**DBScan**

This seems to be another exploratory cluster analysis method. The K-means clustering algorithm can run into issues if the clusters are nested, like these are (it would probably split off the lower part of the green group and add it to the blue group). And also if we have outliers, like these gray dots, that aren’t really supposed to be part of a group (K-means clustering puts *everything* in a group).

Bubble chart

Description automatically generated

So I guess if you don’t know exactly how many groups you’re supposed to have, and whether or not all the points are supposed to be in a group, and are therefore interested in something more like exploratory cluster analysis, then you might want to use DBScan. It identifies a cluster as a contiguous region of highly dense points.

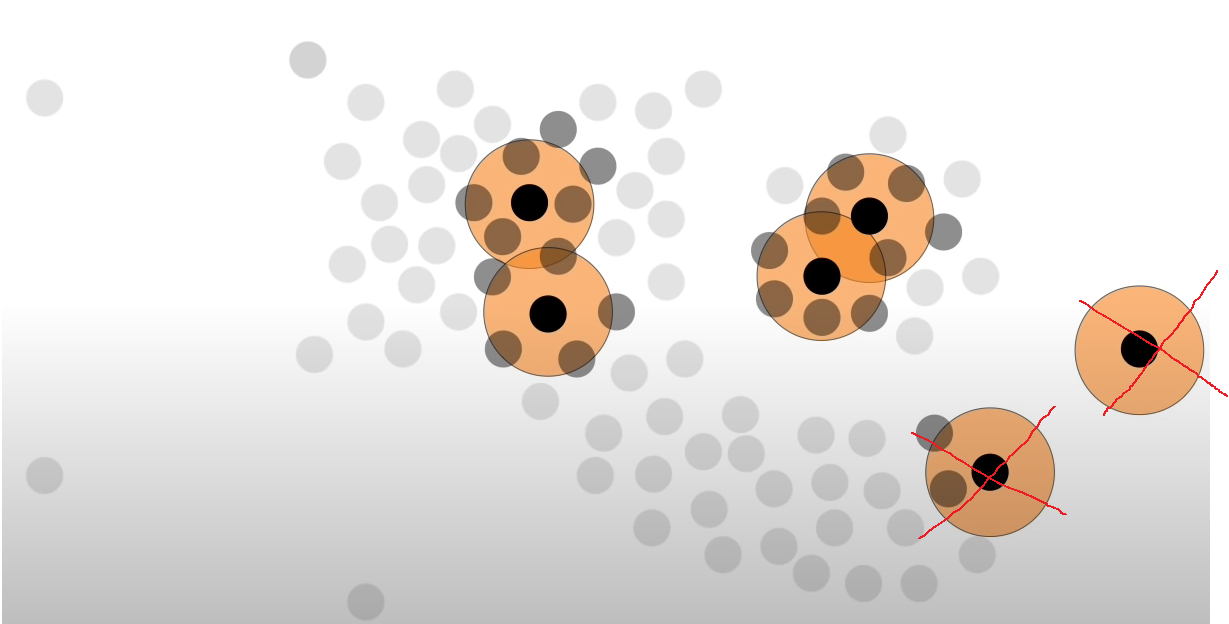
**Step 1. Set Density Parameter(s)**

It starts off with a two user-defined parameters: a circle (hypersphere) *radius* and the *number* *of points* that must be within that radius for the circle to be considered part of the core of the cluster. In the illustration below, we’ll have set the number of points as 4. A rule of thumb is that we need:



**Step 2. Identify Cluster Core Points**

Then the algorithm goes to each point, draws said circle around, and counts the number of points which it encloses. If it is at least the required number, then that point is designated a core cluster point. These points, so-identified, are not necessarily part of the *same* cluster. They’re just ascertained to be core points of *some* cluster. So for instance, those four black dots would be identified as core cluster points. But not the two x-ed out.



Ultimately, we would identify these core points,

Chart, bubble chart

Description automatically generated

**Step 3. Identify all Separate Cluster Cores**

But so far, we have no idea whether one core point is in a cluster with another core point. So now we want identify the different clusters. So we pick a random core point and call it part of Cluster 1. Then we draw the circle around the point, and whichever core points are enclosed by the circle, we add to Cluster 1. Then for each of those core points just added, we draw circles around *them*, and whichever core points are further enclosed, we add to Cluster 1. When our circles no longer enclose any more core points, then we’ve mapped out all the core points of Cluster 1. Then we can pick another random core point, designate it Cluster 2, and repeat the process, adding more and more points to Cluster 2. Ultimately, we’ll have something like this:

Chart, bubble chart

Description automatically generated

**Step 4. Identify Surfaces of Clusters**

Now that the core points of all clusters have been identified, we will want identify the surface points of these clusters. So go back to Cluster 1, run through all the core points, draw the circle, and now include within Cluster 1 *all* points within the circle. Of course the circle will enclose a lot of core points already included, but it will also include these surface points too. And so these will now have been added. Then we do the same thing for the other cluster. And we end up with:

Chart, bubble chart

Description automatically generated

Not sure why the second cluster is now blue again?

**Step 5. Identify Outliers**

All remaining points are considered outliers. Now let’s try out the DBScan class in sklearn. I’ll use the same data set as I’ve been using with the other two clustering algorithms.

**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 sklearn. 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: eps**

eps controls the radius within which core points are to be found. So I’ll play around with a couple values, and keep min\_samples = 5 (the minimum number of points which must be within that radius to establish a core).

A graph of a number of dots

Description automatically generated with medium confidenceA graph of a number of dots

Description automatically generated with medium confidence

A graph of a number of dots

Description automatically generated with medium confidence

Well, basically all points are classed as outliers.

**Hyperparameter: min\_samples**

Okay so let’s try min\_samples. And I get the same kind of thing – all outliers. Have to think about what’s going on here. I would’ve thought it would identify a few clusters, and a bunch of outliers.