Lecture 21

Clustering

Supplemental reading in CLRS: None

Clustering is the process of grouping objects based on similarity as quantified by a metric. Each object should be similar to the other objects in its cluster, and somewhat different from the objects in other clusters.

Clustering is extremely useful; it is of fundamental importance in data analysis. Some applications include

- Scientific data from a wide range of fields
- Medical data, e.g., for patient diagnosis
- Identifying patterns of consumer behavior
- Categorizing music, movies, images, genes, etc.

Clustering is conceptually related to

- *Unsupervised learning* – the notion that objects which produce similar measurements may share some intrinsic property.
- *Dimensionality reduction.*

Depending on the application, we may have to carefully choose which metric to use out of many possible metrics. Or, we may have to apply a transformation to our data before we can get good clustering. But ultimately, the (admittedly vague) problem we are trying to solve is this:

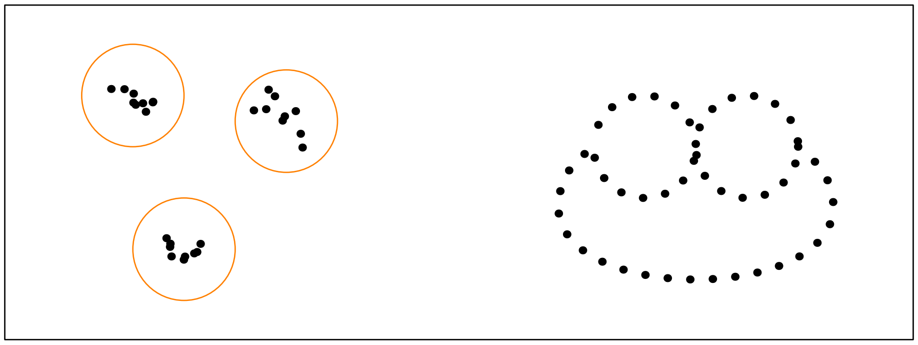

Figure 21.1. Left: Some data sets are relatively straightforward to cluster; most people would cluster these objects in basically the same way. Right: It's not always so easy, though. How would you make two clusters from this data set?

Input: Instance data $D = \{\vec{x}_1, \vec{x}_2, \dots, \vec{x}_n\}$
Desired number of clusters, k
Distance metric[1] $d(\vec{x}_i, \vec{x}_j)$

Output: Assignment of instance data D to clusters $\mathscr{C} = \{C_1, \dots, C_k\}$.

21.1 Hierarchical Agglomerative Clustering

The main idea of hierarchical agglomerative clustering is to build up a graph representing the cluster set as follows:

- Initially, each object (represented as a vertex) is in its own cluster.
- Each time an edge is added, two clusters are merged together.
- Stops when we have k clusters.

The following is an implementation of hierarchical agglomerative clustering known as *single-linkage clustering*.

```
Algorithm: SINGLE-LINKAGE-CLUSTER($D, d, k$)
  1. Let $H$ be an undirected graph with one vertex for each object and no edges.
  2. Sort the set of unordered pairs $\{\{u,v\} : u,v \in D,\ u \neq v\}$ by distance:
         $d(\text{pair } 1) \leq d(\text{pair } 2) \leq \cdots \leq d\left(\text{pair } \binom{n}{2}\right).$
  3. Loop from $i = 1$ to $\binom{n}{2}$:
       • If the two members of pair $i$ are in different connected components:
           – Merge their clusters.
           – Join pair $i$ with an edge in $H$.
       • Exit the loop and halt when there are only $k$ components left.
```

Although we have couched the description of this algorithm in the language of graph theory, the use of a graph data structure is not essential—all we need is a disjoint-set data structure to store the connected components. The advantage of computing the graph is that the graph gives us more information about the influence each object had on cluster formation.

21.1.1 Running time

In step 1, we initialize the graph and make n calls to MAKE-SET. Next, there are $\binom{n}{2} = \Theta(n^2)$ unordered pairs of objects; sorting them in step 2 takes $\Theta(n^2 \lg n^2) = \Theta(n^2 \lg n)$ time. Finally, each iteration of the loop in step 3 makes two calls to FIND-SET and at most one call to UNION. The loop is iterated at most $O(n^2)$ times.

Thus, the total number of operations performed on the disjoint-set data structure is $n + O(n^2) = O(n^2)$. If we use a good implementation of the disjoint-set data structure (such as a disjoint-set forest

[1] What we call here a "distance metric" corresponds to the notion of a *metric space* in mathematics. That is, our distance metric is a symmetric, positive-definite scalar-valued function of two arguments which satisfies the triangle inequality.

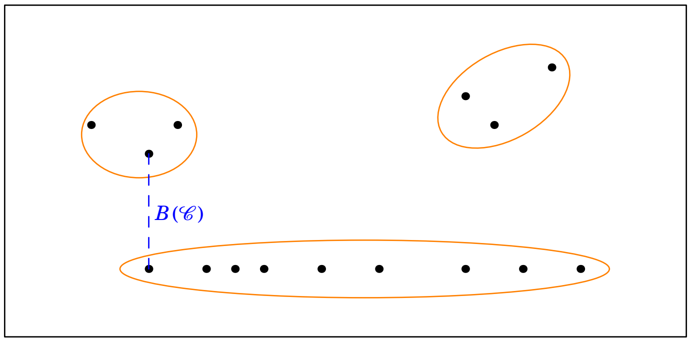

Figure 21.2. Illustration of the quantity $\beta(\mathscr{C})$, which is a measure of the amount of space between clusters.

with union by rank and path compression), these operations will take less time than the $\Theta\left(n^2 \lg n\right)$ sorting. Thus, the total running time of SINGLE-LINKAGE-CLUSTER is

$$\Theta\left(n^2 \lg n\right).$$

21.1.2 ~~Correctness~~ Discussion

This is where we would usually argue for the correctness of our algorithm. In the current case, there is no precise notion of correctness because we didn't state exactly what properties our clusters should have—just that there are k of them and they represent some sort of nearness-based grouping. So instead, we'll discuss the properties of this clustering and consider other possibilities.

- This algorithm is essentially a special case of Kruskal's MST algorithm. For $k = 1$, running SINGLE-LINKAGE-CLUSTER is exactly the same as running Kruskal's algorithm on the complete graph whose edges are weighted by distance.
- For $k > 1$, the graph produced is an MST with the $k - 1$ heaviest edges removed.
- This algorithm maximizes the "spacing" between clusters. More precisely, the minimum distance between a pair of clusters is maximized, in the sense that the quantity

$$\beta(\mathscr{C}) = \min\left\{d(\vec{x},\vec{y}) : \vec{x},\ \vec{y} \text{ not in the same cluster}\right\} \tag{21.1}$$

  is maximized (see Figure 21.2), as we show below.

Proposition 21.1. *Let $\mathscr{C} = \{C_1, \ldots, C_k\}$ be the output of* SINGLE-LINKAGE-CLUSTER. *Then the quantity $\beta(\mathscr{C})$, as defined in (21.1), is maximized.*

Proof. Let $d^* = \beta(\mathscr{C})$. Then d^* is the distance between the first pair of vertices *not considered* by the algorithm. All edges in the graph H *were* considered, so their weights are all $\le d^*$.

Consider a different clustering $\mathscr{C}' = \left\{C'_1, \ldots, C'_k\right\}$. There must exist some $C_r \in \mathscr{C}$ which is not a subset of any $C'_s \in \mathscr{C}'$. (Make sure you can see why.[2]) So we can choose some $\vec{x}, \vec{y} \in C_r$ and some s such that $\vec{x} \in C'_s$ and $\vec{y} \notin C'_s$. Because C_r is a connected component of H, there exists a path $\vec{x} \rightsquigarrow \vec{y}$ in

[2] What would it mean if every C_r were a subset of some C'_s? This should not sit comfortably with the fact that $\mathscr{C}' \neq \mathscr{C}$.

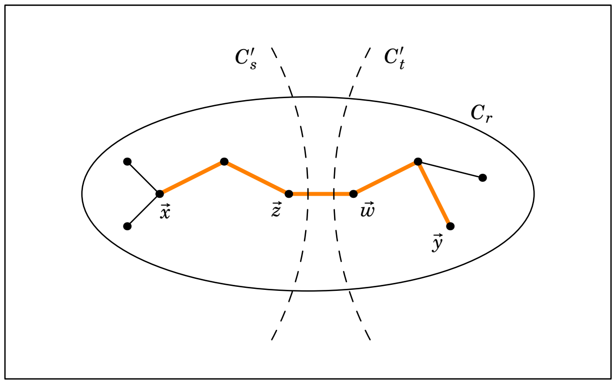

Figure 21.3. Illustration of the proof of Proposition 21.1.

H. Somewhere in that path there must be an edge $(\vec{z},\vec{w})$ which connects a vertex in C'_s to a vertex in some other cluster C'_t (see Figure 21.3). Thus

$$\beta(\mathscr{C}') \le \left(\text{distance between } C'_s \text{ and } C'_t\right) \le d(\vec{z},\vec{w}) \le d^*,$$

since $(\vec{z},\vec{w})$ is an edge in H. $\square$

21.1.3 Other distance criteria

The procedure SINGLE-LINKAGE-CLUSTER is designed to always merge together the two "closest" clusters, as determined by the distance criterion

$$d_{\min}\left(C_i,C_j\right) = \min_{\vec{x}\in C_i,\vec{y}\in C_j} d(\vec{x},\vec{y}).$$

Other forms of hierarchical agglomerative clustering use different distance criteria, such as[3]

$$d_{\max}\left(C_i,C_j\right) = \max_{\vec{x}\in C_i,\vec{y}\in C_j} d(\vec{x},\vec{y})$$

$$d_{\text{mean}}\left(C_i,C_j\right) = \frac{1}{|C_i|\cdot|C_j|}\sum_{\vec{x}\in C_i,\vec{y}\in C_j} d(\vec{x},\vec{y})$$

$$d_{\text{centroid}}\left(C_i,C_j\right) = d\left(\frac{1}{|C_i|}\sum_{\vec{x}\in C_i}\vec{x},\ \ \frac{1}{|C_j|}\sum_{\vec{y}\in C_j}\vec{y}\right).$$

Note that new algorithms are needed to implement these new criteria. Also, different criteria tend to produce quite different clusters.

In general, hierarchical agglomerative clustering tends to perform worse on higher-dimensional data sets.

[3] The last of these, d_{centroid}, assumes that the points $\vec{x}$ belong to a vector space (hence, can be added together and scaled). If you care about this sort of thing, you probably already noticed my choice of notation $\vec{x}$ which suggests that the ambient space is $\mathbb{R}^m$. This is the most important case, but d_{centroid} makes sense in any normed vector space. (And the rest of this lecture makes sense in an arbitrary metric space.)

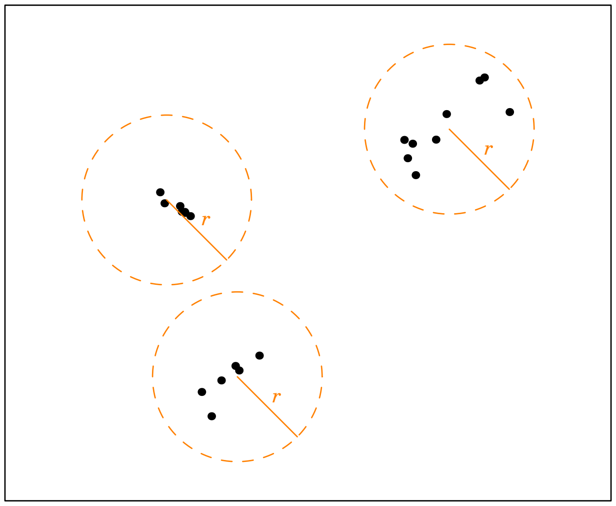

Figure 21.4. Given a positive integer k (in this case $k = 3$) and a desired radius r, we attempt to create k clusters, each having radius at most r.

21.2 Minimum-Radius Clustering

Hierarchical agglomerative clustering focuses on making sure that objects in different clusters are far apart. By contrast, minimum-radius clustering focuses on making sure that objects in the same cluster are close together. Thus, rather than maximizing the minimum distance $\beta(\mathscr{C})$, we will now try to minimize a maximum (actually, maximin) distance. Namely, we will try to assign a "center" $C.center$ to each cluster $C \in \mathscr{C}$ so as to minimize the quantity

$$F(\mathscr{C}) = \max_{\vec{x} \in D} \min_{C \in \mathscr{C}} d(\vec{x}, C.center).$$

In principle, given k, we need only find the best possible set of k centers—then, each vertex will belong to the cluster centered at whichever of these k points is closest to it. Thus, we can view the quantity $F(\mathscr{C})$ which we are trying to minimize as depending only on the choice of centers.

The problem of minimizing $F(\mathscr{C})$ in this way is quite difficult. We will make things easier on ourselves by assuming that we are given a goal radius r to strive for (rather than trying to figure out what the minimum possible radius is) (see Figure 21.4).

Input: Instance data $D = \{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n\}$ with a distance metric $d\left(\vec{x}_i, \vec{x}_j\right)$
Desired number of clusters, k
Desired radius, r

Output: Ideally, a set of k points $\Gamma = \{\vec{c}_1, \ldots, \vec{c}_k\} \subseteq D$ (the cluster centers) such that each $\vec{x} \in D$ is within distance r of some element of Γ.

21.2.1 First strategy: Approximate k

The following algorithm gives the correct r but approximates k: assuming a clustering with the desired k and r is possible, the number of clusters is at most $k(\ln n + 1)$.

Algorithm:
Imagine each $\vec{x} \in D$ as the center of a cluster of radius r. The set

$$S_i = \{\text{points in } D \text{ within distance } r \text{ of } \vec{x}_i\}$$

consists of all points which should be allowed to belong to this cluster. We now face the following problem: we have a collection of sets $S_1, \ldots, S_n \subseteq D$, and we wish to find a size-k subcollection $S_{i_1}, \ldots, S_{i_k}$ such that

$$\bigcup_{j=1}^{k} S_{i_j} = D.$$

This is the set-cover problem, which is NP-hard. However, we can obtain an approximate solution using the approximation algorithm of §18.2:

- Greedily choose the set with the largest coverage until all points are covered.
- Assuming a set cover exists, this algorithm gives a multiplicative α-approximation, where $\alpha = \ln|D| + 1$.

The running time of this algorithm is

$$O\left(\sum_{i=1}^{n} |S_i|\right),$$

which, depending on the geometric configuration of D, could be anywhere from $O(n)$ to $O\left(n^2\right)$.

21.2.2 Second strategy: Approximate r

Alternatively, we could strictly meet the k requirement but approximate r. Assuming that a clustering with the desired k and r is possible, the following algorithm produces a set of k clusters with radius at most $2r$.

Algorithm:

```
1  $i \leftarrow 0$
2  while $D$ is not empty do
3      $i \leftarrow i + 1$
4      Pick an arbitrary $\vec{x} \in D$
5      Define cluster $C_i$ to be all points within distance $2r$ of $\vec{x}$
6      $C_i.center \leftarrow \vec{x}$
7      $D \leftarrow D \setminus C_i$
8  return $\{C_1, \ldots, C_i\}$
```

The costliest line is line 5, which takes $O(n)$ time. As we show below in Lemma 21.2, line 5 is executed at most k times (assuming that a clustering with the desired k and r is possible). Thus, the running time of this algorithm is $O(kn)$.

Lemma 21.2. *Suppose that a clustering with the desired k and r is possible. Then, after k passes through the loop, D will be empty.*

Proof. Let $\mathscr{C}^* = \{C_1^*, \ldots, C_k^*\}$ be a clustering with the desired k and r. Let $\vec{x}$ be as in line 4, and let $C_j^* \in \mathscr{C}^*$ be such that $\vec{x} \in C_j^*$. Let $\vec{c} = C_j^*.center$. We claim that C_i (as defined on line 5) is a superset of $C_j^* \cap D$. (The intersection with D is necessary because D may have changed during prior iterations of line 7.) To prove this, note that for any $\vec{y} \in C_j^* \cap D$, we have by the triangle inequality

$$d(\vec{x}, \vec{y}) \leq d(\vec{x}, \vec{c}) + d(\vec{c}, \vec{y}) \leq r + r = 2r.$$

Thus $\vec{y} \in C_i$, which means that $C_j^* \cap D \subseteq C_i$.

Thus, each time line 7 is executed, a new element of $\mathscr{C}^*$ is removed from consideration. By the time line 7 has been executed k times, all k of the clusters in $\mathscr{C}^*$ will have disappeared, so that D is empty. $\square$