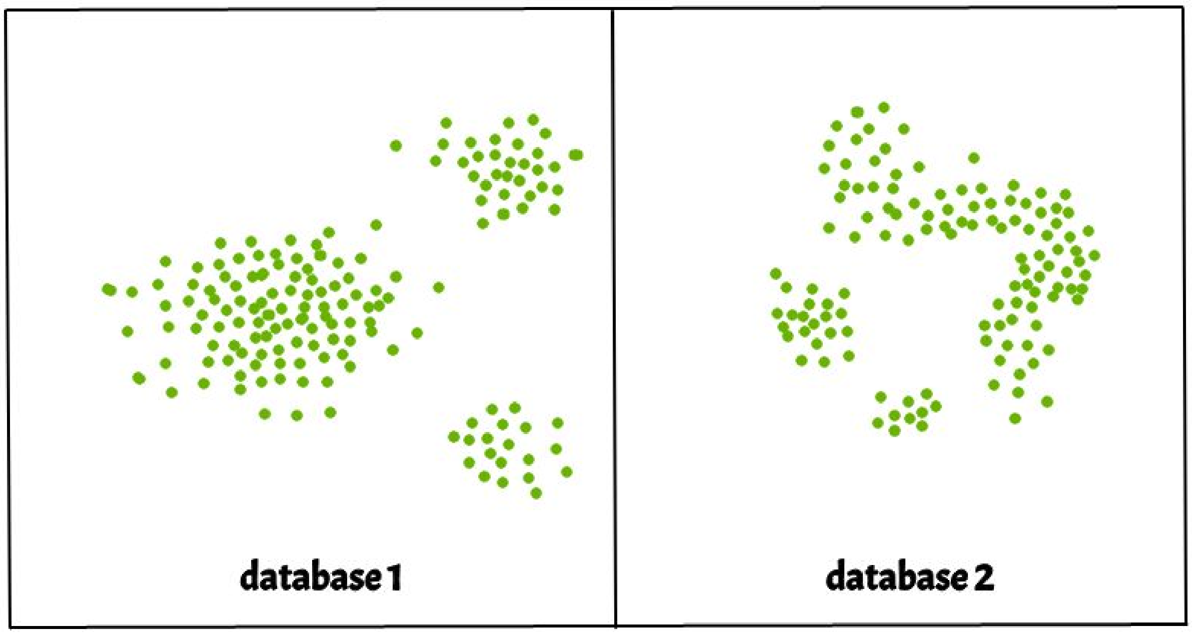
DBSCAN ALGORITHM:

- Based on a set of points (let's think in a bidimensional space as exemplified in the figure), 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.
- Stand for density based spatial clustering of applications with noise
- Clusters are dense regions in the data space, separated by regions of the lower density of points. The *DBSCAN algorithm* is based on this intuitive notion of "clusters" and "noise". The key idea is that for each point of a cluster, the neighborhood of a given radius has to contain at least a minimum number of points.



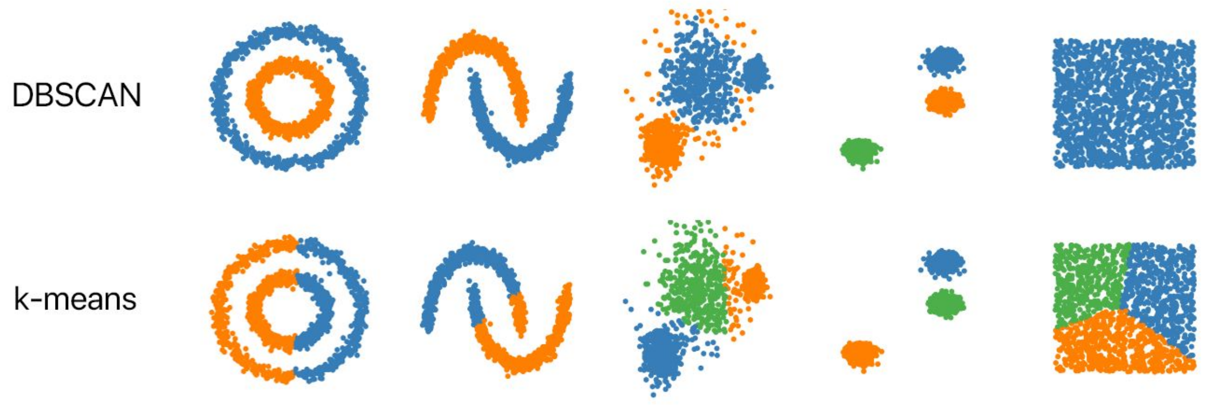
Limitations of traditional methods:

Partitioning methods (K-means, PAM clustering) and hierarchical clustering work for finding spherical-shaped clusters or convex clusters. In other words, they are suitable only for compact and well-separated clusters.

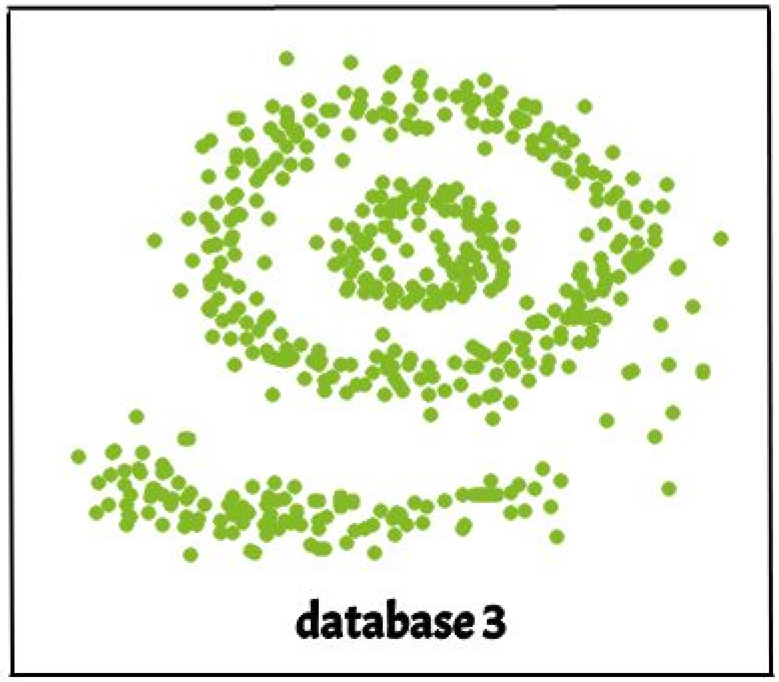
Moreover, they are also severely affected by the presence of noise and outliers in the data.

K-means Vs DBSCAN:

- K-Means clustering may cluster loosely related observations together. Every observation becomes a part of some cluster eventually, even if the observations are scattered far away in the vector space. Since clusters depend on the mean value of cluster elements, each data point plays a role in forming the clusters. A slight change in data points *might* affect the clustering outcome. This problem is greatly reduced in DBSCAN due to the way clusters are formed. This is usually not a big problem unless we come across some odd shape data.
- Another challenge with *k*-means is that you need to specify the number of clusters (“*k*”) in order to use it. Much of the time, we won’t know what a reasonable *k* value is *a priori*.
- What’s nice about DBSCAN is that you don’t have to specify the number of clusters to use it. All you need is a function to calculate the distance between values and some guidance for what amount of distance is considered “close”. DBSCAN also produces more reasonable results than *k*-means across a variety of different distributions. Below figure illustrates the fact:



Real life data may contain irregularities, like –
i) Clusters can be of arbitrary shape such as those shown in the figure below.
ii) Data may contain noise.



The figure above shows a data set containing nonconvex clusters and outliers/noises. Given such data, k-means algorithm has difficulties for identifying these clusters with arbitrary shapes.

DBSCAN algorithm requires two parameters :

minPts: The minimum number of points (a threshold) clustered together for a region to be considered dense.

eps (ε): A distance measure that will be used to locate the points in the neighborhood of any point.

Parameter Estimation:

Every data mining task has the problem of parameters. Every parameter influences the algorithm in specific ways. For DBSCAN, the parameters ε and minPts are needed.

minPts: As a rule of thumb, a minimum *minPts* can be derived from the number of dimensions *D* in the data set, as *minPts* ≥ *D* + 1. The low value *minPts* = 1 does not make sense, as then every point on its own will already be a cluster.. The, *minPts* must be chosen at least 3. However, larger values are usually better for data sets with noise and will yield more significant clusters. As a rule of thumb, *minPts* = 2·*dim* can be used, but it may be necessary to choose larger values for very large data, for noisy data or for data that contains many duplicates.

ε: The value for ε can then be chosen by using a k-distance graph, plotting the distance to the *k* = *minPts*-1 nearest neighbor ordered from the largest to the smallest value. Good values of ε are where this plot shows an “elbow”: if ε is chosen much too small, a large part of the data will not be clustered; whereas for a too high value of ε, clusters will merge and the majority of objects will be in the same cluster. In general, small values of ε are preferable, and as a rule of thumb, only a small fraction of points should be within this distance of each other.

Distance function: The choice of distance function is tightly linked to the choice of ε, and has a major impact on the outcomes. In general, it will be necessary to first identify a reasonable measure of similarity for the data set, before the parameter ε can be chosen. There is no estimation for this parameter, but the distance functions need to be chosen appropriately for the data set.

Method for determining the optimal eps value

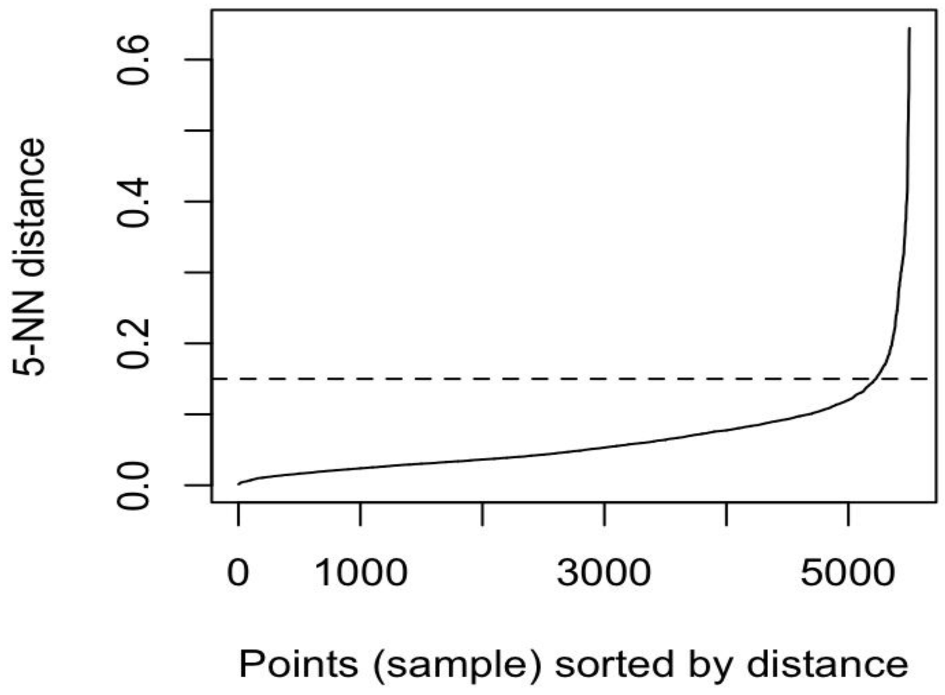
The method proposed here consists of computing the k-nearest neighbor distances in a matrix of points.

The idea is to calculate, the average of the distances of every point to its k nearest neighbors. The value of k will be specified by the user and corresponds to *MinPts*.

Next, these k-distances are plotted in an ascending order. The aim is to determine the "knee", which corresponds to the optimal *eps* parameter.

A knee corresponds to a threshold where a sharp change occurs along the k-distance curve.

The function *kNNdistplot*() [in *dbscan* package] can be used to draw the k-distance plot:



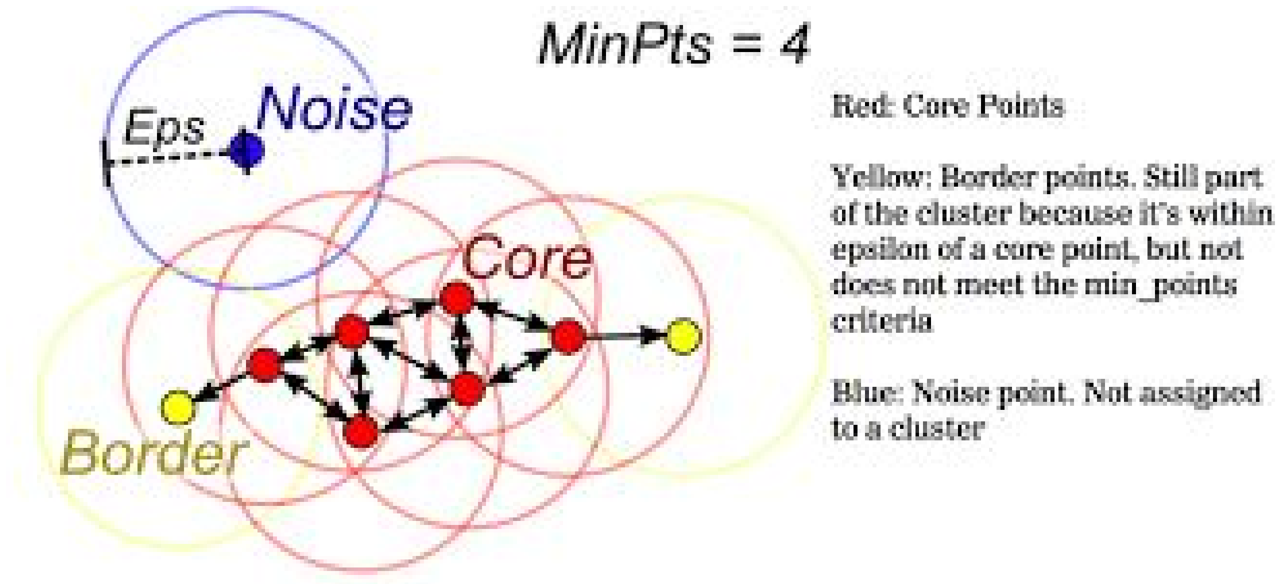
It can be seen that the optimal *eps* value is around a distance of 0.15.

In this algorithm, we have 3 types of data points.

Core Point: A point is a core point if it has more than MinPts points within eps.

Border Point: A point which has fewer than MinPts within eps but it is in the neighborhood of a core point.

Noise or outlier: A point which is not a core point or border point.


MinPts = 4
Eps
Noise
Core
Border
Red: Core Points
Yellow: Border points. Still part of the cluster because it's within epsilon of a core point, but not does not meet the min_points criteria
Blue: Noise point. Not assigned to a cluster

Algorithmic steps for DBSCAN clustering:

1. The algorithm proceeds by arbitrarily picking up a point in the dataset (until all points have been visited).
2. If there are at least 'minPoint' points within a radius of 'ε' to the point then we consider all these points to be part of the same cluster.

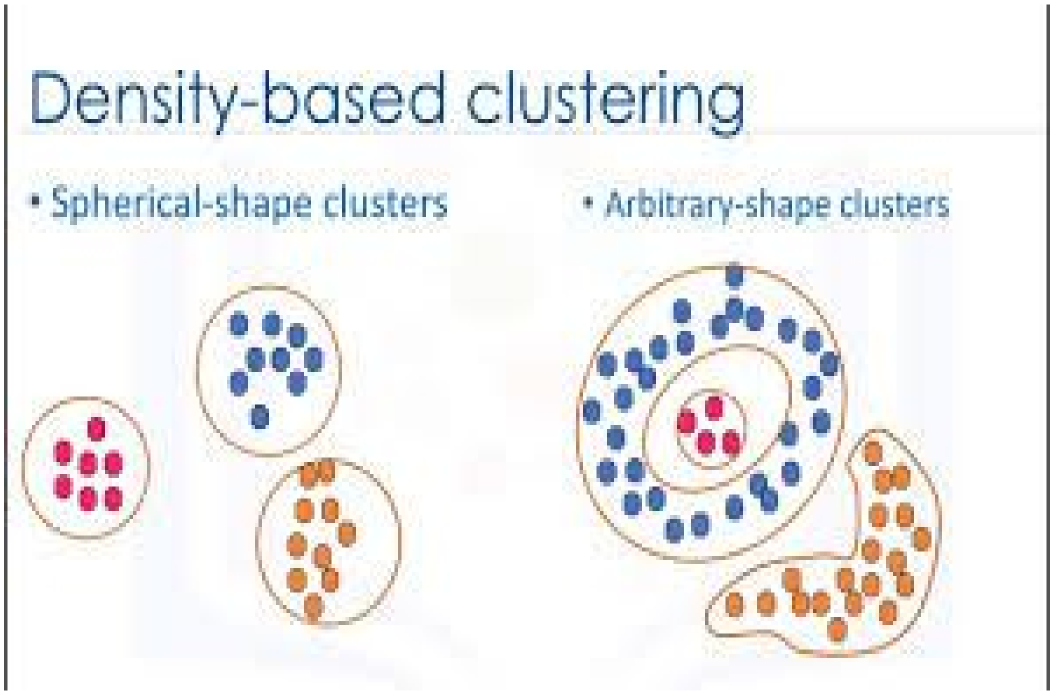
3. The clusters are then expanded by recursively repeating the neighborhood calculation for each neighboring point.

Visualisation of how the clustering is done:

https://miro.medium.com/proxy/1*tc8UF-h0nQqUfLC8-0uInQ.gif

Practical example:

A practical use of DBSCAN. 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 this 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. This is just a little example of use of DBSCAN, but it can be used in a lot of applications in several areas.



Implementation of DBSCAN:

https://colab.research.google.com/drive/1VkMWP1mvvLk6dbjwSbewDQakn7RrDtTI?usp=sharing

Implementation of DBSCAN VS KMEANS:
https://colab.research.google.com/drive/1kDdOHx1fzXZNK9bAHzvY9KBEqU6j8Tav?usp=sharing