**DBSCAN Clustering Algorithm — How to Build Powerful Density-Based Models**

# Types of clustering algorithms

Not all clustering algorithms are created equal. Different clustering algorithms implement different ideas on how to best cluster your data. There are 4 main categories:

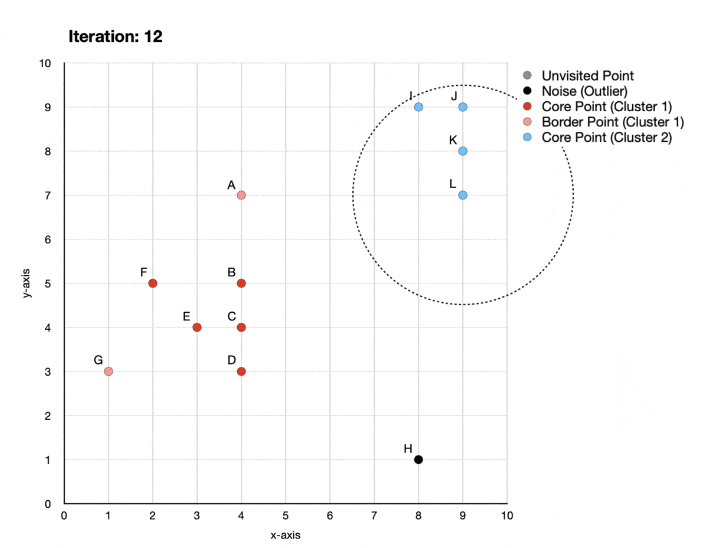
* **Centroid-based** — uses Euclidean distance to assign every point to the nearest cluster center. Example: [K-Means](https://towardsdatascience.com/k-means-clustering-a-comprehensive-guide-to-its-successful-use-in-python-c3893957667d)
* **Connectivity-based**— assumes that nearby objects (data points) are more related than far away objects. Example: [Hierarchical Agglomerative Clustering (HAC)](https://towardsdatascience.com/hac-hierarchical-agglomerative-clustering-is-it-better-than-k-means-4ff6f459e390).
* **Density-based** — defines clusters as dense regions of space separated by low-density regions. Example: Density-Based Spatial Clustering of Applications with Noise (**DBSCAN**).
* **Distribution-based**— assumes the existence of a specified number of distributions within the data. Each distribution with its own mean (μ) and variance (σ²) / covariance (Cov). Example: [Gaussian Mixture Models (GMM)](https://towardsdatascience.com/gmm-gaussian-mixture-models-how-to-successfully-use-it-to-cluster-your-data-891dc8ac058f).

# How does the DBSCAN algorithm work?

## Defining parameters

As mentioned above, density-based algorithms work by identifying dense regions in space (i.e., populated with many data points) separated by less dense regions. To enable the algorithm to find these dense regions, we first need to establish what we consider to be sufficiently dense. We do this by specifying two hyperparameters:

* **Epsilon** (ϵ, sklearn: eps) — the radius of the area around the point defining the maximum distance between such point and any other points for one to be considered in the neighborhood of the other.
* **Min points** (MinPts, sklearn: min\_samples) — the minimum number of points present in the neighborhood required to form a cluster.



* **Iteration 0** — none of the points have been visited yet. Next, the algorithm will randomly pick a starting point taking us to iteration 1.
* **Iteration 1** — point A has only one other neighbor. Since 2 points (A+1 neighbor) is less than 4 (minimum required to form a cluster, as defined above), A is labeled as noise.
* **Iteration 2** — point B has 5 other points in its neighborhood. Since 6 points (B + 5 neighbors) is more than 4, the start of the first cluster is identified, with **B** becoming a **core point**. At the same time, previously considered to be noise, point **A is relabeled as a border point** since it sits within the neighborhood of core point B.
* **Iterations 3 to 6** — points C, D, E, F are labeled as core points and assigned to cluster 1.
* **Iteration 7**— point G is labeled as a border point because it is within the neighborhood of core points E and F but does not reach the minimum number of 4 points to be labeled as a core point.
* **Iteration 8** — point H is labeled as a noise (outlier) because there are no other points within its neighborhood.
* **Iterations 9 to 12**— a new cluster is identified and labeled, containing 4 core points (I, J, K, L).

## Pros and cons of DBSCAN compared to other clustering methods

**Pros:**

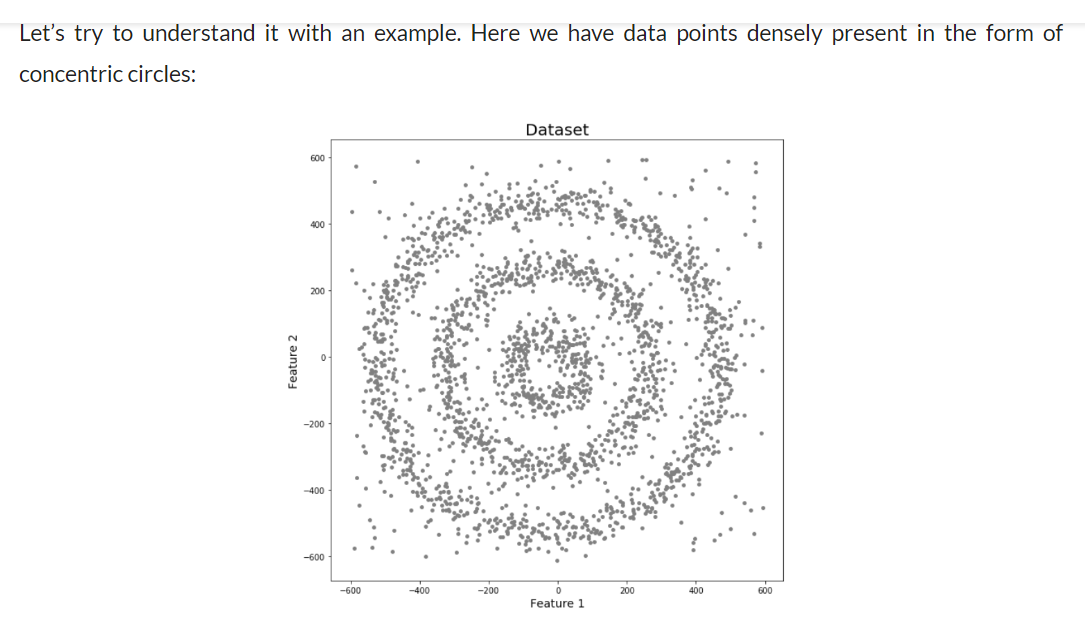
First, DBSCAN doesn’t require users to specify the number of clusters.

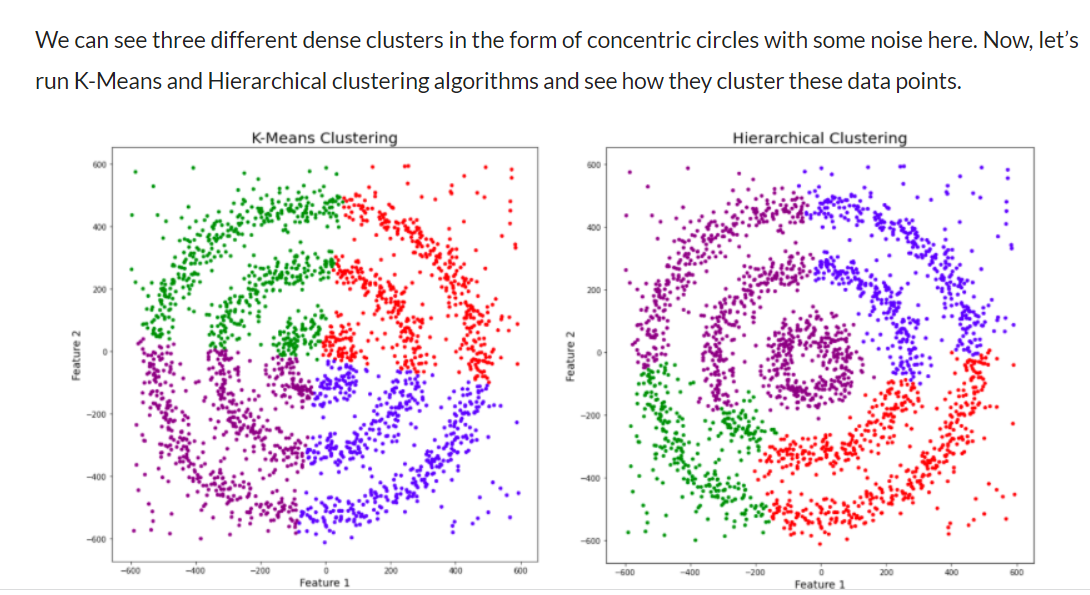
Second, DBSCAN is NOT sensitive to outliers.

Third, the clusters formed by DBSCAN can be any shape, which makes it robust to different types of data.

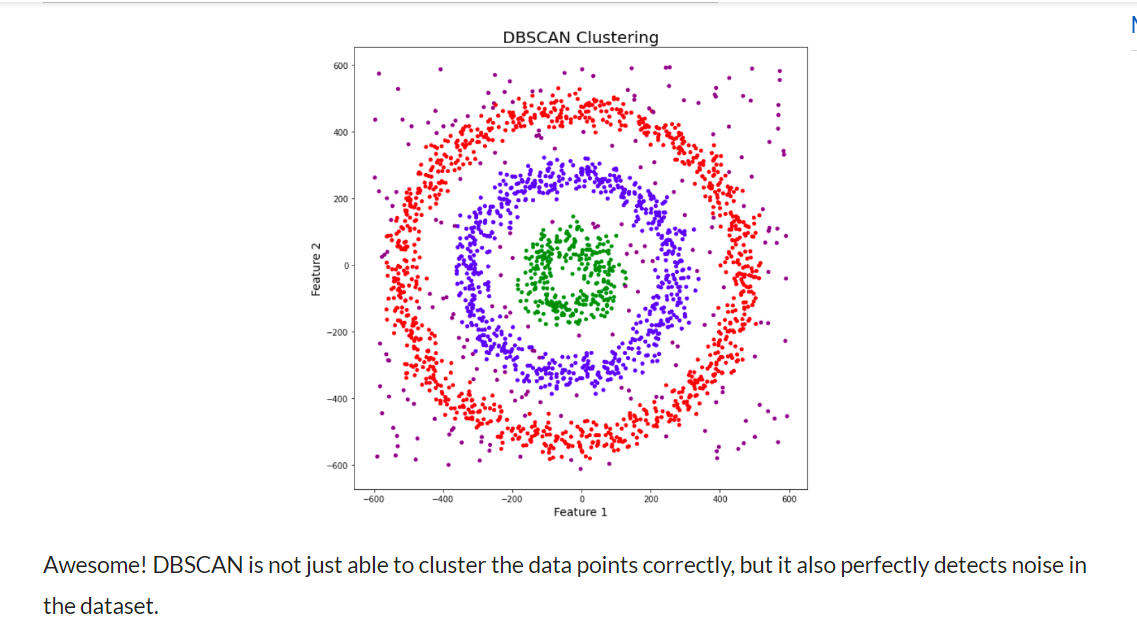
* **Cons**: It does not work well when there is varying density across the data since Epsilon and MinPts are fixed.

K-Means and Hierarchical Clustering both fail in creating clusters of arbitrary shapes. They are not able to form clusters based on varying densities. That’s why we need DBSCAN clustering.

****

****

# You might be wondering why there are four colors in the graph? As I said earlier, this data contains noise too, therefore, I have taken noise as a different cluster which is represented by the purple color. Sadly, both of them failed to cluster the data points. Also, they were not able to properly detect the noise present in the dataset. Now, let’s take a look at the results from DBSCAN clustering.

****

Purple points are identified as outliers.

# It groups ‘densely grouped’ data points into a single cluster. It can identify clusters in large spatial datasets by looking at the local density of the data points. The most exciting feature of DBSCAN clustering is that it is robust to outliers. It also does not require the number of clusters to be told beforehand, unlike K-Means, where we have to specify the number of centroids.

# In higher dimensions the circle becomes hypersphere, epsilon becomes the radius of that hypersphere, and minPoints is the minimum number of data points required inside that hypersphere.

# DBSCAN creates a circle of epsilon radius around every data point and classifies them into Core point, Border point, and Noise. A data point is a Core point if the circle around it contains at least ‘minPoints’ number of points. If the number of points is less than minPoints, then it is classified as Border Point, and if there are no other data points around any data point within epsilon radius, then it treated as Noise.

# 

All the data points with at least 3 points in the circle including itself are considered as Core points represented by red color. All the data points with less than 3 but greater than 1 point in the circle including itself are considered as Border points. They are represented by yellow color. Finally, data points with no point other than itself present inside the circle are considered as Noise represented by the purple color.

For locating data points in space, DBSCAN uses [Euclidean distance](https://www.analyticsvidhya.com/blog/2020/02/4-types-of-distance-metrics-in-machine-learning/?utm_source=blog&utm_medium=DBSCAN).

## **Parameter Selection in DBSCAN Clustering.**

# DBSCAN is very sensitive to the values of epsilon and minPoints. Therefore, it is very important to understand how to select the values of epsilon and minPoints. A slight variation in these values can significantly change the results produced by the DBSCAN algorithm.

## **Minimum Samples (“MinPts”)**

There is no automatic way to determine the MinPts value for DBSCAN. Ultimately, the MinPts value should be set using domain knowledge and familiarity with the data set. From some research I’ve done, here are a few rules of thumb for selecting the MinPts value:

* The larger the data set, the larger the value of MinPts should be
* If the data set is noisier, choose a larger value of MinPts
* Generally, MinPts should be greater than or equal to the dimensionality of the data set
* For 2-dimensional data, use DBSCAN’s default value of MinPts = 4.
* If your data has more than 2 dimensions, choose MinPts = 2\*dim, where dim= the dimensions of your data set.
* **The value of minPoints should be at least one greater than the number of dimensions of the dataset:**
* **minPoints>=Dimensions+1.**

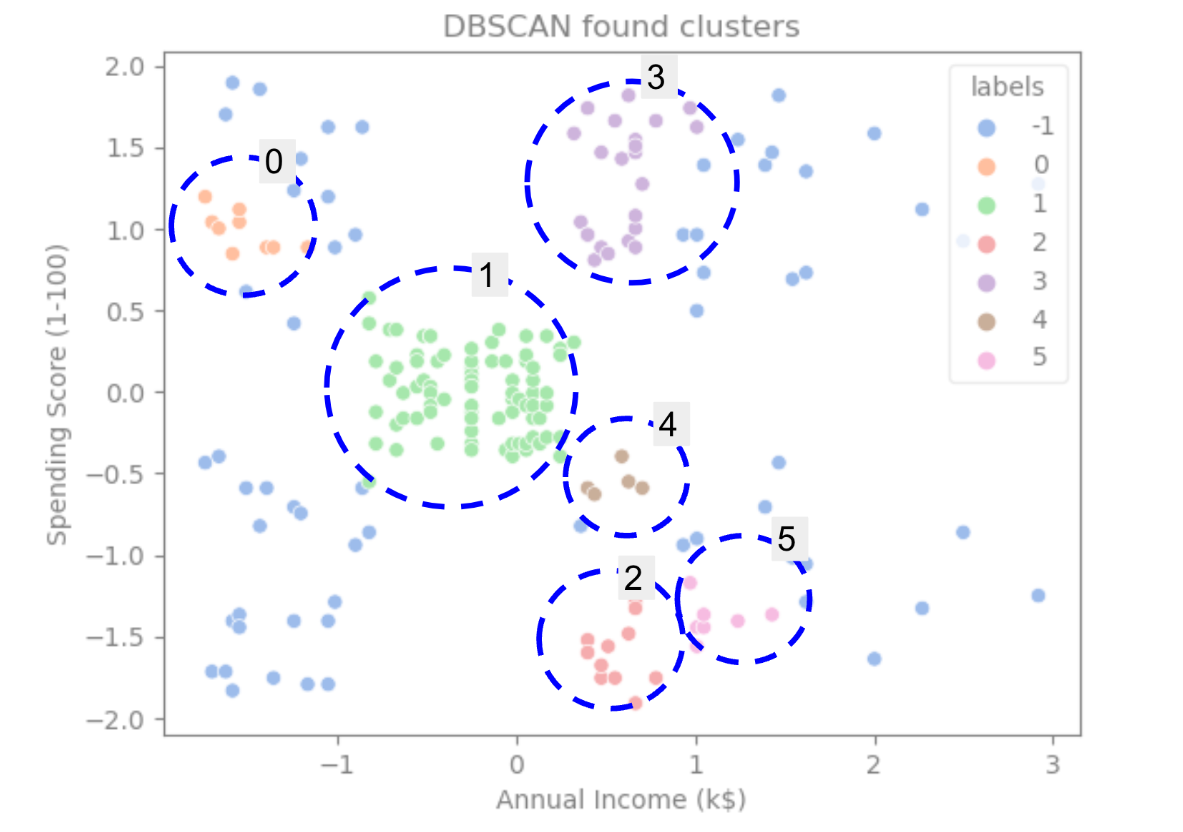
It does not make sense to take minPoints as 1 because it will result in each point being a separate cluster. Therefore, it must be at least 3. Generally, it is twice the dimensions. But domain knowledge also decides its value.

**Epsilon:**

The value of epsilon can be decided from the K-distance graph. The point of maximum curvature (elbow) in this graph tells us about the value of epsilon. If the value of epsilon chosen is too small then a higher number of clusters will be created, and more data points will be taken as noise. Whereas, if chosen too big then various small clusters will merge into a big cluster**, and we will lose details.**

It is built on the premise that we have a high-density of points.

* **eps(Epsilon)**: Two points are considered neighbors if the distance between the two points is below the threshold epsilon.
* **min\_samples**: The minimum number of neighbors a given point should have in order to be classified as a core point. **It’s important to note that the point itself is included in the minimum number of samples.**
* **metric**: The metric to use when calculating distance between instances in a feature array (i.e. euclidean distance).



If we highlight the clusters, notice how DBSCAN gets cluster 1 completely, which is the cluster with less space between points. Then it gets the parts of clusters 0 and 3 where the points are closely together, considering more spaced points as noise. It also considers the points in the lower left half as noise and splits the points in the lower right into 3 clusters, once again capturing clusters 4, 2, and 5 where the points are closer together. We can confirm with these example that:

**It does not work well when there is varying density across the data since Epsilon and MinPts are fixed.**

**We can start to come to a conclusion that DBSCAN was great for capturing the dense areas of the clusters but not so much for identifying the bigger scheme of the data**, this case should be 5 clusters and we have more. It would be interesting to test more clustering algorithms with our data.