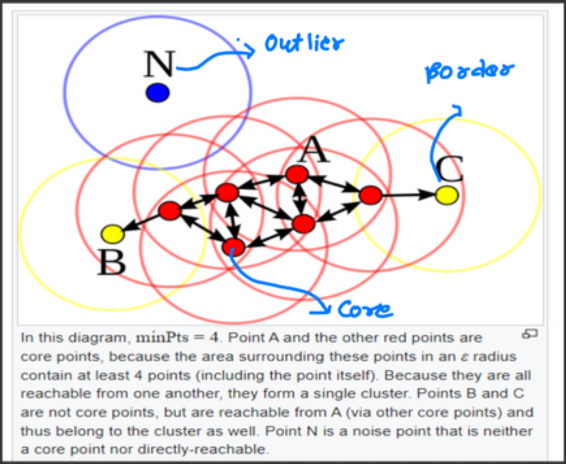
Complete Dbscan Clustering

★ #Value_freeContent ★

3. DB-SCAN CLUSTERING

↳ Robust to outliers



In this diagram, minPts = 4. Point A and the other red points are core points, because the area surrounding these points in an ε radius contain at least 4 points (including the point itself). Because they are all reachable from one another, they form a single cluster. Points B and C are not core points, but are reachable from A (via other core points) and thus belong to the cluster as well. Point N is a noise point that is neither a core point nor directly-reachable.

● → Core points

● → Border points

● → Noise / Outlier

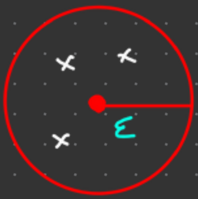
Important Hyper parameter

① MinPts (Minimum Points) = 4

② ε = radius (using to draw circle)

Core points :

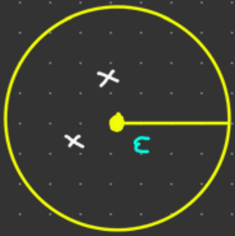
↳ No. of points within the ε should be $\geq$ MinPts



core point → also counts

Border Points :-

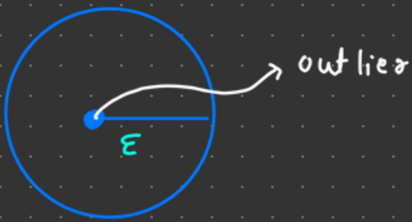
No. of data points within the radius will be < MinPts = 4



Noise / outliers :-

It Robust the outliers in DbSCAN

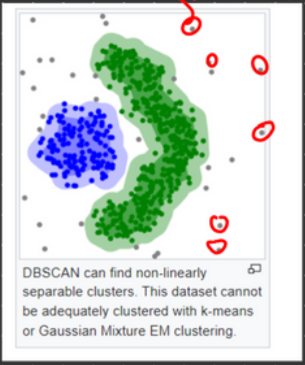
Noise → There is No points in the circle



These 3 Techniques helpful in Non linear Clustering

Some Example after we apply Dbscan Clustering :-

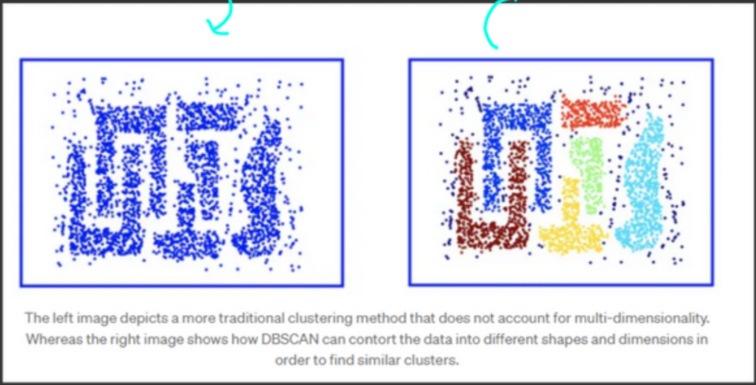
outliers not consider



DBSCAN can find non-linearly separable clusters. This dataset cannot be adequately clustered with k-means or Gaussian Mixture EM clustering.

K Means

Dbscan



The left image depicts a more traditional clustering method that does not account for multi-dimensionality. Whereas the right image shows how DBSCAN can contort the data into different shapes and dimensions in order to find similar clusters.

dfn1j4x6z

October 18, 2023

1 DBSCAN Algorithms Implementation

[1]:

```
# importing the libraries
from sklearn.cluster import DBSCAN
from sklearn.datasets import make_moons     ##creating dbscan dataset
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

[2]:

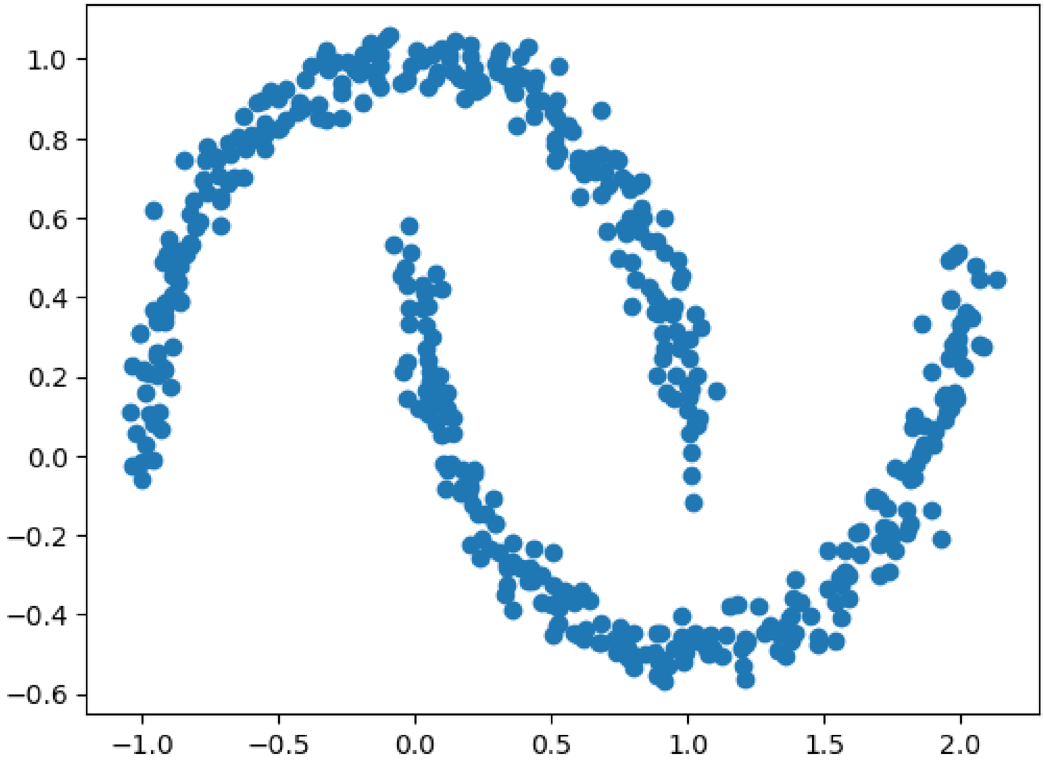
```
x,y = make_moons(n_samples= 500, noise= 0.05)
```

[5]:

```
plt.scatter(x[:,0], x[:,1])      ##Non-linear data
```

[5]:

```
<matplotlib.collections.PathCollection at 0x7fc4b96a1720>
```



[6]:

```
# Applying Standard scalling
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
```

2 Applying Dbscan

[7]:

```
# importing the DBSCAN
from sklearn.cluster import DBSCAN
```

[9]:

```
dbcan = DBSCAN(eps= 0.5)
dbcan.fit(x_scaled)
```

[9]:

```
DBSCAN()
```

[10]:

```
dbcan.labels_
```

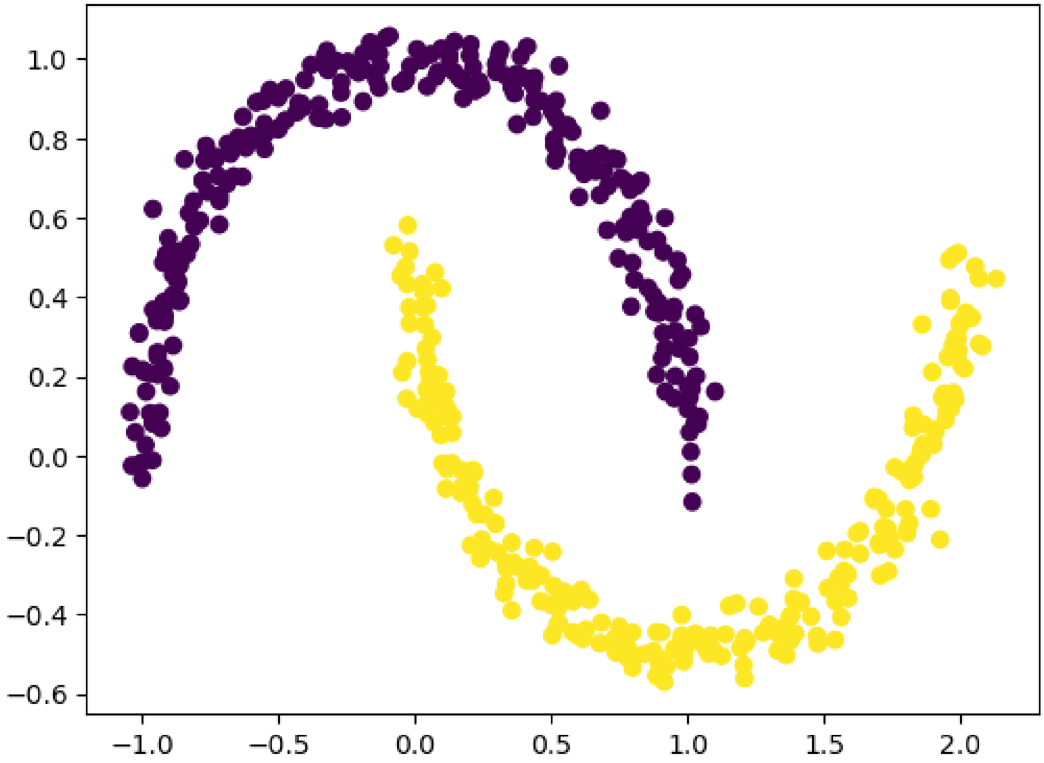
[10]:

```
array([0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0,
       0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0,
       1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0,
       1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
       0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1,
       0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
       1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1,
       0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0,
       0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0,
       1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0,
       0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1,
       0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0,
       0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0,
       0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1,
       1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1,
       1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0,
       0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0,
       0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0,
       0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1,
       1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1])
```

[11]:

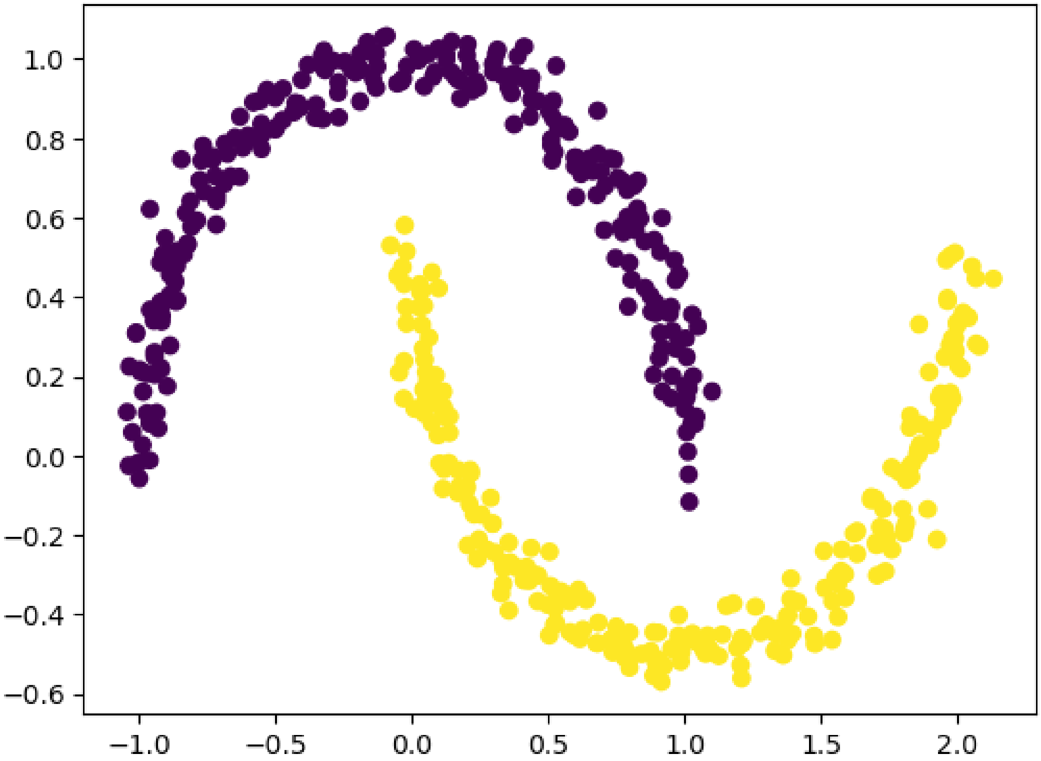
```
# ploting
plt.scatter(x[:,0], x[:,1], c = dbcan.labels_)
```

[11]: <matplotlib.collections.PathCollection at 0x7fc4b0eb4850>



[12]:
```
plt.scatter(x[:,0], x[:,1], c = y)
```

[12]: <matplotlib.collections.PathCollection at 0x7fc4b0ed62f0>



[ ]:
```
# Observations:
when we apply Kmeans on this dataset then it will consider it in a one cluster but when we apply DBSCAN on this data
then we got two beautiful cluster. it shows that your
```

[ ]:
```
```

[ ]:
```
```