Implementation of DBSCAN in Python Programming

Hey there coder! DBSCAN is a very famous clustering algorithm because, unlike other clustering algorithms like Kmeans, it can correctly cluster complex data shapes. In this tutorial, we will understand the DBSCAN algorithm and learn how to implement the same in Python programming language.

# Introduction to DBSCAN Algorithm

DBSCAN is a density-based clustering algorithm that works on the assumption that clusters are dense regions in space separated by regions of lower density.

DBSCAN computes a matrix of distances between the different points in the space (Euclidean distance is generally used). Then, the algorithm requires only two parameters: epsilon and minPoints. Epsilon is the radius of the circle to be created around each data point to check the density and minPoints is the minimum number of data points required inside that circle for that data point to be classified as a Core point.

Taking into account these parameters, the algorithm classifies each point as one out of three categories: core point, border point, and noise. Core points are those points that comply with the density conditions that we have set. Border points are those points that are close to other core points that do not meet the density conditions. Noise points are the points that do not meet the density conditions and, in their radius, do not have other points.

As a result, different core points may emerge since there may be several density zones. Each of those core points will belong to one separate cluster. Then the algorithm assigns points to each core forming the clusters.

# Advantages and disadvantages of DBSCAN

Some of the advantages of the algorithm include:

1. Able to detect complex geometric shapes that other clustering models fail to detect.
2. It remains affected by the outliers
3. The number of clusters is generated automatically.

Along with all the advantages, the algorithm also has one major disadvantage which is: The algorithm is not deterministic which implies that the same point can be reached by various clusters and, therefore, in different executions it can be assigned to different clusters.

# Implementation of the DBSCAN Algorithm

This section of the tutorial will cover the implementation of the DBSCAN algorithm in several steps.

## Step 1 – Importing Libraries and modules

Let’s start by importing the necessary libraries.

import NumPy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import math

from sklearn.preprocessing import normalize

from sklearn.cluster import KMeans, DBSCAN

## Step 2 – Create a dataset for the implementation

We will be creating a dataset with the help of the make\_circle function which takes the radius, the number of points, and some noise as arguments and returns an array of data points that when plotted will be forming a circle. The function will make use of the sin and cosine curves as shown in the code snippet below.

def make\_circle(r, n, noise = 30):

return [(math.cos(2\*math.pi/n\*x)\*r+np.random.normal(-noise,noise), math.sin(2\*math.pi/n\*x)\*r+np.random.normal(-noise,noise)) for x in range(1,n+1)]

# make concentric circles for the dataset - (radius, number of points, noise)

circle\_1 = make\_circle(100, 300, 30)

circle\_2 = make\_circle(300, 700, 30)

circle\_3 = make\_circle(500, 500, 30)

circle\_4 = make\_circle(700, 900, 30)

circle\_5 = make\_circle(900, 1000, 30)

# add some noise in the dataset

noise = [(np.random.randint(-1000,1000),np.random.randint(-1000,1000)) for i in range(1000)]

One circle obviously won’t be sufficient to see the efficiency and proper working of the clustering algorithm. Therefore, we will be creating five concentric circles of different radii. Along with this, we will be adding some noise to this data so that we can see the behavior of the algorithm in presence of some noise. We convert the separate datasets along with noise into a single data frame using the code snippet below.

# Convert the array to a dataframe

def arrray\_to\_df(arr, i):

df = pd.DataFrame(arr)

df['cluster'] = str(i)

return df

# create data from the circle data and noise data

data = [arrray\_to\_df(arr, i) for i, arr in enumerate([circle\_1, circle\_2, circle\_3,circle\_4,circle\_5, noise])]

# combine the spearate datasets

data = pd.concat(data)

data.columns = ['x\_values', 'y\_values', 'cluster']

# normalize the dataset

data\_norm = normalize(data)

## Step 3 – Visualizing the Initial Dataset Formed

We will be plotting the points in the dataset and look at how they look in the space. We will be using the scatter plot for plotting these data points. The code snippet for the same is below:

plt.rcParams['figure.figsize'] = [15,15]

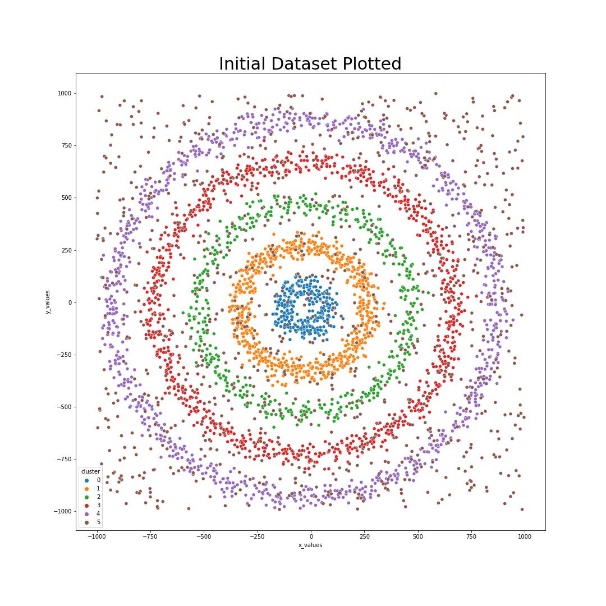
sns.scatterplot(data = data, x = 'x\_values',y = 'y\_values',hue = 'cluster')

plt.title("Initial Dataset Plotted",fontsize=30)

plt.savefig('Output\_Images/001. DBSCAN/001.Initial\_Data\_Plotted.jpg')

plt.show()

After the code execution, the output comes out to be the plot which is displayed below:



## Step 4 – Applying DBSCAN Algorithm to the dataset

The DBSCAN algorithm can be found inside the **sklearn.cluster** module as the **DBSCAN function**. Like the rest of the clustering models inside the sklearn module, the DBSCAN algorithm consists of two major steps:

1. Initially, the fit function is applied and,
2. then the predictions are made using the predict function.

An easier option is to combine the above-mentioned two steps and implement both with just one fit\_predict function using the code snippet below. We will also visualize the results of the clustering algorithm.

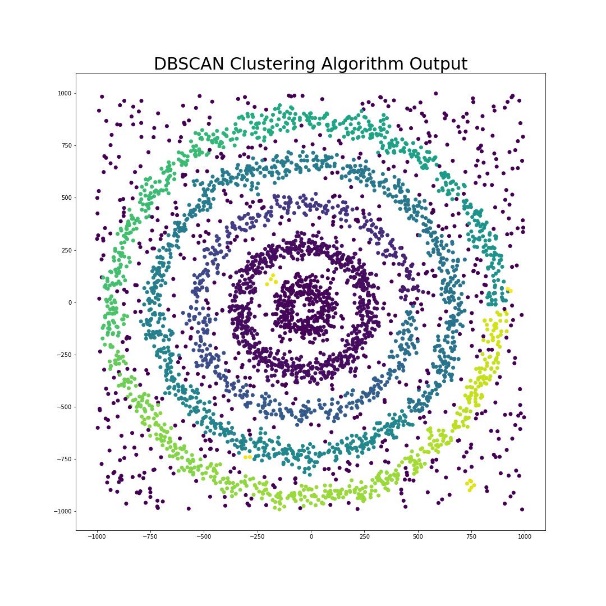
data['dbscan\_cluster'] = DBSCAN(eps=32, min\_samples=5).fit\_predict(data[['x\_values', 'y\_values']])

plt.scatter(data['x\_values'],data['y\_values'],c = data['dbscan\_cluster'])

plt.title("DBSCAN Clustering Algorithm Output",fontsize=30)

plt.savefig('Output\_Images/001. DBSCAN/002.DBSCANs\_Clustering\_Output.jpg')

plt.show()



As we can see, the DBSCAN algorithm has been able to adequately cluster the data, as we have taken into account the complex shapes of the data and the algorithm has not considered noise in any of the clusters.

# Implementation of the Kmeans algorithm and comparing the results

In this section we will be implementing Kmeans Clustering algorithm and comparing the results of the same with the DBSCAN algorithm. The code and output for the same is below:

preds = KMeans(n\_clusters = 5, random\_state =123).fit\_predict(data\_norm)

cols = {0: 'r',1: 'g',2: 'b',3:'c',4:'m'}

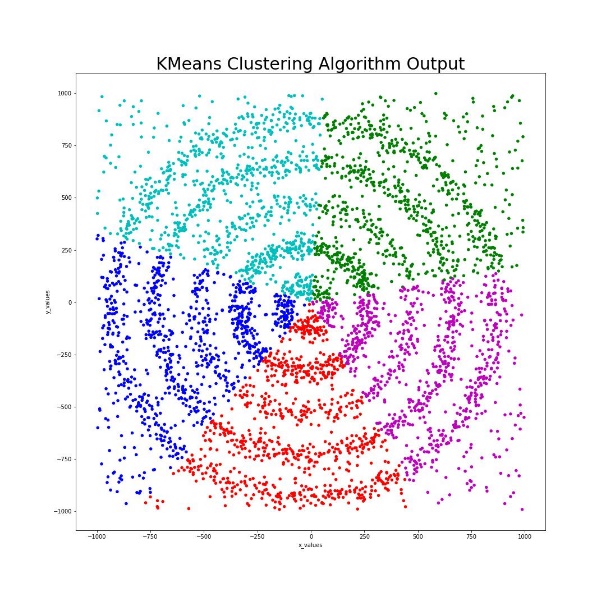
data['kmeans\_pred'] = [cols.get(pred) for pred in preds]

data.plot.scatter('x\_values', 'y\_values', c='kmeans\_pred')

plt.title("KMeans Clustering Algorithm Output",fontsize=30)

plt.savefig('Output\_Images/001. DBSCAN/003.KMeans\_Clustering\_Output.jpg')

plt.show()



Now, you can see, that Kmeans has done a very bad clustering which is very evident by the fact that it did not cluster according to the complex shapes of the model and has not taken into account that there are outliers as well.

# Conclusion

I hope this tutorial helped you learn and understand more about the DBSCAN data clustering algorithm. The tutorial also showcased how it is useful in comparison to the Kmeans Clustering algorithm. I would also like to mention that there also exists a much better and more recent version of the DBSCAN algorithm known as HDBSCAN (Hierarchical + DBSCAN) which is much faster and more accurate than DBSCAN.

Keep learning and read the following tutorials as well:

1. Blog 1
2. Blog 2
3. Blog 3