3

NEIGHBORHOOD APPROACHES AND DBSCAN

OVERVIEW

In this chapter, we will see how neighborhood approaches to clustering work from start to end and implement the **Density-Based Spatial Clustering of Applications with Noise** (**DBSCAN**) algorithm from scratch by using packages. We will also identify the most suitable algorithm to solve your problem from k-means, hierarchical clustering, and DBSCAN. By the end of this chapter, we will see how the DBSCAN clustering approach will serve us best in the sphere of highly complex data.

INTRODUCTION

In previous chapters, we evaluated a number of different approaches to data clustering, including k-means and hierarchical clustering. While k-means is the simplest form of clustering, it is still extremely powerful in the right scenarios. In situations where k-means can't capture the complexity of the dataset, hierarchical clustering proves to be a strong alternative.

One of the key challenges in unsupervised learning is that you will be presented with a collection of feature data but no complementary labels telling you what a target state will be. While you may not get a discrete view of what the target labels are, you can get some semblance of structure out of the data by clustering similar groups together and seeing what is similar within groups. The first approach we covered to achieve this goal of clustering similar data points is k-means. K-means clustering works best for simple data challenges where speed is paramount. Simply looking at the closest data point (cluster centroid) does not require a lot of computational overhead; however, there is also a greater challenge posed when it comes to higher-dimensional datasets. K-means clustering is also not ideal if you are unaware of the potential number of clusters you are looking for. An example we worked with in *Chapter 2*, *Hierarchical Clustering*, entailed looking at chemical profiles to determine which wines belonged together in a disorganized shipment. This exercise only worked well because we knew that three wine types were ordered; however, k-means would have been less successful if you had no idea regarding what the original order constituted.

The second clustering approach we explored was hierarchical clustering. This method can work in two ways – either agglomerative or divisive. Agglomerative clustering works with a bottom-up approach, treating each data point as its own cluster and recursively grouping them together with linkage criteria. Divisive clustering works in the opposite way by treating all data points as one large class and recursively breaking them down into smaller clusters. This approach has the benefit of fully understanding the entire data distribution, as it calculates splitting potential; however, it is typically not implemented in practice due to its greater complexity. Hierarchical clustering is a strong contender for your clustering needs when it comes to not knowing anything about the data. Using a dendrogram, you can visualize all the splits in your data and consider what number of clusters makes sense after the fact. This can be really helpful in your specific use case; however, it also comes at a higher computational cost than is associated with k-means.

In this chapter, we will cover a clustering approach that will serve us best in the sphere of highly complex data: **Density-Based Spatial Clustering of Applications with Noise** (**DBSCAN**). Canonically, this method has always been seen as a high performer in datasets that have a lot of densely interspersed data. Let's walk through why it does so well in these use cases.

CLUSTERS AS NEIGHBORHOODS

Until now, we have explored the concept of likeness being described as a function of Euclidean distance – data points that are closer to any one point can be seen as similar, while those that are further away in Euclidean space can be seen as dissimilar. This notion is seen once again in the DBSCAN algorithm. As alluded to by the lengthy name, the DBSCAN approach expands upon basic distance metric evaluation by also incorporating the notion of density. If there are clumps of data points that all exist in the same area as one another, they can be seen as members of the same cluster:

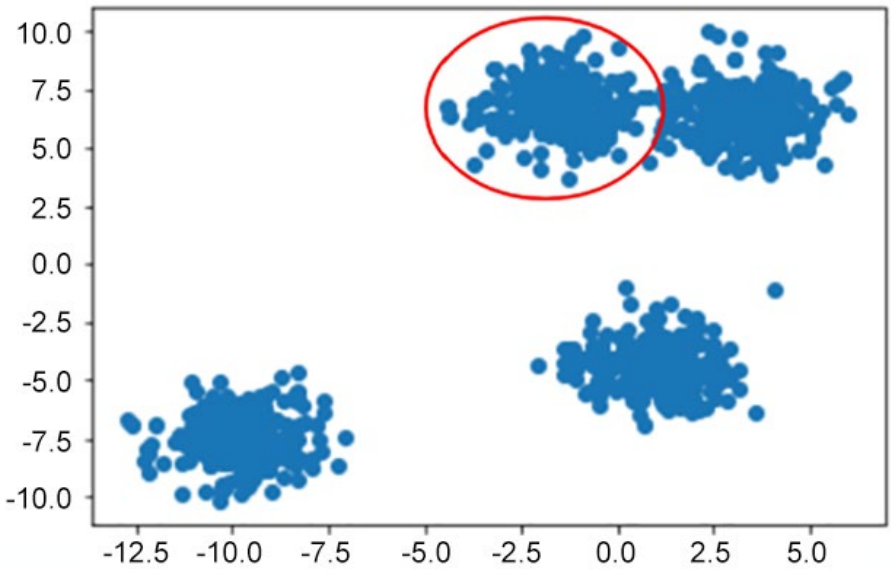

Figure 3.1: Neighbors have a direct connection to clusters

In the preceding figure, we can see four neighborhoods. The density-based approach has a number of benefits when compared to the past approaches we've covered that focus exclusively on distance. If you were just focusing on distance as a clustering threshold, then you may find your clustering makes little sense if faced with a sparse feature space with outliers. Both k-means and hierarchical clustering will automatically group together all data points in the space until no points are left.

While hierarchical clustering does provide a path around this issue somewhat, since you can dictate where clusters are formed using a dendrogram post-clustering run, k-means is the most susceptible to failure as it is the simplest approach to clustering. These pitfalls are less evident when we begin evaluating neighborhood approaches to clustering. In the following dendrogram, you can see an example of the pitfall where all data points are grouped together. Clearly, as you travel down the dendrogram, there is a lot of potential variation that gets grouped together since every point needs to be a member of a cluster. This is less of an issue with neighborhood-based clustering:

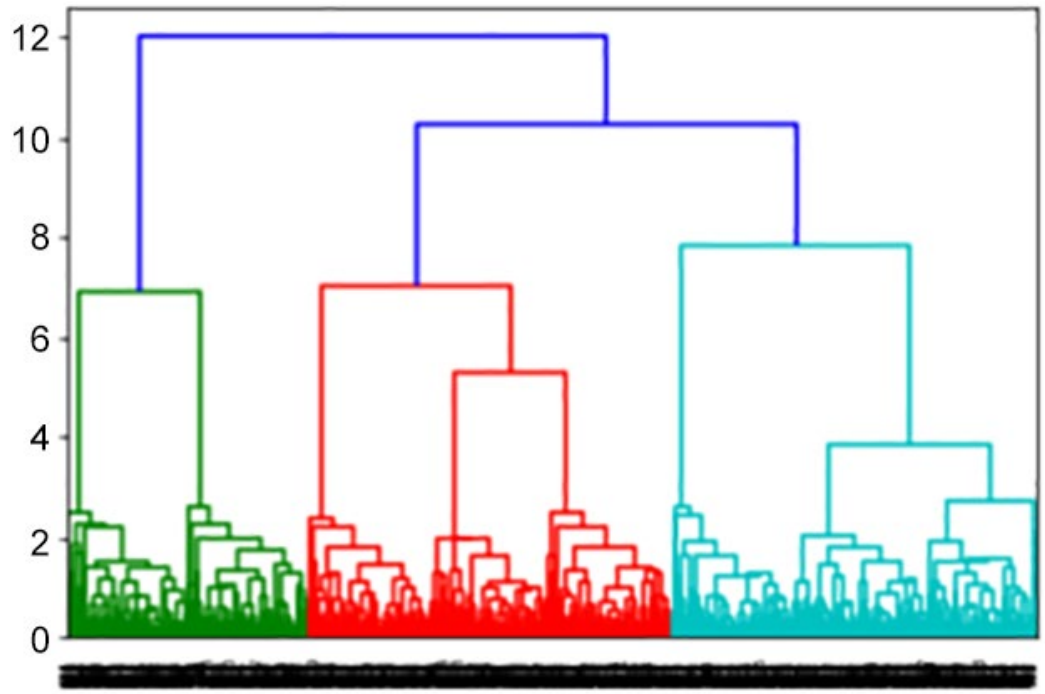

Figure 3.2: Example dendrogram

By incorporating the notion of neighbor density in DBSCAN, we can leave outliers out of clusters if we choose to, based on the hyperparameters we choose at runtime. Only the data points that have close neighbors will be seen as members within the same cluster, and those that are farther away can be left as unclustered outliers.

INTRODUCTION TO DBSCAN

In DBSCAN, density is evaluated as a combination of neighborhood radius and minimum points found in a neighborhood deemed a cluster. This concept can be driven home if we reconsider the scenario where you are tasked with organizing an unlabeled shipment of wine for your store. In the previous example, it was made clear that we can find similar wines based on their features, such as chemical traits. Knowing this information, we can more easily group together similar wines and efficiently have our products organized for sale in no time. In the real world, however, the products that you order to stock your store will reflect real-world purchase patterns. To promote variety in your inventory, but still have sufficient stock of the most popular wines, there is a highly uneven distribution of product types that you have available. Most people love the classic wines, such as white and red; however, you may still carry more exotic wines for your customers who love expensive varieties. This makes clustering more difficult, since there are uneven class distributions (you don't order 10 bottles of every wine available, for example).

DBSCAN differs from k-means and hierarchical clustering because you can build this intuition into how we evaluate the clusters of customers we are interested in forming. It can cut through the noise in an easier fashion and only point out customers who have the highest potential for remarketing in a campaign.

By clustering through the concept of a neighborhood, we can separate out the one-off customers who can be seen as random noise, relative to the more valuable customers who come back to our store time and time again. This approach calls into question how we establish the best numbers when it comes to neighborhood radius and minimum points per neighborhood.

As a high-level heuristic, we want our neighborhood radius to be small, but not too small. At one end of the extreme, you can have the neighborhood radius quite high – this can max out at treating all points in the feature space as one massive cluster. At the opposite end of the extreme, you can have a very small neighborhood radius. Overly small neighborhood radii can result in no points being clustered together and having a large collection of single-member clusters.

Similar logic applies when it comes to the minimum number of points that can make up a cluster. Minimum points can be seen as a secondary threshold that tunes the neighborhood radius a bit, depending on what data you have available in your space. If all of the data in your feature space is extremely sparse, minimum points become extremely valuable, in tandem with the neighborhood radius, to make sure you don't just have a large number of uncorrelated data points. When you have very dense data, the minimum points threshold becomes less of a driving factor than neighborhood radius.

As you can see from these two hyperparameter rules, the best options are, as usual, dependent on what your dataset looks like. Oftentimes, you will want to find the perfect "goldilocks" zone of not being too small in your hyperparameters, but also not too large.

DBSCAN IN DETAIL

To see how DBSCAN works, we can trace the path of a simple toy program as it merges together to form a variety of clusters and noise-labeled data points:

1. Out of *n* unvisited sample data points, we'll first move through each point in a loop and mark each one as visited.
2. From each point, we'll look at the distance to every other point in the dataset.
3. All points that fall within the neighborhood radius hyperparameter should be considered as neighbors.
4. The number of neighbors should be at least as many as the minimum points required.
5. If the minimum point threshold is reached, the points should be grouped together as a cluster, or else marked as noise.
6. This process should be repeated until all data points are categorized in clusters or as noise.

DBSCAN is fairly straightforward in some senses – while there are the new concepts of density through neighborhood radius and minimum points, at its core, it is still just evaluating using a distance metric.

WALKTHROUGH OF THE DBSCAN ALGORITHM

The following steps will walk you through this path in slightly more detail:

1. Given six sample data points, view each point as its own cluster [(1,3)], [(-8,6)], [(-6,4)] , [(4,-2)],] (2,5)], [(-2,0)]:

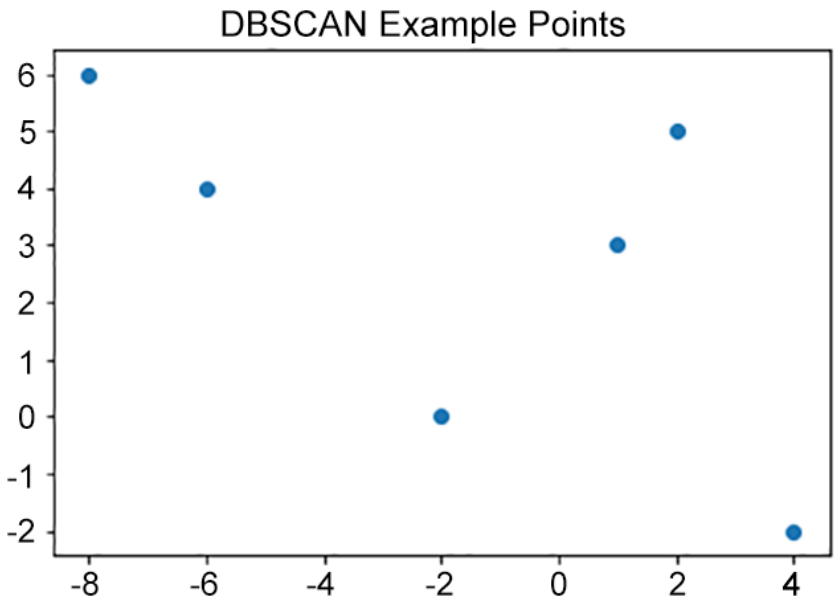

Figure 3.3: Plot of sample data points

2. Calculate the pairwise Euclidean distance between each of the points:

POINTS	(1,3)	(-8,6)	(-6,4)	(4,-2)	(2,5)	(-2,0)
(1,3)	[0.	9.48683298	7.07106781	5.83095189	2.23606798	4.24264069]
(-8,6)	[9.48683298	0.	2.82842712	14.4222051	10.04987562	8.48528137]
(-6,4)	[7.07106781	2.82842712	0.	11.66190379	8.06225775	5.65685425]
(4,-2)	[5.83095189	14.4222051	11.66190379	0.	7.28010989	6.32455532]
(2,5)	[2.23606798	10.04987562	8.06225775	7.28010989	0.	6.40312424]
(-2,0)	[4.24264069	8.48528137	5.65685425	6.32455532	6.40312424	0.]

Figure 3.4: Point distances

3. From each point, expand a neighborhood size outward and form clusters. For the purpose of this example, imagine you pass through a neighborhood radius of five. This means that any two points will be neighbors if the distance between them is less than five units. For example, point (1,3) has points (2,5) and (-2,0) as neighbors.

   Depending on the number of points in the neighborhood of a given point, the point can be classified into the following three categories:

   Core Point: If the point under observation has data points greater than the minimum number of points in its neighborhood that make up a cluster, then that point is called a core point of the cluster. All core points within the neighborhood of other core points are part of the same cluster. However, all the core points that are not in same neighborhood are part of another cluster.

   Boundary Point: If the point under observation does not have sufficient neighbors (data points) of its own, but it has at least one core point (in its neighborhood), then that point represents the boundary point of the cluster. Boundary points belong to the same cluster of their nearest core point.

   Noise Point: A data point is treated as a noise point if it does not have the required minimum number of data points in its neighborhood and is not associated with a core point. This point is treated as pure noise and is excluded from clustering.

4. Points that have neighbors are then evaluated to see whether they pass the minimum points threshold. In this example, if we had passed through a minimum points threshold of two, then points (1,3), (2,5), and (-2,0) could formally be grouped together as a cluster. If we had a minimum points threshold of four, then these three data points would be considered superfluous noise.

5. Points that have fewer neighbors than the minimum number of neighboring points required and whose neighborhood does not contain a core point are marked as noise and remain unclustered. Thus, points (-6,4), (4,-2), and (-8,6) fall under this category. However, points such as (2,5) and (2,0), though don't satisfy the criteria of the minimum number of points in neighborhood, do contain a core point as their neighbor, and are therefore marked as boundary points.

6. The following table summarizes the neighbors of a particular point and classifies them as core, boundary, and noise data points (mentioned in the preceding step) for a neighborhood radius of 5 and a minimum-neighbor criterion of 2.

Point of observation	**Neighbor**	**Has min number of neighbors**	**Does contain a core point as a neighbor**	**Classification**
(1,3)	(2,5) , (-2,0)	Yes	-	Core Point
(-8,6)	(-6,4)	No	No	Noise
(-6,4)	(-8,6)	No	No	Noise
(4,-2)		No	No	Noise
(2,5)	(1,3)	No	Yes	Boundary Point
(-2,0)	(1,3)	No	Yes	Boundary Point

Figure 3.5: Table showing details of neighbors for given points

7. Repeat this process on any remaining unvisited data points.

At the end of this process, you will have sorted your entire dataset into either clusters or unrelated noise. DBSCAN performance is highly dependent on the threshold hyperparameters you choose. This means that you may have to run DBSCAN a couple of times with different hyperparameter options to get an understanding of how they influence overall performance.

Note that DBSCAN does not require the centroids that we saw in both k-means and centroid-focused implementation of hierarchical clustering. This feature allows DBSCAN to work better for complex datasets, since most data is not shaped like clean blobs. DBSCAN is also more effective against outliers and noise than k-means or hierarchical clustering.

Let's now see how the performance of DBSCAN changes with varying neighborhood radius sizes.