**Hierarchical Clustering**

Clustering involves grouping similar things together. This Hierarchical Clustering thing seems to be a way of exploring how we might group elements in a list, to see what clusters, if any, they might naturally fall into. The list of features of an object are called its feature vector (uh, where did I get these features?).

Table

Description automatically generated with medium confidence

There are different choices of scale in each column - can see that all columns go from 0 to 1 except legs which is 0 to 4. Sometimes it’s better to make the columns’ scales comparable in some way. One is **interpolation**, which linearly rescales values between the min and max. Another is **Z-scaling** which is a linear rescaling to a set of #’s with some given mean and standard deviation (often 0, 1, hence name).



We can see the importance of scaling in this example. Consider following data of three clusters (identified by marker type) and the hierarchical clustering algorithm run on it. And we’ll tell it to group into three clusters.

A diagram of a diagram

Description automatically generated A diagram of a number of red and green dots

Description automatically generated

So the algorithm identified red, blue and green clusters. Red looks okay, but we would’ve separated the blue and green differently, into vertical, not horizontal groups. It did it this way, however, because of the x1 scale vs. the x2 scale. It’s trying to minimize the distance between (see below) clusters as calculated via the “complete” method, which is to at each stage combine two groups whose max distance between respective members is the smallest of all pairs of groups. And since the x1 scale is so large, the variation in the x1 coordinate almost wholly accounts for this distance present. The variation in x2 is so small, in comparison, that we might as well just flatten all the data onto the x1 axis and have no x2 axis at all. In that case, it makes sense that if we are to split the data into three groups, we would split where we would (I think? hope?). Furthermore, from the dendrogram, we can also see that n = 2 is probably the most natural number of groups. But clearly there should really be three groups present. We can see this because when we look at the graph, we aren’t paying attention to the absolute numbers along the axes, just the relative distances. We can achieve this numerically by normalizing the data along each axis. When we do, we get what we were expecting rather.

A diagram of a diagram

Description automatically generated A diagram of a number of red and blue dots

Description automatically generated

Furthermore, we see that n = 3 groups is most natural when scaled properly, as we expect.

*Applications*

We can use clustering in a lot of different ways. We can take an image and flatten it to get a 1D array of 5 values (x, y, r, g, b), i.e., the pixels location in the 2D array, and its three color values. We can consider these features. Then we can run our clustering algorithm to locate pixels which are connected in space and close in color. Often pixels related in this way will be from a common thing, like a body of water, or a face (as opposed to clothing), etc.

**Hiearchical Clustering Algorithm (one version)**

Say we have some objects plotted in a d = 2 feature space. Hierarchical clustering works as follows.

Chart, scatter chart

Description automatically generated

We find the pair closest together (and distance can be measured using Euclidean metric, or Manhattan metric, etc.). Manhattan distance is d = |x2 – x1| + |y2 – y1| + |z2 – z1| + |w2 – w­1|, etc. This is also called cityblock distance. It’s basically the distance you’d have to walk to get to two points in a (d-dimensional) city with grid layout.

Chart, scatter chart

Description automatically generated

And group them in a cluster, centered at the average of their two positions.

Chart, scatter chart, bubble chart

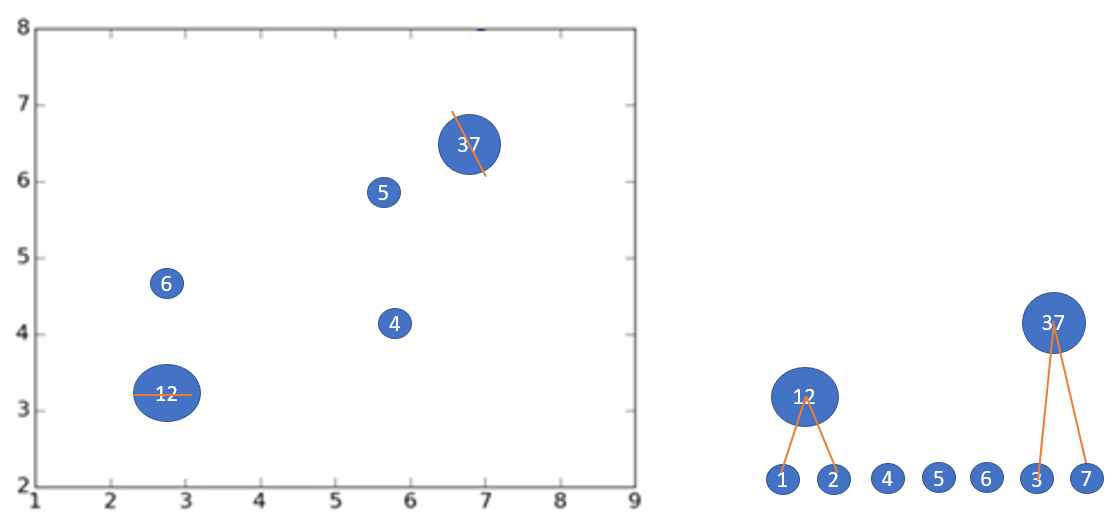
Description automatically generated

Then we repeat. Find the nearest two points (including clusters). These are 7 and 3. And we will move 3 to be contiguous with 7 (or vice versa). We would do this to the two corresponding row entries in the table as well.

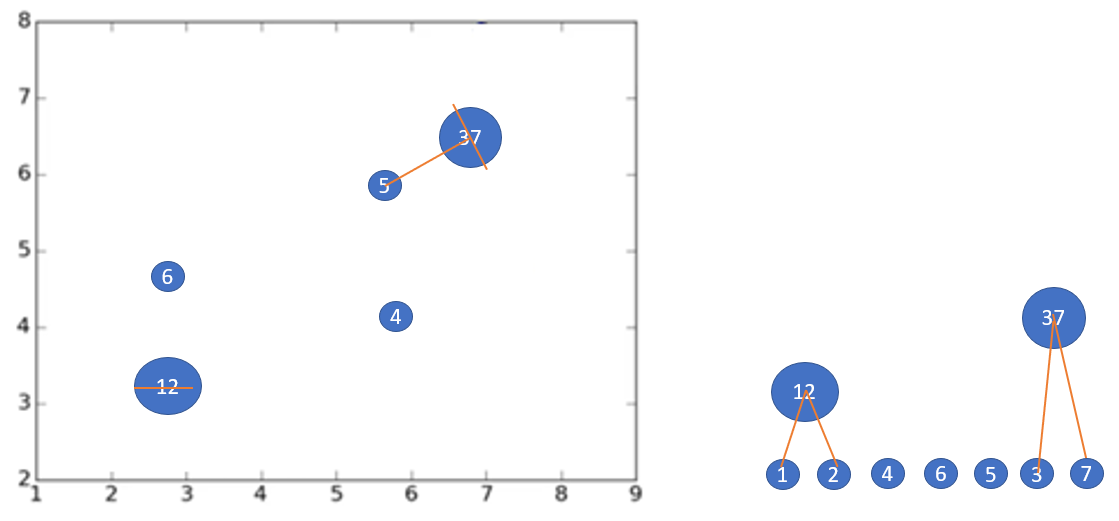
Chart, scatter chart

Description automatically generated

And group into a cluster, centered at the average of the two coordinates.



And then find the nearest neighbors again. This is 5, and we’lll move it to be contiguous with its group.



And join together, at the average coordinate. Note, when doing the average, I mean the average coordinate of all three points in the cluster. Unfortunately, below, it looks like I drew it in such a fashion that it appears that the centroid of cluster (375) is just the average of the positions of point (5), and cluster (37), when it fact it would be the average of points (5), (3) and (7) individually. So keep that in mind below, and elsewhere. Also, joining based on the criterion of minimum distance between centroids is just one way to group into clusters. There are other ways spelled out in scipy notes, like joining based on minimum smallest distance between two points of the different clusters, or minimum largest distance between two points of the different clusters, or minimum average distance between all the points of the two clusters, etc.

Chart, scatter chart, bubble chart

Description automatically generated

Finding closest neighbors again. This time it’s 6 with the 12 cluster. So we move the 6 to be next to 12.

Chart, scatter chart, bubble chart

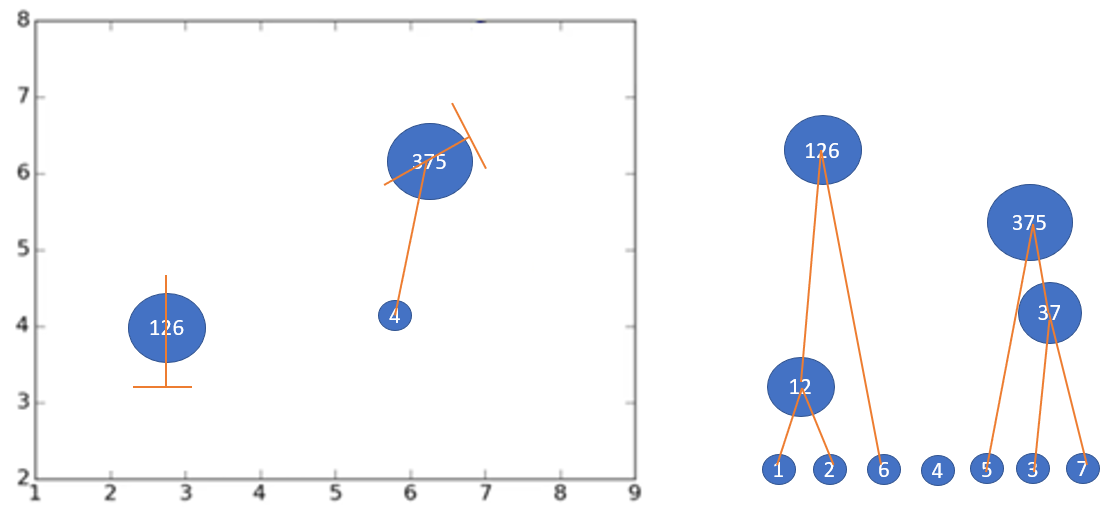
Description automatically generated

And joining.

Chart, scatter chart, bubble chart

Description automatically generated

Closest neighbors.



And joining.

Chart

Description automatically generated

Closest neighbors.

Chart

Description automatically generated

And joining.

Chart, bubble chart

Description automatically generated

Taking a look at the dendrite tree or whatever it’s called,

Diagram

Description automatically generated Chart, scatter chart

Description automatically generated

Might observe that if we were to try to group the data into 2 clusters (two intersections of horizontal line with dendritic lines), which we have by step 5, the groups (126), (3754) would indeed constitute the most natural partitioning. And if we were to group into 3 clusters, which we have by step 4 (three intersections of horizontal line with dendritic lines), the groups (126), (4), (375) would also be the most natural partitioning. So we can use hierarchical clustering to explore what groups/clusters our data might be congregating into. We could think of the ‘y’ axis, along which the numbers 1, 2, 3, 4, 5 are running, as representing the distance between clusters when joined.

**Liabilities**

One last note! Hierarchical clusters takes time O(n2), where n is number of samples. So not the best for large datasets.

**Example**

I created a (better) dataframe of animal features,

A number of scales with black text

Description automatically generated with medium confidence

and scaled the Legs feature to between (0, 1),

A number of scales and other symbols

Description automatically generated with medium confidence

and then created a hierarchical clustering diagram in scipy,

A diagram of a cluster of animals

Description automatically generated

The distance on the y-axis is the distance between clusters when joined. Can see at the level of distance = 2 (I drew a horizontal line at this height for visual clarity), we have two clusters left. Used scipy to output the identity clusters each row/animal belonged to at this height:

A number on a white background

Description automatically generated

and appended it to the dataframe,

A list of blood type

Description automatically generated with medium confidence

Can see this matches with the diagram.

**Exploring the Model and Hyperparameters**

Here’s a set of 2D data.

x1 x2 marker

0 -5.581006 2.453801 +

1 -4.775744 6.493566 +

2 -2.498410 3.883308 +

3 -7.721780 4.778397 +

4 -4.800134 2.625327 +

5 -5.095982 2.983548 +

6 -5.712459 3.620277 +

7 -7.177118 4.828715 +

8 -5.702035 6.645507 +

9 0.176800 6.832048 +

10 -6.011302 8.433696 +

11 -6.168618 6.038520 +

12 0.657432 -4.065840 o

13 -0.835175 -4.746274 o

14 -2.262751 -3.368424 o

15 -2.410987 -4.609493 o

16 -5.672407 -3.274687 o

17 -5.086841 -5.121301 o

18 -6.273988 -4.602747 o

19 -0.343871 -3.356554 o

20 -2.165032 -3.279266 o

21 -1.564511 -4.390938 o

22 -0.995685 -6.681225 o

23 -3.409054 -1.810766 o

24 1.648725 -0.563825 ^

25 3.404706 -1.865709 ^

26 2.892774 0.711521 ^

27 2.889918 -1.585560 ^

28 4.892444 2.111042 ^

29 5.665094 3.221986 ^

30 3.398538 1.512025 ^

31 2.619147 2.509115 ^

32 3.591971 -2.887745 ^

33 4.067365 1.741439 ^

34 5.401179 1.658608 ^

35 2.007351 -0.467921 ^

I markered the points according to their classes. And the class coordinates were taken from a normal distribution with a mean (centroid) and standard deviation (different for each class). Then I colored the points according to their classes as well, and plotted below:

A diagram of a plot of classes

Description automatically generated

The plots *below* are done with scipy, using the mergings = linkage(X, metric = “metric”, method = “method) and fcluster(mergings, t=#, criterion = “distance”) functions. And of course I’ll be varying the metric and method arguments, and t as well, but keeping the criterion argument set to “distance”. And I’ll have it color the points according to their predicted classes, while keeping the markers the same so we can tell which points ‘actually’ belong together.

**Hyperparameter: metric = “euclidean”, “minkowski”, “cityblock”, etc.**

The default metric is euclidean. And this tells the computer how to measure distance between points. Not sure what “minkowski” is, ‘cause I don’t think it’s the space-time metric. Anyway, I’ll keep method set to “single” (the default) and explore different metrics. So if we were to split into 2 groups, this is what we’d get:

A diagram of a city

Description automatically generated A diagram of a graph

Description automatically generated

This is how it’d do three groups.

A diagram of a city

Description automatically generated A diagram of a graph

Description automatically generated

And this is four groups.

A diagram of a city

Description automatically generated A diagram of a graph

Description automatically generated

Case could be made, visually, for the four group guy. Finally, this is six groups.

A diagram of a graph

Description automatically generated A diagram of a graph

Description automatically generated

Interestingly, the ‘correct’ grouping is not an option here. And I’d say no particular classification scheme really stands out, based on the dendrogram. I see I haven’t used any other metrics. Here’s cityblock for three groups.

A diagram of a city

Description automatically generated A diagram of a graph

Description automatically generated

Can see we still don’t get the ‘correct’ grouping, no matter the metric, even though the groupings we have gone are fairly plausible. So apparently, we need to change the method. That’s next section. But first, I’ll four groups with cityblock,

A diagram of a city

Description automatically generated A diagram of a graph

Description automatically generated

So we’re getting the same results as we did with metric = euclidean.

**Hyperparameter: method = “single”, “complete”, “average”, “centroid”, “median”, “ward”**

The default method is single. “single”, “average”, “complete” treats the distance between clusters as the min, average, max distances between points in the respective clusters. “centroid”, “median” treats the distance as that between the clusters’ centroids, or medians. “ward” treats as something else. I think it treats the ‘distance’ between clusters as the variability those two clusters would have, if they were joined into one (see Kmeans notes). And at any stage, the clusters with the smallest values of these things will be joined. Let’s do ‘complete’, for three groups.

A diagram of a diagram

Description automatically generated A diagram of a graph

Description automatically generated

So this gives us our desired grouping. And from the dendrogram, it looks like three groups is really the prefered grouping, as its legs are by far the longest. Let’s do “average” and three groups.

A diagram of a graph

Description automatically generated A diagram of a number of numbers

Description automatically generated

This also returns the desired grouping, although the case for three is less convincing from the dendrogram. Now let’s do centroid,

A diagram of a diagram

Description automatically generated A diagram of a number of numbers

Description automatically generated

And get same results. Let’s do median:

A diagram of a diagram

Description automatically generated A diagram of a graph

Description automatically generated

and ward,

A diagram of a diagram

Description automatically generated A diagram of a graph

Description automatically generated

ward makes a very strong case for three groups. So if we were to judge solely from all the dendrograms we’ve created, it would seem to be that we’d have to take three groups as the first (and correct) choice. As none of the dendrograms seemed to strongly advocate for any number of groupings, except the “complete” and “ward” method dendrograms, which advocated for three groups.