**DBSCAN Worksheet**

Consider the dataset pictured below, on the left. Ignore xy-coordinates; for now, the only thing that matters is the relative spacing of the samples. It should be fairly obvious that there are two clusters of data, both shaped like crescent moons. The K-means algorithm, as good as it is for clustering blob shaped data, is unable to handle the crescent moons, as shown on the right.

|  |  |
| --- | --- |
| **Raw Data** | **K-means Clusters** |

The DBSCAN algorithm is good for clustering data that is organized into dense, complex shapes. It searches through every sample in a dataset, one at a time, counting *neighboring* samples. DBSCAN has a parameter called *epsilon* that defines the maximum distance apart for two points to be in the same neighborhood. The second parameter, *minimum samples*, sets the number of neighbors a sample must have to be a *core point*.

Every core point is part of the same cluster as shown by the red samples. Some samples are within the same neighborhood as a core point, but they themselves do not have enough neighbors to reach the core point threshold; these are called *border points*. Border points are also part of the cluster, but they are dead-ends, as shown by the yellow samples. Any remaining points that are neither core points nor border points are considered *noise points*, as shown by the blue sample.

min\_samples = 3 (or 4)

**DBSCAN**

Epsilon = 2"

Minimum Points = 4 (including self)

How many clusters?

How many border points?

How many noise points?

**DBSCAN**

Epsilon = 3"

Minimum Points = 7 (including self)

How many clusters?

How many border points?

How many noise points?

**DBSCAN**

Epsilon = 2.5"

Minimum Points = 7 (including self)

How many clusters?

How many border points?

How many noise points?

**DBSCAN**

Epsilon = 2"

Minimum Points = 4 (including self)

How many clusters?

How many border points?

How many noise points?

**DBSCAN**

Epsilon = 3"

Minimum Points = 4 (including self)

Draw a dataset that contains two clusters, each having five core points and three border points. Add four noise points, at least one of which has two border points in its neighborhood.

Now shift to programmer mode, thinking about the overall design and organization of a class that would implement the DBSCAN algorithm. Based on your previous work, (A) identify the main functions that you would use and (B) lay out some pseudo code that shows how they work together to find the clusters. You’ll find that a queue is helpful.