Hierarchical clustering

dendrogram
construct a hierarchy/tree of clusters that relates the points in our dataset
go from bottom to top
based on a certain distance between points (y axis), you'll be able to merge a couple points, it'll tell you which points to merge
first merge 1 and 2, then 3 and 4, then 1 2 with 5, then 3 4 with 6, then all together
constructing this hierarchy provides some insight into the data

- ex: analyzing dna of different species
    - each is a different species, 3 and 4 is one species, 1 2 5 is another, but they're all one kingdom or something

we can cut the dendrogram at any point and generate a number of clusters

- deciding different thresholds generates different number of clusters

how do we get this kind of hierarchical clustering?

agglomerative

- start with every point in its own cluster
- at each step, merge the two closest clusters
- that's how we looked at the dendrogram, from the bottom up

divisive

- start with every point in the same cluster
- we'll split until every point is in its own cluster
- we would read the dendrogram from the top down

we'll mostly look at agglomerative because they're a bit simpler

algorithm for agglomerative clustering

- start with each point in its own cluster
- compute the distance between all pairs of clusters
- merge the two closest clusters
- now we have one fewer cluster than we had before
- repeat second and third steps until all points are in the same cluster

what are we missing?

- we defined distances between points, but it's unclear what we mean by distance between clusters what are some ideas for taking distances between two clusters?

- compute the distance between the centers of the clusters
- use the sum of the distances of all possible pairs of points in one and another cluster
- compute the distance between the means of different clusters, centroids
- take the shortest path, the two points that are closest together in the two clusters
- take the maximum distance
- all of these are distance functions we will look at

let's first define:

distance between points: d(p1, p2)
distance between clusters: D(C1, C2)

single-link distance is the minimum of all pairwise distances between a point from one cluste ra nd a point from the other cluster
$D_{SL}(C_1, C_2)$ = min {$d(p_1, p_2)$ | p_1 in C_1, p_2 in C_2}

We're taking the set of all pairs of points for which one is in cluster 1 and the other is in cluster 2, and we're finding the pair from that set with the minimum distance between the two points.

this is advantageous because we can handle clusters of different sizes, whereas k-means struggles to do that

however, if there is overlap between the clusters, it doesn't do so well because all the points in the intersection get jumbled up and all show up as the min distance

complete-link distance is the maximum of all pairwise distances between a point from one cluster and a point frmo th either cluster

just like min distance but with max instead

does a better job at handling the noise points
tends to create more balanced, but tends to split up large clusters

average-link distance is the average of all pairwise distances between a point from one cluster and a point from the other cluster

$D_{AL}(C_1, C_2)$ = 1 / $|C_1|$*$|C_2|$ Sum of all p_1 in C_1, p_2 in C_2 $d(p_1, p_2)$

We multiply the cardinalities of C_1 and C_2 in the denominator because there are that many pairs of points p_1, p_2 where p_1 in C_1 and p_2 in C_2

less susceptible to noise and outliers, but tends to be biased toward globular clusters

Centroid distance is the distance between the centroids (means) of the clusters.

Ward's distance is the difference between the spread / variance of points in the merged cluster and the unmerged clusters

$D_{WD}(C_1, C_2)$ = variance of merged cluster - variance of cluster 1 - variance of cluster 2

we are trying to compare the spread of points in each cluster vs the spread of points if they were merged together. if the difference is high vs if the difference is low will tell us about the distance between the clusters because the spread is the distance to the mean.

for the example, look at how the dendrogram is drawn when we merge points
also, note how we update the distances in the distance matrix when we form a cluster because we now need to find the distances between points and clusters

finding the threshold with which to cut the dendrogram requires exploration and tuning, but in general hierarchical clustering is used to expose a hierarchy in the data
to capture the difference between clusterings you can use a cost function, or methods that we will discuss later when we look at clustering aggregation

we can't use the cost function for k-means for our hierarchical method, so we'll need some other way to evaluate which clustering to choose

Density-based clustering

goal: cluster together points that are densely packed together

how should we define density?

not just that they're close, but that they're close and there has to be a lot of points

- in this small location, there are this many points

given a fixed radius epsilon around a point, if there are at least min_pts number of points in that area, then this section is dense

- epsilon-neighborhood of point

not every point in a dense region will be dense itself

what db scan does is it distinguishes between points that are at the core of a dense region and points that are at the border of a dense region

core point: if its epsilon-neighborhood contains at least min_pts

border point: if it is in the epsilon-neighborhood of a core point

noise point: if it is neither a core nor border point

we are able to label all our data according to core, border, or noise
then, we can scan all the core points and put them all together in one cluster, then attach border points to the closest cluster or randomly assign them to a cluster
this is neat because we can handle clusters of different shapes and sizes, whereas k-means would not be a good method for that

DBScan Algorithm

inputs to this algorithm are 2d parameters, which we will usually have and are parameters that we need to tinker with

epsilon and min_pts given:

- find the epsilon-neighborhood of each point
- label the point as core if it contains at least min_pts
- label points in its neighborhood that are not core as border
- label points as noise if they are neither core nor border
- for each core point, assign to the same cluster all core points in its neighborhood

- assign border points to nearby clusters

might be a little abstract, how would we do this? labelling each point and computing epsilon-neighborhoods and such

DBScan benefits
- can identify clusters of different shapes and sizes
- resistant to noise

Limitations
- can fail to identify clusters of varying densities
- tends to create clusters of the same density
- notion of dxensity is problematic in high-dimensional spaces