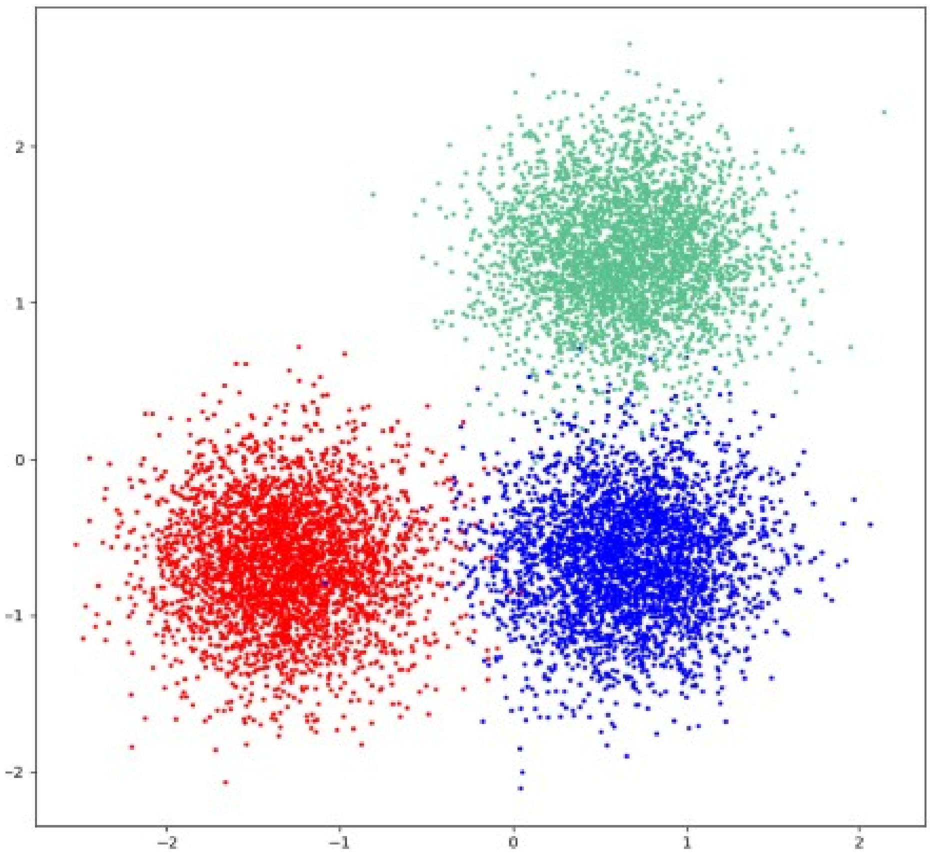
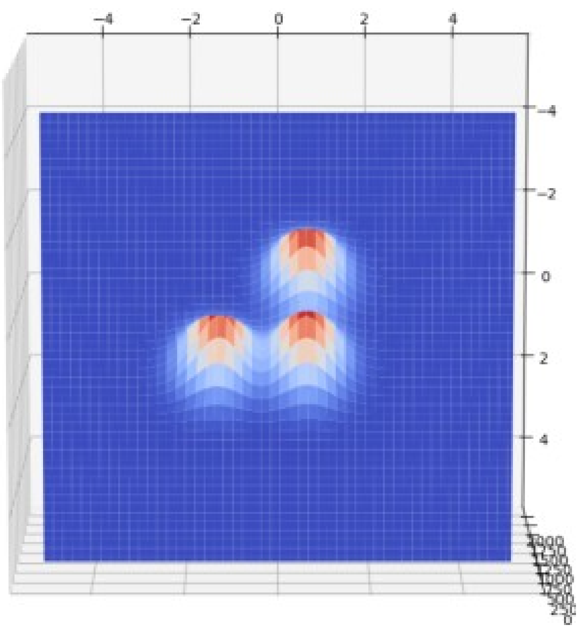
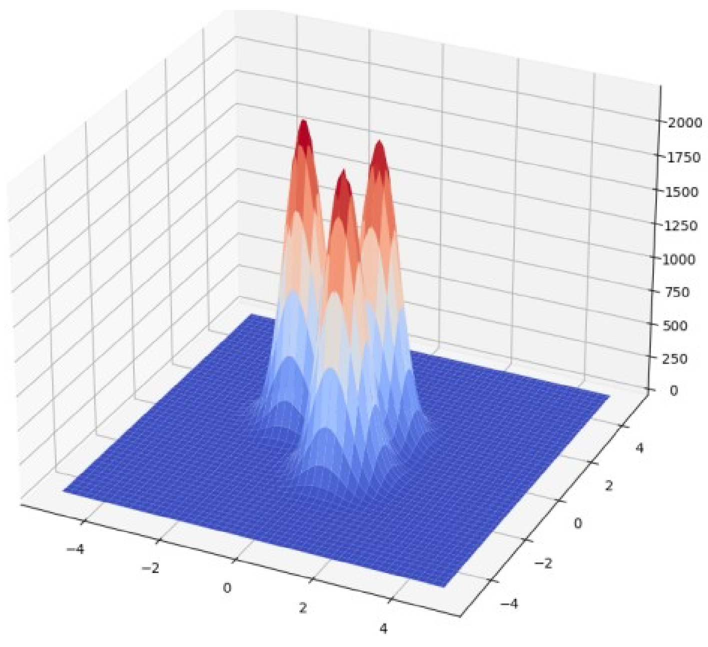
The different clustering methods

Original data



Gaussian 3-D representation of the data

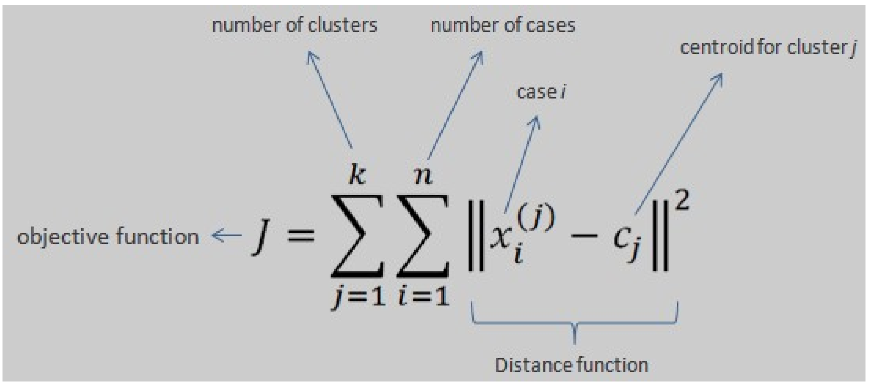




K-Means

One of the most commonly used unsupervised learning algorithm which is fast and efficient
It works on the principal of the nearest cluster grouping.
It is a Centroid/Parametric based clustering and it makes assumptions over the shape of the data and clusters it, Random points are selected as the center and the clustering starts.
It requires the number of clusters to be hard coded or be optimized by Within-Cluster-Sum-of-Squares (WCSS) .
It requires one parameter which is number of clusters.

Equation for the K-Means algorithm



Drawbacks

It picks new centers every-time on running even for the same data set thus it may cluster differently each time.
It work well over limited data as it assumes the shape

On every rerun the output changes for **K-Means** algorithm, as to create a more stable and better clusters **K-Means++** was introduced.

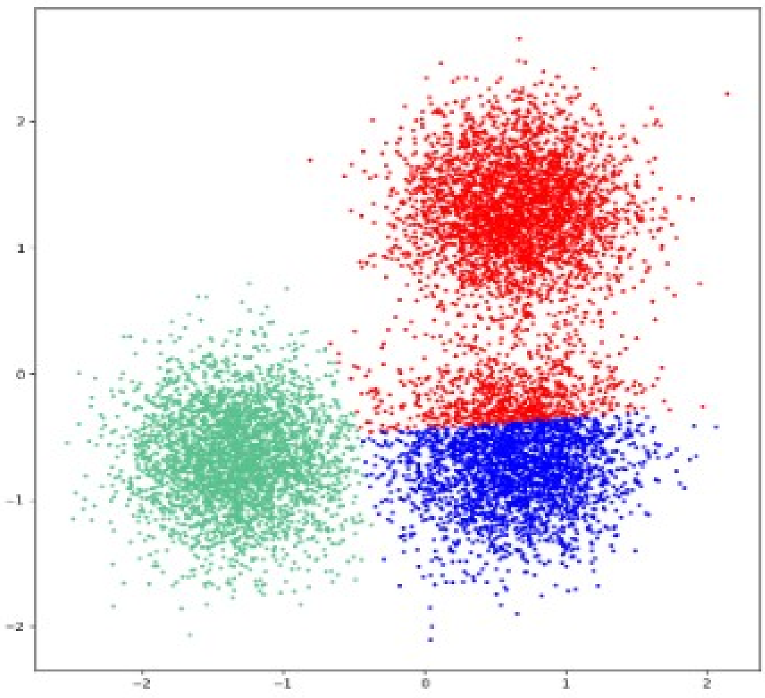
Output

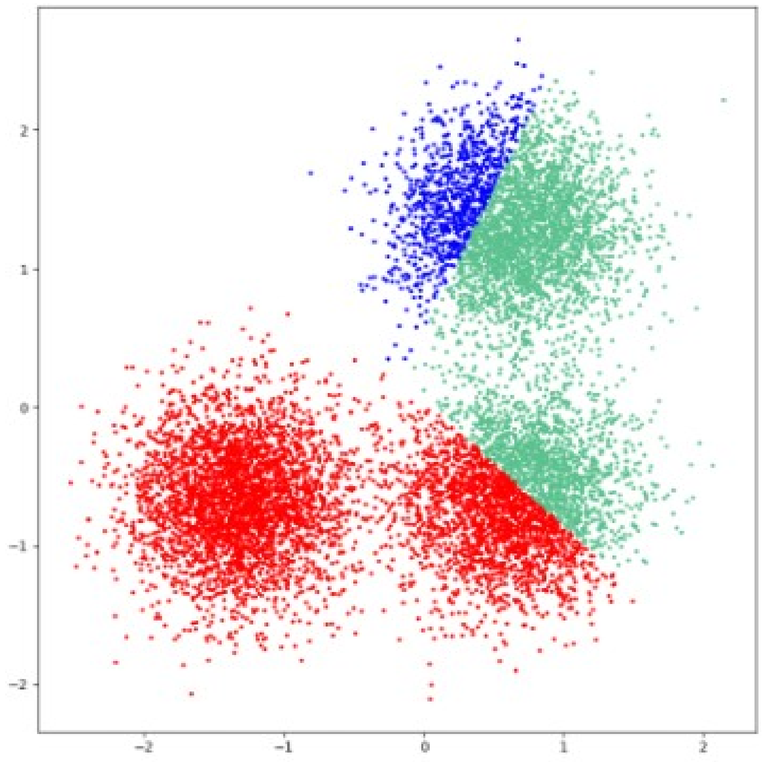
1st Iteration

Centers = 3

2nd Iteration

Centers = 3





Time of execution: 0.0124521920010011 s

Time of execution: 0.01414068800149835 s

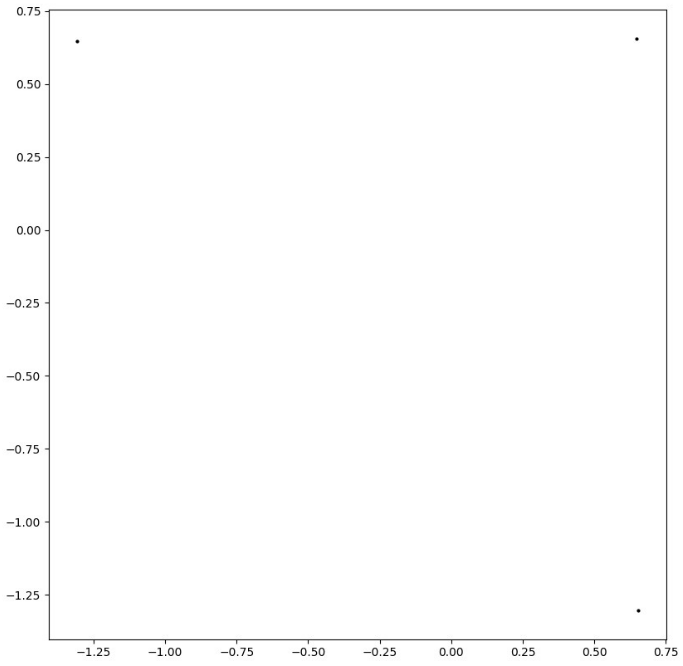
K-Means++

It is identical to **K-Means** except in this the points which have the most distance between them are selected as the centers.
This provides the centers to be same on rerunning and thus providing us with a clearer clustering which doesn’t change with iterations.
This solves the issue of random allocation by selecting the farthest points as the initial centroids thus creating more meaningful and better clusters and most of the libraries use **K-Means++** by default like Sklearn, Scipy.

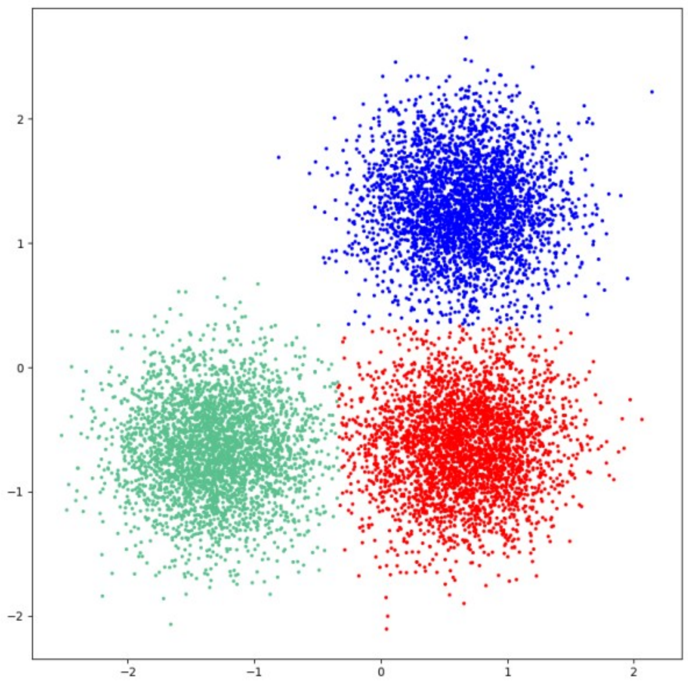
Drawbacks
The algorithm doesn’t work well with multi-dimension data.
Chance of running into local minima phenomena.
Normalization is required.
Equal treatment to outlines i.e. equal weights to all data points.
Cannot cluster arbitrary shapes.

Output

Initial points



Centers: 3



Time of execution: 0.03704648299935798

Mean shift

It is an algorithm which works on the principal of Kernel Density Estimation (**KDE**) by placing a kernel over all the points on the data, it is Non-Parametric/Density with a flat clustering i.e. which gives us a single grouping or partitioning of our data it can make use of different types of Kernel like Gaussian, Triangular, Flat (**Parzen window**) etc. it is mode seeking algorithm as it uses weights in the data i.e. all the points have different weight for the algorithm points closer to the denser region has a higher value and clustered accordingly with the kernel; It works well with data in higher dimensions.
It requires one parameter which is radius of the sphere (ε).

Kernel types

Flat kernel

$$k(x) = \begin{cases} 1 & \text{if } x \leq \lambda \\ 0 & \text{if } x > \lambda \end{cases}$$

Gaussian kernel

$$k(x) = e^{-\frac{x}{2\sigma^2}}$$

Equation for the Mean Shift algorithm

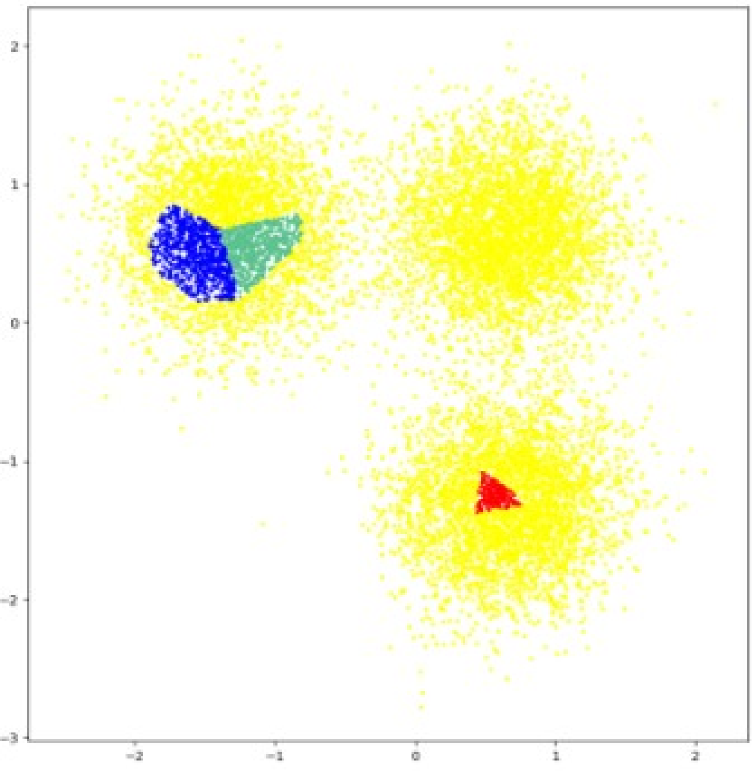
$$f(x) = \sum_i K(x - x_i) = \sum_i k\left(\frac{\|x - x_i\|^2}{h^2}\right)$$

h is the Kernel width parameter (ε).

Drawbacks
It requires more computational power.
It takes considerable more time to run.
It requires more data to work properly.
It doesn’t allow selection of number of clusters.
It is sensitive to the ε parameter in clustering.

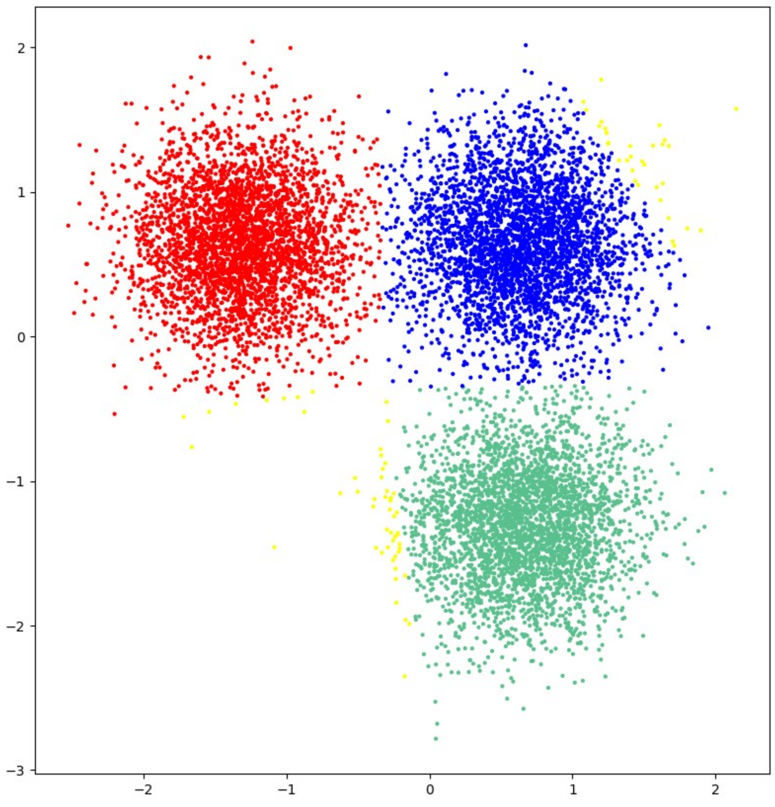
Outputs

ε = 0.01



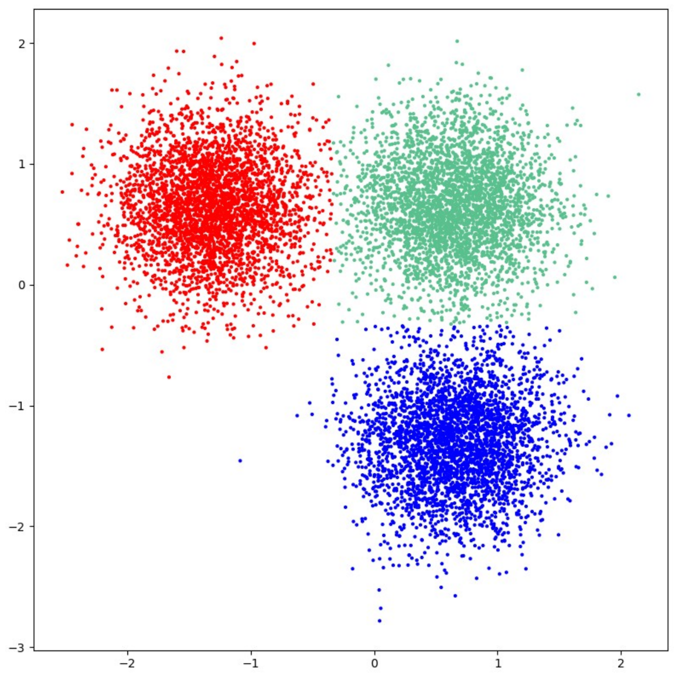
Time of execution: 0.3812082719996397 s

ε = 0.05



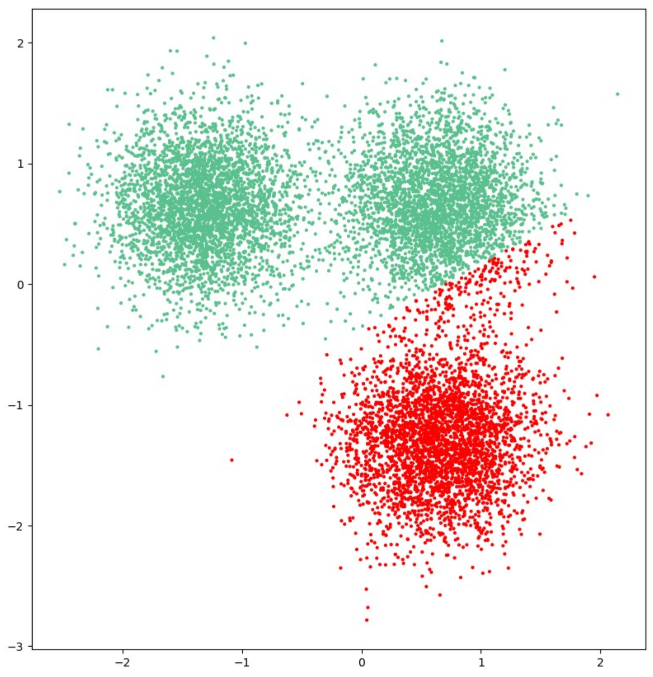
Time of execution: 1.9592083800007458 s

ε = 0.2



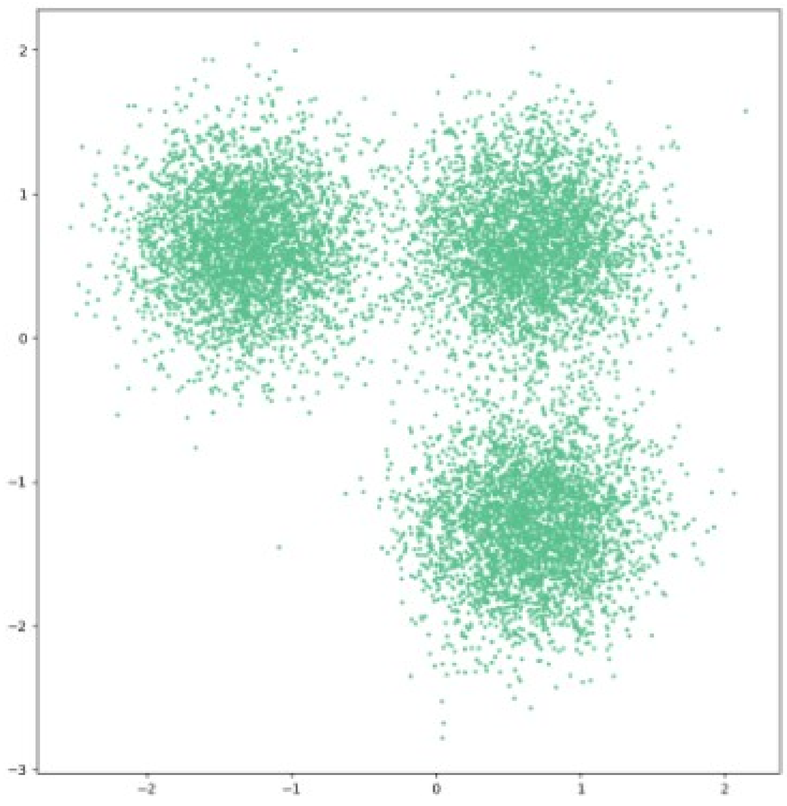
Time of execution: 7.562872915999833 s

ε = 0.34



Time of execution: 12.30047987799935 s

ε = 0.39



Time of execution: 13.45363450900004 s

Density–based spatial clustering of applications with noise (DBSCAN)

It works on the notion of density reachability, it is a data clustering Non-Parametric/Density with a flat clustering algorithm, it works by finding all points satisfying the minimum points parameter are **Core points** and then finding the relationship between the points if they are connected they form a cluster and points which are far away are classified as **Noise**, the points which are connected to the core point but don't have enough points to be classified as a Core point are called **Border points**, all the points in a cluster are mutually density connected thus forming clusters based on partitioning of data in mutual reachability distance.

It requires two parameters Minimum points and Radius (ε) .

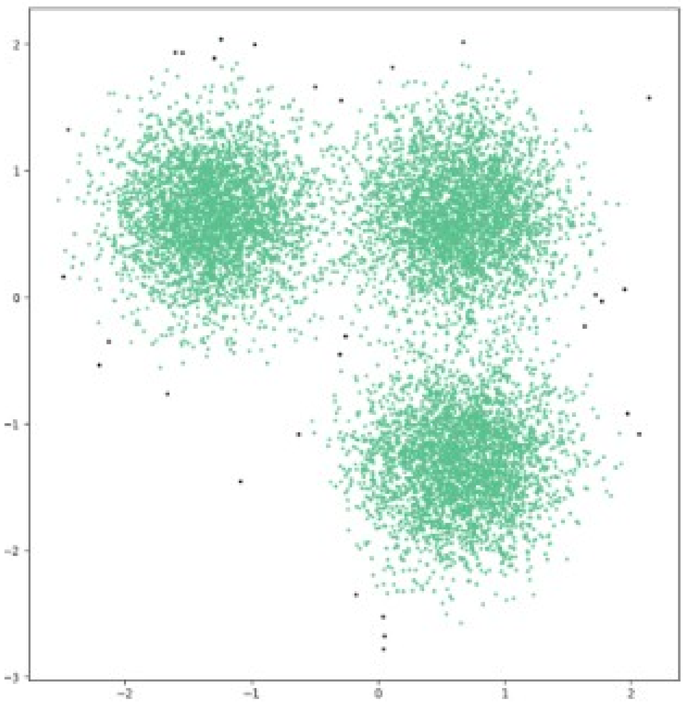
Drawbacks

Sensitive to clustering parameters ε and Minimum Points.
It is indifferent to datasets having altering densities.
Fails to identify cluster if density varies.
Fails to identity cluster if the data is sparse.
Not partition-able for multiprocessor systems.

Output

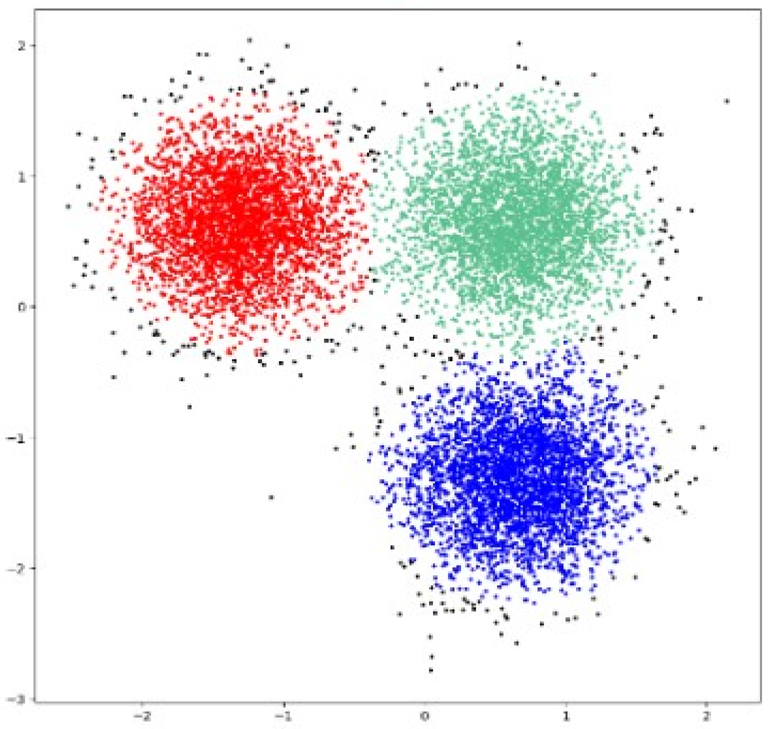
For Minimum number of points = 50

ε = 0.3



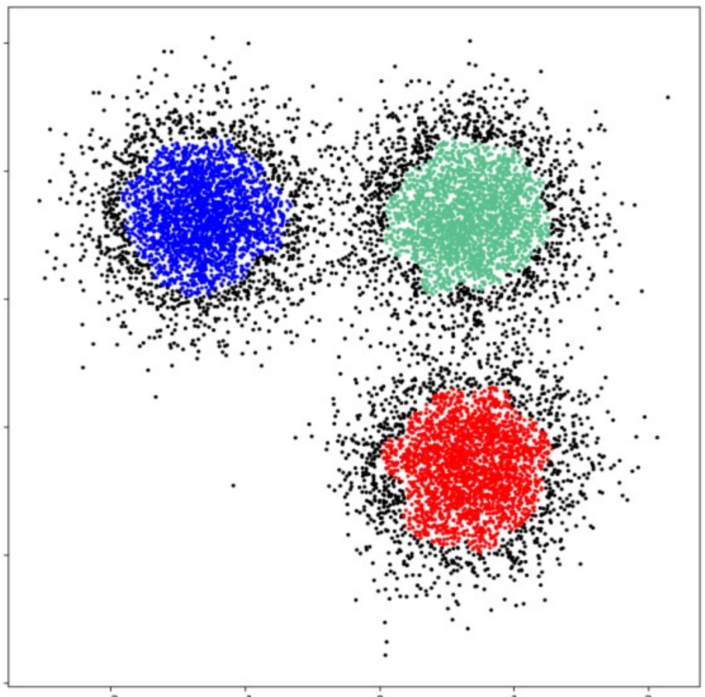
Time of execution: 0.16023094600041077 s

ε = 0.2



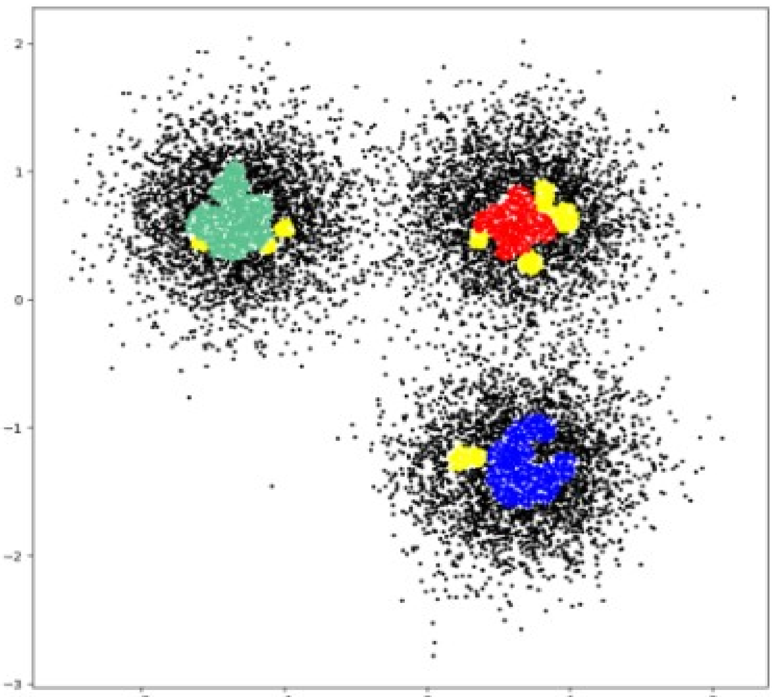
Time of execution: 0.09683981900070648 s

ε = 0.1



Time of execution: 0.05325513200023124 s

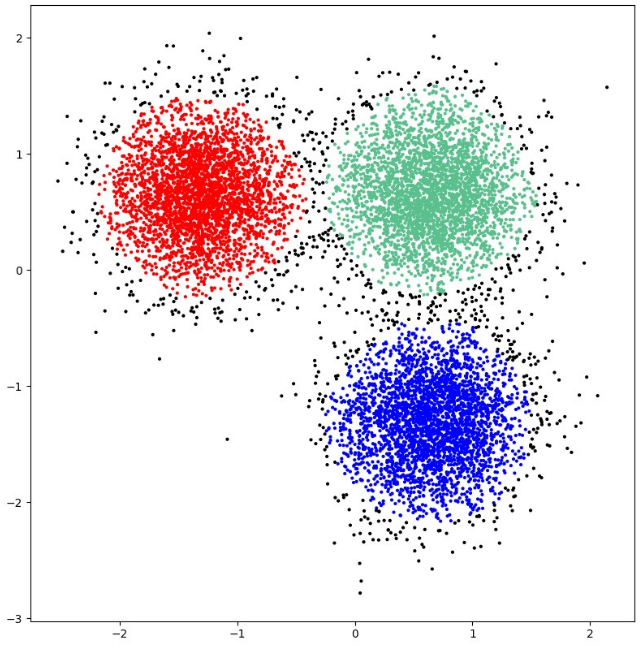
ε = 0.07



Time of execution: 0.04245215199989616 s

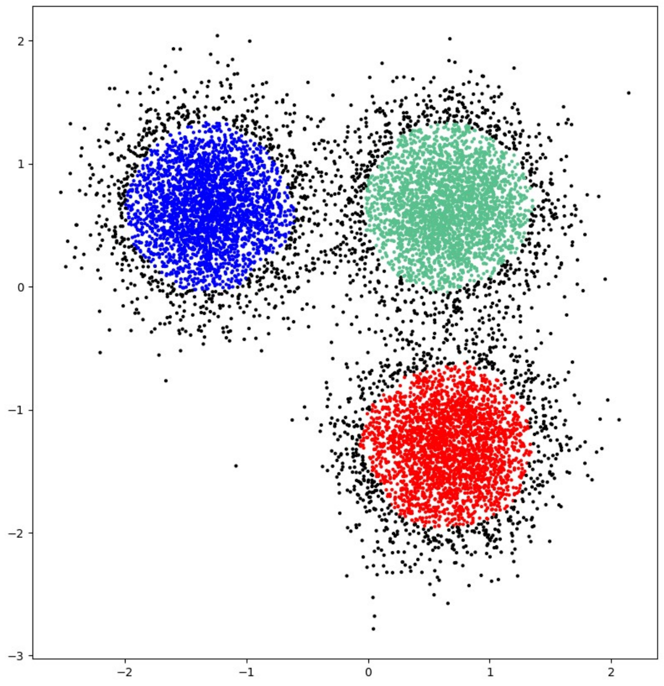
for ε = 0.2

Minimum number of points = 100



Time of execution: 0.09151794399986102 s

Minimum number of points = 200



Time of execution: 0.09815092499957245 s

Hierarchical density–based spatial clustering of applications with noise (HDBSCAN)

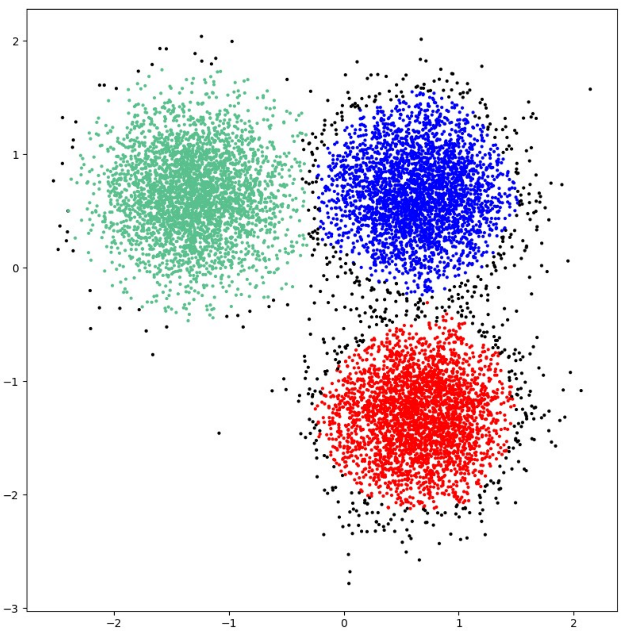
It performs **DBSCAN** over varying ε value and integrates the result to find a clustering that gives the best stability over ε. It is a Non-Parametric/Density with a hierarchical clustering algorithm This allows it to work over the data with varying densities and be more robust to parameter selection. It works by trying to find ε for which the point has enough Minimum points to be clustered by **DBSCAN** and after that it uses hierarchical clustering to form the final clusters.

It requires one parameter Minimum Points.

Drawbacks
Requires more processing power.
Takes more time than traditional **DBSCAN**.
Sensitive to the clustering parameters Minimum points.

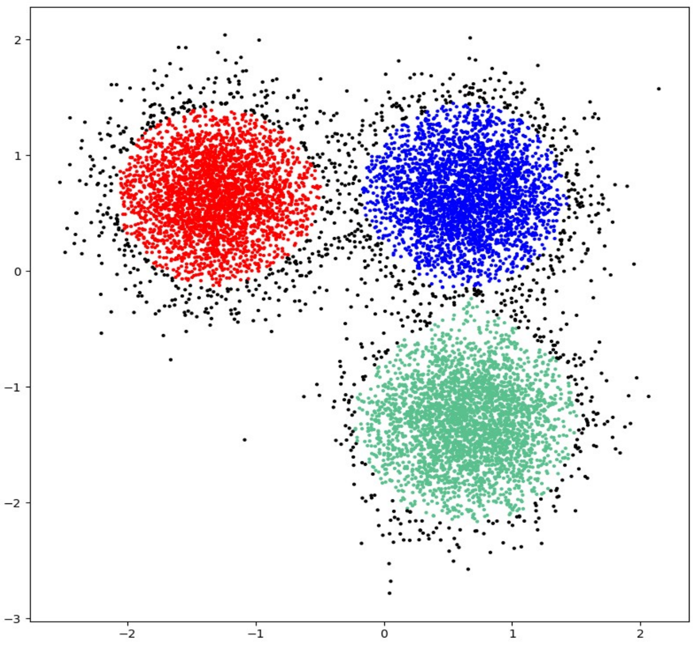
Output

For Minimum number of points = 50



Time of execution: 0.1841256509997038 s

For Minimum number of points = 200



Time of execution: 0.33597995300078765 s

By Saumya Vilas Roy
Email: saumya@criptext.com

All the algorithms ran on a Intel i7-9750H processor (6 cores @ 4.5 GHz) and NVIDIA GeForce GTX 1650 Mobile / Max-Q (3963 MiB) on Kernel 5.15.15-76051515-generic with Pop!_OS 21.10 x86_64 using Atom IDE on 15852 MiB of Memory from Scipy, Sklearn and timing has been reported by timeit library and plotted with the help of matplotlib.
Data generation was done using Sklearn