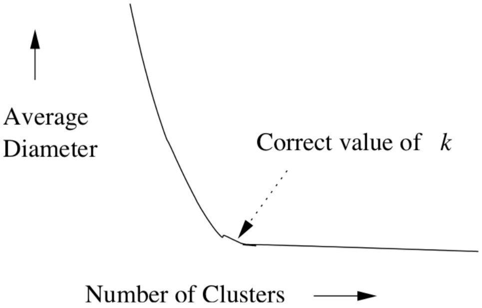
How to choose far way point set:

$\textsc{ChooseCentroids}(\mathcal{D}, k)$
    Choose a random point $\mathbf{c}_1 \in \mathcal{D}$
    $C \leftarrow \{\mathbf{c}_1\}$
    while $|C| < k$
        Choose $\mathbf{c} \in \mathcal{D} \setminus \{C\}$ that maximize $d(\mathbf{c}, C)$
        Add $\mathbf{c}$ to C
    return C

Recall:

$$d(\mathbf{x}, C) = \min_{\mathbf{c} \in C} d(\mathbf{x}, \mathbf{c}) \tag{6}$$

Choosing k



Time complexity

$\text{K-MEANS}(\mathcal{D}, k)$
    Choose k points $\{\mathbf{c}_1, \ldots, \mathbf{c}_k\}$ in $\mathcal{D}$ to be initial centroids
    repeat
        for each point $p \in \mathcal{D}$
            Assign p to the cluster of the closest centroid.
        Let $C_1, \ldots, C_k$ be the clusters after the assignment.
        Recompute centroids $\mathbf{c}_i$ for each C_i, $1 \leq i \leq k$.
.   **until** no assignment change
    return $\{C_1, \ldots, C_k\}$

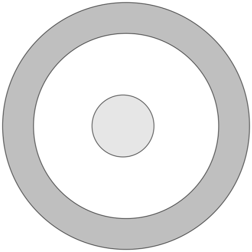
The time complexity is $O(NkdT)$ where:

- d is the dimension of the data.
- T is the number of iterations. Typically, $T \sim 100 - 1000$.

CURE

CURE (Clustering Using REpresentatives). It has two prominent properties:

- It can work with very large data.
- It can produce clusters of arbitrary shape.



CURE - 1st Phase

1. Sample the dataset and cluster it in main memory. It is advisable to use Hierarchical Clustering.
2. Choose a small set of *representative points* for each cluster. We should choose points that are far away from each other, as in initializing K-means.
3. Move each of the representative points by, say 20% distance from each point to its cluster centroid, along the line to the centroid.
4. Merge two clusters of their representative point set are close to each other. We can repeat this step until every clusters are sufficiently far from each other.

CURE - 1st Phase

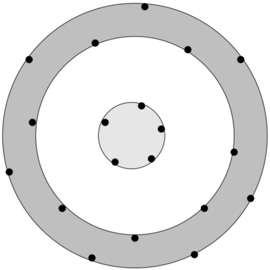


Figure: Choosing representatives

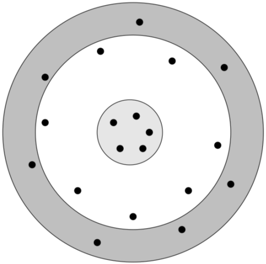


Figure: Moving representatives.

CURE - 2nd Phase

For each point $\mathbf{p}$ in the full dataset $\mathcal{D}$, assign to the cluster of *closest* representative points.