Clustering by Connectivity 1: Agglomerative

Yufei Tao

Department of Computer Science and Engineering
Chinese University of Hong Kong
The clusters found by centroid-based clustering (e.g., $k$-center and $k$-means) tend to have “ball shapes”.
Sometimes clusters may have arbitrary shapes, e.g.:

Which clusters make more sense? **It depends.** The next few slides contain a discussion about this.
Recall that, in classification, we were given a labeled dataset, namely, every point’s label was revealed. Finding a good classifier on such datasets is a form of **supervised learning**.

Opposite to this is **unsupervised learning**. Imagine, e.g., in classification, we are given an unlabeled dataset, where we do not know which points have label 0, and which points have label 1. How do we learn a classifier?
A good approach in this scenario is to do clustering. We can treat each cluster as a label, and thereby, get ourselves a “labeled” dataset, from which a classifier can be learned.

Now you may start to sense why we want to discover clusters of arbitrary shapes. Indeed, this is because a classification boundary may have an arbitrary shape!
This is a form of clustering that deviates significantly from centroid-based clustering. Specifically, “connectivity-based clustering” is built on the idea that points of a cluster ought to demonstrate a continuity of probabilistic density. Imagine you look at the dataset from a distance, and ask yourself: how many “clouds” do I see? Each “cloud” is a cluster.
Clustering by Connectivity

We will discuss two clustering methods under this category:

- Agglomerative clustering (this lecture) — also known as “hierarchical clustering”.
- Density-based clustering (the next lecture)
Given a set $P$ of $n$ objects, the agglomerative method works as follows:

1. At the beginning, each object in $P$ forms a cluster by itself.
2. Merge the two clusters that are most similar to each other.
3. Repeat the previous step until only one cluster is left.

The above framework can be instantiated in many ways depending on how cluster similarity is defined. Specifically, let $C_1$ and $C_2$ be two clusters, each being a set of objects. To measure their similarity, we need a function $d(C_1, C_2)$ such that the smaller the function's value, the more similar the two clusters.
Some common definitions for cluster similarity are:

\[
\begin{align*}
    d_{\text{min}}(C_1, C_2) &= \min_{o_1 \in C_1, o_2 \in C_2} \text{dist}(o_1, o_2) \\
    d_{\text{max}}(C_1, C_2) &= \max_{o_1 \in C_1, o_2 \in C_2} \text{dist}(o_1, o_2) \\
    d_{\text{mean}}(C_1, C_2) &= \frac{1}{|C_1||C_2|} \sum_{o_1 \in C_1, o_2 \in C_2} \text{dist}(o_1, o_2)
\end{align*}
\]

Among the three, \(d_{\text{min}}\) is the most popular—when this function is chosen, the agglomerative framework on the previous slide is known as the single linkage algorithm. We will focus on \(d_{\text{min}}\) in the rest of the lecture.
Example 1.

Execution of the agglomerative method using the $d_{\text{min}}$ metric:

1. Initially, 5 clusters: $\{a\}, \{b\}, \{c\}, \{d\}, \{e\}$.
2. Merging $\{d\}, \{e\} \Rightarrow \{a\}, \{b\}, \{c\}, \{d, e\}$.
3. Merging $\{a\}, \{b\} \Rightarrow \{a, b\}, \{c\}, \{d, e\}$.
4. Merging $\{c\}, \{d, e\} \Rightarrow \{a, b\}, \{c, d, e\}$.
5. Merging $\{c\}, \{d, e\} \Rightarrow \{a, b, c, d, e\}$.

The merging history of the algorithm can be represented as a tree (see above), which is called a dendrogram.
Think:

- How many merges are there in total if we have $n$ objects?
- Given a dendrogram, how would you obtain $k$ clusters quickly?

Next, we will explain that a dendrogram can be regarded as a minimum spanning tree. This naturally leads to an algorithm that computes a dendrogram in $O(n^2 \log n)$ time.
As before, let $P$ be the set of $n$ objects to be clustered. Define a distance graph $G(V, E)$ as follows:

- Every vertex of $V$ corresponds to a distinct object in $P$.
- $G$ is a complete graph, namely, there is an edge between each pair of vertices.
- The edge between vertex $o_1$ and $o_2$ carries a weight equal to $\text{dist}(o_1, o_2)$.

Let $T$ be a set of $n - 1$ edges of $G$. If $T$ induces no cycles, we say that $T$ is a spanning tree. Define $\text{cost}(T)$ to be the sum of the weights of all the edges in $T$. 

Example 2.

The figure on the right shows the distance graph. The set of red edges is a spanning tree with cost 28.5.
The agglomerative framework essentially produces a spanning tree.

**Example 3.**

The figure on the right shows the edges that the agglomerative algorithm uses to produce the dendrogram on Slide 10. Recall that the algorithm picks these edges in ascending order of weight.
Let $T^*$ be a spanning tree of the distance graph $G$. If for any other spanning tree $T$, it always holds that $\text{cost}(T^*) \leq \text{cost}(T)$, we say that $T^*$ is a minimum spanning tree (MST) of $G$.

**Lemma 4.**
The agglomerative framework returns a minimum spanning tree of $G$.

**Proof:** The algorithm works in the same way as the Kruskal’s algorithm, which is a well-known algorithm for finding an MST, and runs as follows. At the beginning, initiate an empty set $T$. At each step, among all the edges $e$ satisfying

- $e$ is not in $T$ yet;
- the addition of $e$ to $T$ does not create a cycle;

add to $T$ the one with the smallest weight. Repeat the step until $T$ has $n - 1$ edges.

Next we will prove that the algorithm indeed finds an MST.
**Proof (cont.):** Label the edges of $T$ as 1, 2, ..., $n$ in the order they are discovered by the algorithm (i.e., the edge with label $i$ is the $i$-th one discovered).

Let $T^*$ be an arbitrary MST of $G$. Let $t$ be the smallest integer such that the edge with label $t$ does not belong to $T^*$. If $t$ does not exist, then $T = T^*$, and we are done. Otherwise, denote that edge as $e$. Let $S$ be the set of edges with labels 1, 2, ..., $t-1$.

Now, add $e$ to $T^*$, which definitely gives a cycle. In this cycle, at least one edge — say $e'$ — does not belong to $S$ (otherwise, the entire cycle is in $S$, which is impossible because $T$ has no cycles). Observe that the weight of $e'$ cannot be smaller than that of $e$: otherwise, Kruskal’s algorithm would have used $e'$, instead of $e$ (notice that, the edges with labels 1, 2, ..., $t-1$ cannot form a cycle with $e'$ because, by definition, all those edges are in $T^*$).

We now obtain another MST $T'^*$ from $T^*$ by deleting $e'$ and adding $e$. Repeat the above argument using $T'^*$ — note that when we do so, the value of $t$ increases by 1.

With this, we complete the proof. □
Although not required in this course, it is worth mentioning that sub-quadratic time algorithms exist for computing a dendrogram in $d$-dimensional space where $d$ is a constant (for point objects and Euclidean distance). Specifically, the computation time is

$$O\left(\frac{n^2}{n^{\frac{d}{2}} + 1 - \epsilon}\right)$$

time, where $\epsilon$ can be an arbitrarily small constant. Interested students may refer to: