**DBSCAN Clustering Algorithm**

DBSCAN (**Density-based spatial clustering of applications with noise**) is a density-based clustering algorithm that divides a dataset into subgroups of high density regions. DBSCAN groups together points that are close to each other based on a distance measurement (usually Euclidean distance) and a minimum number of points. It also marks as outliers the points that are in low-density regions.

**Parameters**:

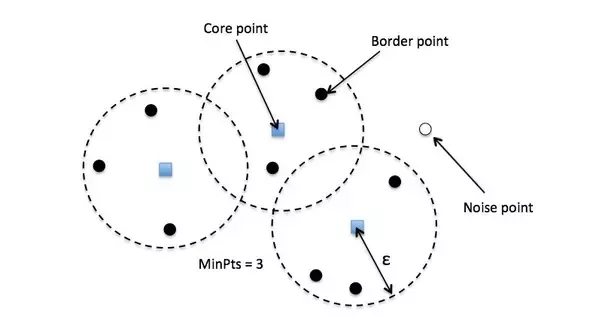
There are two parameters required for DBSCAN: epsilon (**ε**) and minimum amount of points required to form a cluster (**minPts**).

**eps**: the minimum distance between two points. It means that if the distance between two points is lower or equal to this value (eps), these points are considered neighbours.

**minPoints**: the minimum number of points to form a dense region. For example, if we set the minPoints parameter as 5, then we need at least 5 points to form a dense region.

Using **ε** and**minPts**, we can classify each data point as:

* **Core point** – a point that has at least a minimum number of other points (**minPts)** within its **ε** radius.
* **Border point** – a point is within the **ε** radius of a *core point* BUT has less than the minimum number of other points (**minPts)** within its own **ε** radius
* **Noise point** – a point that is neither a *core point* or a *border point*



**Python** implementation of 'Density Based Spatial Clustering of Applications with Noise:

**Usage:**

from sklearn.cluster import DBSCAN

db = DBSCAN(m,eps,min\_points)

Inputs:

m – A matrix whose columns are feature vectors

eps – Maximum distance two points can have to be regionally related

min\_points – The minimum number of points to make a cluster

As the input m,

NumPy arrays are the preferred data format ; lists of lists are accepted as a convenience but will be converted internally.

Eg.

X = np.array([[37.9358, -122.3478],[33.8312, -117.6053]])

db = DBSCAN(eps= 0.3,min\_samples = 10).fit(X)

#### Parameter estimation:

**eps**: if the eps value chosen is too small, a large part of the data will not be clustered. It will be considered outliers because don’t satisfy the number of points to create a dense region.

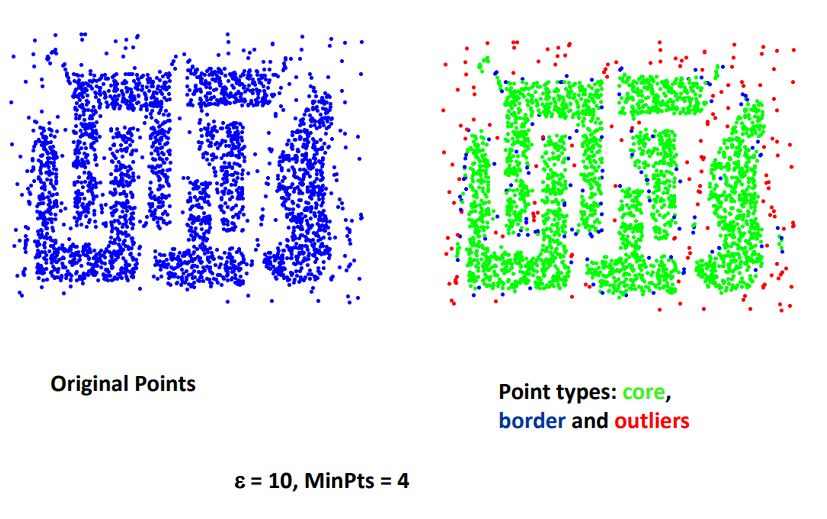
On the other hand, if the value that was chosen is too high, clusters will merge and the majority of objects will be in the same cluster.

The eps should be chosen based on the distance of the dataset (we can use a k-distance graph to find it), but in general small eps values are preferable.

**minPoints**: Minimum minPoints can be derived from a number of dimensions (D) in the data set, as minPoints ≥ D + 1.

Larger values are usually better for data sets with noise and will form more significant clusters.

The minimum value for the minPoints must be 3, but the larger the data set, the larger the minPoints value that should be chosen.



Working:

First, we choose two parameters, a positive number epsilon and a natural number minPoints. We then begin by picking an arbitrary point in our dataset. If there are more than minPoints points within a distance of epsilon from that point, (including the original point itself), we consider all of them to be part of a "cluster". We then expand that cluster by checking all of the new points and seeing if they too have more than minPoints points within a distance of epsilon, growing the cluster recursively if so. Eventually, we run out of points to add to the cluster. We then pick a new arbitrary point and repeat the process. Now, it's entirely possible that a point we pick has fewer than minPoints points in its epsilon ball, and is also not a part of any other cluster. If that is the case, it's considered a "noise point" not belonging to any cluster.

**Algorithm:**

for each o ∈ D do

if o is not yet classified then

if o is a core-object then

collect all objects density-reachable from o and assign them to a new cluster.

Else

assign o to NOISE

Example:

Suppose we have an e-commerce and we want to improve our sales by recommending relevant products to our customers. We don’t know exactly what our customers are looking for but based on a data set we can predict and recommend a relevant product to a specific customer. We can apply the DBSCAN to our data set (based on the e-commerce database) and find clusters based on the products that the users have bought. Using these clusters, we can find similarities between customers, for example, the customer A have bought 1 pen, 1 book and 1 scissors and the customer B have bought 1 book and 1 scissors, then we can recommend 1 pen to the customer B.

**Advantages:**

* DBSCAN does not require us to specify the number of clusters a-priori and works well at finding arbitrarily shaped clusters while being robust to outliers.
* DBSCAN requires just two parameters and is mostly insensitive to the ordering of the points in the database. (However, points sitting on the edge of two different clusters might swap cluster membership if the ordering of the points is changed, and the cluster assignment is unique only up to isomorphism.)

**Disadvantages:**

* The drawbacks of DBSCAN are that we have to choose **ε**, and clusters of varying densities won’t be assigned properly.
* DBSCAN is not entirely deterministic: border points that are reachable from more than one cluster can be part of either cluster, depending on the order the data are processed.