Clustering - Grouping objects based on similarity as quantified by a metric

- Objects sufficiently similar to one another are assigned to the same cluster
- Objects in different clusters are further apart

Some data is relatively straightforward and different people would cluster them similarly.

How would you make 2 clusters from this?
Useful — Very common data analysis tool
- Scientific data from wide range of fields
- Medical Data — Patient diagnosis
- Identify patterns of consumer behavior
- Categorize music, movies, images, genes, etc

Conceptually related to
- Unsupervised learning — notion that objects that produce similar measurements may share some intrinsic property
- Dimensionality reduction

For some application specialized distance metrics and/or data transformations may be required.

Problem Statement:

Given: Instance data \( D = \{ \vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n \} \)

Number of desired clusters, \( k \)

Distance metric \( d(\vec{x}_i, \vec{x}_j) \)

Output: Assignment of instance data \( D \) to clusters \( \{ C_1, \ldots, C_k \} \)
Hierarchical Agglomerative Clustering

Main Idea - Progressively grow graph representing cluster set.

- Initially each object (vertex) in own cluster
- Add edges to create one new cluster by joining two previous clusters
- Stop when have k clusters

Algorithm - also called single-linkage clustering

1) \( H \) is a graph with one vertex for each object and no edges
2) Compute distances between all pairs of objects
3) Sort pairs of objects by distance
   \[ d(u_i, v_i) \leq d(u_2, v_2) \leq d(u_3, v_3), \ldots \leq d(u_m, v_m) \]
4) Loop over \( i = 1 \) to number of distances \( m \)
   - If \( u_i \) & \( v_i \) in different clusters (different connected components)
     \[ \rightarrow \] Merge their clusters
     \[ \rightarrow \] Insert the edge \((u_i, v_i)\) into \( H \)
   - Exit loop and stop when only \( k \) clusters left
Running Time

Step 1 - Initialize data structure → \( n \) MAKE-SETs
Step 2 - all distances \( \Theta(n^2) \)
Step 3 - sort \( \rightarrow \Omega(n^2 \log n^2) = \Omega(n^2 \log n) \)
Step 4 - loop \( n^2 \) times
\[
\frac{n(n-1)}{2} \quad 2 \text{ FIND-SETs} \\
\quad 1 \text{ UNION} \\
\]
Overall \( \Theta(n^2 \log n) \)

Correctness

Usually we argue correctness here. In this case we didn't state exactly the properties our clusters should have, except that there are \( k \) of them and that they represent nearest-distance based grouping. So here we'll discuss instead the properties of this clustering and consider other possibilities.

For \( k=1 \), this is exactly Kruskal's MST algorithm (except here we compute all distances and there we are given edge weights).

For \( k>1 \), we produce a graph that is an MST with \( k-1 \) costliest edges (cheapest edges) removed.
**Example**

```
\[ \text{Example graph: shortest distance between two clusters in final graph.} \]
```

**Claim:** Output is a $k$-clustering of maximum "spacings".

**Proof**

Observe that Output $\mathcal{C} = \langle C_1, \ldots, C_k \rangle$ has spacings $d^*$, which is distance of first pair not considered by algorithm. Observe further that all edges in $H$ appeared earlier in the heap and so have distances $\leq d^*$.

Consider different clustering $\mathcal{C}' = \langle C'_1, \ldots, C'_k \rangle$

$\Rightarrow \exists C_r$ that is not a subset of any $C'_s$ in $\mathcal{C}'$

$\& \exists \pi, \beta \in C_r$ such that $\pi \in C'_s \neq \beta \in C'_r$
Output adds $p_i \rightarrow p_j$ (orange) path to $H$ before stopping, so each orange edge $e \leq d^*$ \implies d(p_i, p_j) \leq d^*$, so spacing of $C^i \leq d^*$.

\[\therefore\] any other clustering with $k$ clusters has smaller or equal spacing.

---

Alternative forms of Hierarchical Agglomerative Clustering vary the distance criterion used to find closest clusters to merge.

This is the example we have been discussing.

\[\begin{align*}
\text{MIN:} & \quad d_{\text{min}}(C_i, C_j) = \min_{x \in C_i, y \in C_j} d(x, y) \\
\text{MAX:} & \quad d_{\text{max}}(C_i, C_j) = \max_{x \in C_i, y \in C_j} d(x, y) \\
\text{MEAN:} & \quad d_{\text{mean}}(C_i, C_j) = \frac{1}{|C_i| \cdot |C_j|} \sum_{x \in C_i, y \in C_j} d(x, y) \\
\text{CENTROID:} & \quad d_{\text{centroid}}(C_i, C_j) = d\left(\frac{1}{|C_i|} \sum_{x \in C_i} x, \frac{1}{|C_j|} \sum_{y \in C_j} y\right)
\end{align*}\]

\[\rightarrow\] Other algorithms needed for other criteria

\[\rightarrow\] These other criteria tend to produce different clusters

\[\rightarrow\] Hierarchical Agglomerative Clustering tends to perform worse for higher-dimensional datasets.
A Different Approach: Min-Radius Clustering

Given: 1) Set of points \( r \in V \) with \( V \subseteq \mathbb{R}^m \) so triangle inequality satisfied
2) Number of clusters \( k \), cluster radius \( r \)

Find: Subset of points \( C \subseteq V \) [cluster centers] of size \( k \) s.t. \( \forall r \in V \) is within distance \( r \) of some element of \( C \).

\[
F[C] = \max_{r \in V} \min_{c \in C} d(r, c)
\]

Cost of clustering according to \( C \)

Single point most distant from its cluster center
Idea:

Imagine each \( v \in V \) as the center of a cluster, so \( S_v \leftarrow \) points in \( V \) within distance \( r \) of \( v \).

From this collection of sets \( S_1, S_2, \ldots, S_n \) can we find a subcollection of size \( k \) \( C_1, C_2, \ldots, C_k \) s.t. \( \bigcup_{j=1}^{k} S_j = V \) ?

This corresponds to Set Cover, which is NP-hard.

So does this mean \( k,r \)-clustering is NP-hard?

Not necessarily, as the reduction is in the wrong direction (but it is known to be NP-hard).

However, we can use approximate algorithm for Set Cover [Lecture 18] for approximate \( k,r \)-clustering.

- Greedily choose set with largest coverage until all points covered
- \( d \)-approximate \( d = \left( \log n + 1 \right) \# \text{ of sets} \)
- Running time depends on data structure

If there is a set cover of size \( k \), our output will be of size \( \leq k \left( \log n + 1 \right) \)

So this algorithm gives correct \( r \) but approximates \( k \).
Alternatively, could strictly meet $k$ requirement but approximate.

Assuming points in $V$ are $k,v_r$-clusterable, then this algorithm outputs a $k,v_r$ cluster.

$i \leftarrow 0$

While $V$ not empty:

$i \leftarrow i + 1$

pick arbitrary $v \in V$

define cluster $C_i$ be all points within $2r$ of $v$

$V \leftarrow V - C_i$

Output $C_1 ... C_i$

Runtime: Time to compute all distances $O(n^2)$

Claim: After $k$ passes through loop, $V$ will be empty

Corollary: $O(nk)$ run time by only computing distances needed

Proof of Claim: Let $G^* = \{C_1^*, ..., C_k^*\}$ be $k,v_r$ clustering with centers $c_1, ..., c_k$. For each $C_i^*$ and any 2 points $u,v \in C_i^*$

d(u,v) \leq d(u,c_i) + d(c_i,v) \leq 2r$

Once algorithm picks a $v \in C_i^*$, all other points in $C_i^*$ will go into the same cluster. In each round will pick point from new cluster ⇒ After $k$ rounds, no points left.