****

**Fall 2022**

**Project Title: Implementation of DBSCAN Algorithm**

**Submitted to**

**Jesan Ahmed Ovi**

**Senior Lecturer**

**Department of Computer Science & Engineering**

**Submitted by**

|  |  |
| --- | --- |
| **Student ID** | **Student Name** |
| **2019-1-60-022** | **Shuvo Kumar Das** |
| **2019-1-60-210** | **Anika Anmol Sara** |
| **2019-2-60-006** | **Syada Tasfia Rahman** |
| **2019-2-60-037** | **Ashiqur Rahman Shohan** |
| **2019-2-60-092** | **Amir Khabbab Ahammed** |

1. **Problem Statement:**

To solve the difficulty with non-globular clusters and clusters of multiple sizes DBSCAN is used. DBSCAN is an unsupervised clustering algorithm. It can deal with clusters of various sizes and configurations and is not significantly influenced by noise or outliers. Unless there is a specific outcome variable that has to be predicted, it should be used.

The goal of DBSCAN is to find the neighborhoods of data points that exceed a certain density threshold. It requires two major inputs: minimum point and epsilon to find the core point, boundary point and noise point. Using these parameters will detect areas where points are concentrated and where they are separated by areas that are empty or sparse. Points that are not part of a cluster are labeled as noise.

1. **System Requirement**

Processor: Core i5

RAM: 8GB

Operating System: Windows 10

IDE: Colaboratory Web IDE

1. **System Design**

In terms of the algorithm, density is nothing but a number of points which are located in a given area. The algorithm has two major inputs: minimum point and epsilon(radius) which are taken from the user.

1. Epsilon (

It is used for the max radius of a neighborhood or a circle.

1. MinPts:

Minimum number of neighbors (data points) within eps radius. Larger the dataset, the larger value of MinPts must be chosen. As a general rule, the minimum MinPts can be derived from the number of dimensions D in the dataset as, MinPts >= D+1. The minimum value of MinPts must be chosen at least 3.

With the help of these two points we will find the core point, boundary point and noise point.

1. Core Point:

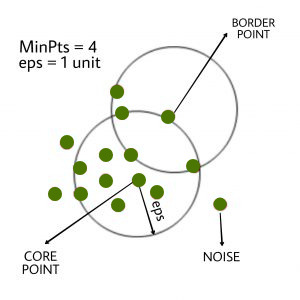
The point at which the circle is made with this as the center is called the core point. To be a core point, the condition of the minimum point must be satisfied. Core point is a strong constituent of a cluster.

1. Boundary Point:

After extracting the core points, it should be seen whether the remaining points are boundary points or noise points. A point is a boundary point if it is a neighborhood of the core point.

1. Noise point:

Noise point means outlier. It is the point which is neither core point nor boundary point.



**Working flow of DBScan Algorithm**

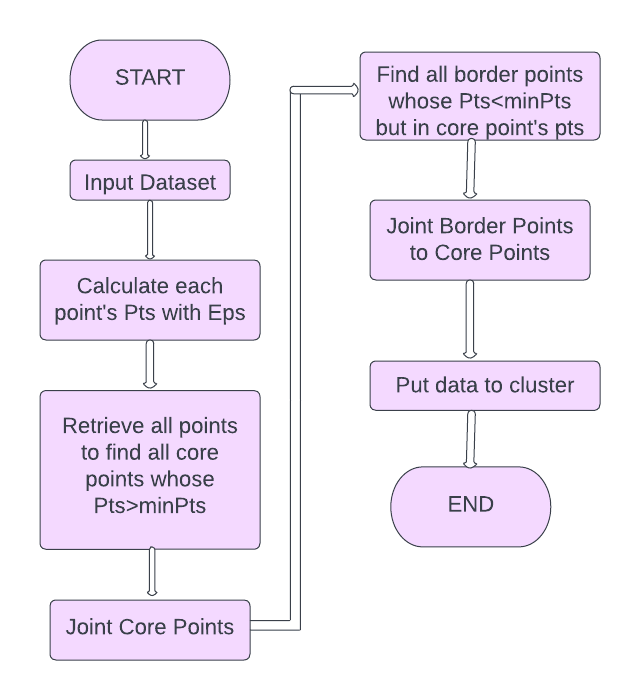
****

Fig:Flowchart of DBSCAN Algorithm

**DBSCAN clustering algorithm in pseudocode:**

DBSCAN(D, eps, MinPts){

C = 0

for each unvisited point p in dataset D

mark P as visited

I N = getNeighbour(P,eps)

If size of (N) < MinPts

mark P as NOISE

else

C = next cluster

expand Cluster(P,N,C, eps,MinPts)

expandCluster(P,N,C,eps,MinPts)

add P to cluster C

For each point P in N

if P’ is not visited

mark P’ as visited

N = getNeighbors(P’,eps)

If size of(N’) >= MinPts

N = N joined with N’

if P' is not a member of any cluster

add P' to cluster C

**4. Implementation**

We have implemented the DBScan clustering algorithm. The important functions of the algorithm which is used is below:

def check\_point(eps,minPts, df, index):

x, y = df.iloc[index]['X'] , df.iloc[index]['Y']

neighbors = df[((np.abs(x - df['X']) <= eps) & (np.abs(y - df['Y']) <= eps)) & (df.index != index)]

#print(neighbors)

if len(neighbors) >= minPts:

#return (neighbors\_index, is\_core, is\_border, is\_noise)

return (neighbors.index , True, False, False)

elif (len(neighbors) < minPts) and len(neighbors) > 0:

#return (neighbors\_index, is\_core, is\_border, is\_noise)

return (neighbors.index , False, True, False)

elif len(neighbors) == 0:

#return (neighbors\_index, is\_core, is\_border, is\_noise)

return (neighbors.index , False, False, True)

In this check\_point function check if a point is core point or border or noise. The function takes radius(eps), minimum point(minPts), dataframe(df), index(Index) as parameters and checks that index’s point is core point or border or noise. If the number of neighbors of a point is greater than minPts then that point is the core point, if the number of neighbors is less than minPts and greater than zero then that point is the border point and if there is no neighbor for a point then that point is the noise(outlier).

Finally, the function returns neighbor’s index, core point’s result, border point’s result, noise’s result.

def fit\_data(eps, minPts, df):

#initiating cluster number

cluster\_count = 1

current\_set = set()

unvisited = list(df.index)

clusters = []

while len(unvisited) > 0:

first\_point = True

current\_set.add(random.choice(unvisited))

while len(current\_set) > 0:

curr\_idx = current\_set.pop()

neigh\_indexes, iscore, isborder, isnoise = check\_point(eps, minPts, df, curr\_idx)

if (isborder==True & first\_point==True):

clusters.append((curr\_idx, 0))

clusters.extend(list(zip(neigh\_indexes, [0 for \_ in range(len(neigh\_indexes))])))

unvisited.remove(curr\_idx)

unvisited = [i for i in unvisited if i not in neigh\_indexes]

continue

unvisited.remove(curr\_idx)

neigh\_indexes = set(neigh\_indexes) & set(unvisited)

if iscore==True:

first\_point = False

clusters.append((curr\_idx,cluster\_count))

current\_set.update(neigh\_indexes)

elif isborder==True:

clusters.append((curr\_idx,cluster\_count))

continue

elif isnoise==True:

clusters.append((curr\_idx, 0))

continue

if first\_point==False:

cluster\_count+=1

return clusters

This fit\_data function mainly classifies the points into the specific clusters. The function takes radius(eps), minimum point(minPts), dataframe(df) as parameters. Initially, we initialized the cluster\_count=1 which is the number of clusters. Then we take a set (current\_set ) which store same cluster’s index, a list(unvisited) which store the unvisited index, initially, we take all indexes of the dataframe as unvisited and we take a list(cluster) which store pair of the index and that index cluster level.

While all the indexes of the data frame are not visited it takes a random index from the unvisited list and stores current\_set set. While all the indexes of current\_set are not visited it takes an index randomly and checks that index’s point is core point or border or noise.

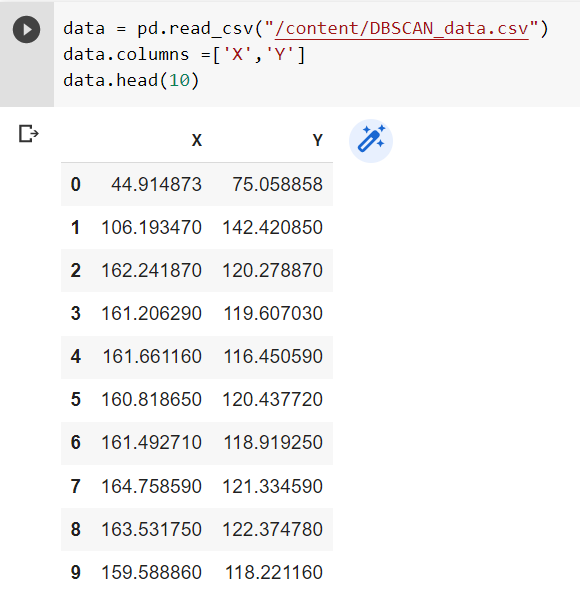
If a point is a border point and that point checks as the first point then this model considers the point as outliers. If a point is a core point then the model categorizes that point and its neighbors in the same cluster and checks for its neighbors. If a point is a border point and that point checks as the last point then this model considers the point in the same cluster as its core point’s. If a point is noise then it will classify it as outlier.

While all the indexes of current\_set are visited the cluster number will be increased.

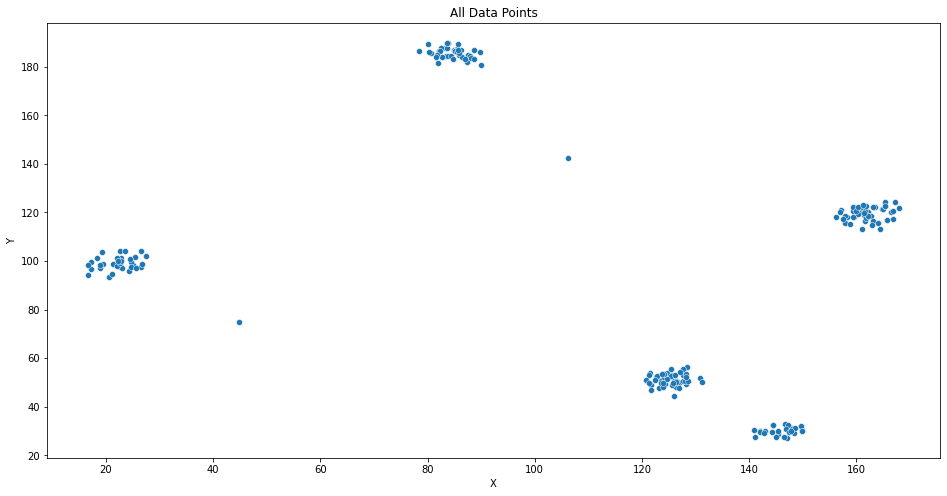
Finally, the function returns all the data point’s index and cluster level.

**5. Testing Result**

We have got a dataset from kaggle and read the dataset and set the column names as X and Y. Then print the dataset to see points.

****

Here is the scatter plot of our dataset.

****

**#radius of the circle defined**

**eps = 3**

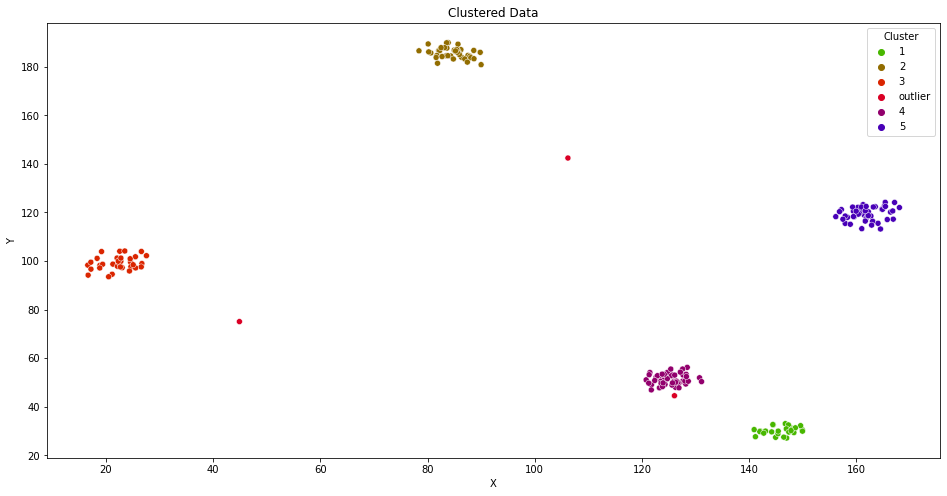
**#minimum neighboring points**

**minPts = 3**

When we have taken the input of radius(eps) as 3 and minPts as 3, we have got the total number of clusters is 5 and number of outliers is 3.

****

In the below scatter plot, there are 5 clusters and outliers. Outliers are blue in color. Every different clusters’ points are different in color.

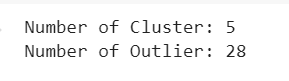
****

**eps = 2**

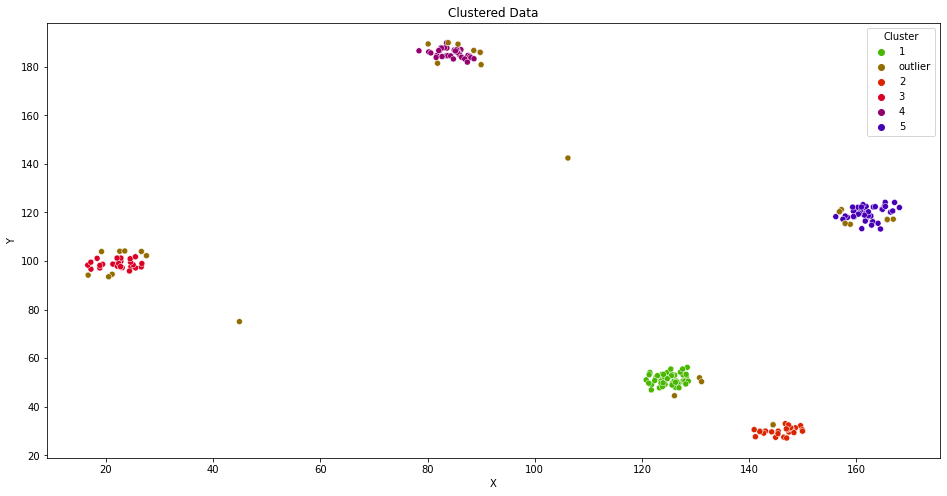
**#minimum neighboring points**

**minPts = 3**

When we have taken the input of radius(eps) as 2 and minPts as 3, we have got the total number of clusters is 5 and number of outliers is 28.

****

In the below scatter plot, there are 5 clusters and outliers. Outliers are brown in color. Every different clusters’ points are different in color.

****

**6. Limitations**

* DBSCAN cannot cluster data-sets with large differences in densities well, since then the minPts-eps combination cannot be chosen appropriately for all clusters.
* Choosing a meaningful eps value can be difficult if the data isn't well understood.
* DBSCAN is not entirely deterministic. That's because the algorithm starts with a random point. Therefore border points that are reachable from more than one cluster can be part of either cluster.
* Struggles with high dimensionality data. If given data with too many dimensions, DBSCAN suffers.
* As this model is picking values randomly so that it can generate different clusters at different times.

**7. Conclusion and Future Scope**

The most well-liked data mining technique is clustering, which allows for the grouping of data kinds that are similar and different in order to evaluate large amounts of complex data. density-based technique Based clustering is used to group comparable and dissimilar types of data in accordance with the dataset's data density. In the density-based clustering, the most dense area is determined from which similarity approaches are used to determine the types of data that are similar and dissimilar. The EPS value, which will serve as the dataset's focal point, has been determined and is used in the DBSCAN algorithm used in this work.

To achieve utmost accuracy, the EPS value is dynamically calculated. The Euclidean distance method is used to determine how similar the data points in the datasets are to one another.