

**Project Name:**

**Density-based spatial clustering of applications with noise (DBSCAN)**

**Course: CSE477**

**Section: 01**

**Submitted By**

|  |  |
| --- | --- |
| **Name** | **Student ID** |
| **Md.Tohidul Haque Sagar** | **2019-1-60-156** |

**Submitted to**

**Jesan Ahmed Ovi**

**Senior Lecturer**

**Department of Computer Science and Engineering**

**East West University**

**Density-based spatial clustering of applications with noise (DBSCAN)**

1. **Problem Statement:**

A well-known data clustering approach that is commonly used in data mining is density-based spatial clustering of applications with noise (DBSCAN). Three subcategories of clustering activities can be performed using various approaches and techniques: Partition-based clustering, Hierarchical clustering and Density-based clustering. Density-based methods are more effective at recognizing outliers or arbitrarily formed clusters. DBSCAN is an unsupervised clustering approach based on data point density. Clusters in DBSCAN are created from dense regions and separated by density regions.

Based on a distance measurement and a minimal number of points, it groups points that are close to one another. Each group call as a cluster. To measure the distance, we use the Euclidean method. Additionally, it identifies points in low-density areas as outliers.

DBSCAN 1st finding out core point. To understanding core point we have to know two parameters of it: eps and min\_points. The eps indicates the minimum distance between points that must exist for them to be considered as a cluster. It denotes that two points are classified as neighbours if their distances are less than or equal to this value (eps). The min\_points is the minimum number of points to form a dense region.

DBSCAN clustering, there are three different kinds of points: Core-point, Border-point and Noise-point. A point is considered a core point if it has a minimum number of other points (min\_points) within a given distance (eps) of it. Border -point that has at least one Core point at a distance (eps).

The fact that core points generally have a significant number of surrounding points makes them suitable candidates for seeds in developing clusters. If non-core points are placed near enough to a core point, the cluster may also contain those non-core points. The final collection of clusters excludes noise points, which are points that do not belong to any clusters.

1. **System Requirements:**

* **Processor:** Intel or AMD processor with 64-bit support;

Recommended: 2.8 GHz or faster processor

* **RAM:** Minimum 8GB
* **GPU:** NVidia GeForce GTX 1050 or equivalent;

Recommended: NVidia GeForce GTX 1660 or Quadro T1000

* **Operating system**: Windows 7 with SP1;

Recommended: Windows 10

* **IDE**: pycharm, jupyter notebook and Internet connection required for software activation
* **Disk Storage**: 4 GB of free disk space

1. **System Design:**

**Algorithm:**

DBSCAN works by starting with a point in the dataset and then identifying all points within a distance of eps from that point. If there are at least min\_points within that distance, then a new cluster is started with those points and the process is repeated for each point in the cluster. Points that are not in a cluster or that do not have at least min\_points within a distance of eps are considered as noise.

Pseudo-code

For each point in our given the dataset

* If the point is not yet visited, mark it as visited
  + - 1. Call get\_neighbors(data, point, eps)
      2. Then call euclidean\_distance(point, p)
      3. distance compare with eps
      4. Append the point if condition true
* If the point is not a core point, then mark it as noise and move to the next point
* If the point is a core point, then start a new cluster and do a depth-first search of the surrounding points, marking them as visited and part of the cluster if they are within eps distance

In the end, return the clusters.

Also, the performance of DBSCAN is affected by the select of **eps** and **min\_points** this need to be chosen carefully to get a good cluster.

Flowchart:

Import packages

Input dataset

Checking NULL data in this data set and Visualize Data by scatter plot

Check visited or not

DBSCAN (data3, eps, min\_points)

get\_neighbors(data, point, eps)

expand\_cluster(data, point, neighbors, cluster, visited, eps, min\_points)

euclidean\_distance(point, p) < eps: neighbors.append(p)

Return clusters

Print each cluster in different colour and Visualize Data by scatter plot

1. **Implementation:**

Important parts of the code:

eps = 5

min\_points = 5

clusters = DBSCAN(data1, eps, min\_points)

def DBSCAN(data, eps, min\_points):

clusters = []

visited = set()

for point in data:

if tuple(point) in visited:

continue

visited.add(tuple(point))

neighbors = get\_neighbors(data, point, eps)

if len(neighbors) < min\_points:

continue

cluster = []

clusters.append(expand\_cluster(data, point, neighbors, cluster, visited, eps, min\_points))

return clusters

In this function, we check the point is not yet visited, mark it as visited and call get\_neighbors(data, point, eps) function.

def get\_neighbors(data, point, eps):

neighbors = []

for p in data:

if euclidean\_distance(point, p) < eps:

neighbors.append(p)

return neighbors

In this function we find the neighbors of a point within a given radius eps. Then we call euclidean\_distance(point, p).

def euclidean\_distance(point1, point2):

distance = 0

for i in range(len(point1)):

distance += (point1[i] - point2[i]) \*\* 2

return distance \*\* 0.5

Here we Calculates the Euclidean distance between two points. Distance compare with eps.

def expand\_cluster(data, point, neighbors, cluster, visited, eps, min\_points):

cluster.append(point)

visited.add(tuple(point))

i = 0

while i < len(neighbors):

neighbor = neighbors[i]

if tuple(neighbor) in visited:

i += 1

continue

visited.add(tuple(neighbor))

new\_neighbors = get\_neighbors(data, neighbor, eps)

if len(new\_neighbors) >= min\_points:

neighbors += new\_neighbors

cluster.append(neighbor)

i += 1

return cluster

In this function, expands the cluster to include density-reachable items.

clusters.append(expand\_cluster(data, point, neighbors, cluster, visited, eps, min\_points))

After creating single cluster we returned to the DBSCAN(data1, eps, min\_points) and append cluster in clusters variable.

l=len(clusters)

print("NUMBER OF CLUSTERS",l)

c=["red","blue","green","black","cyan","yellow","magenta","lightcoral", "darkorange", "olive", "teal","iolet","kyblue"]

j=0

for i in clusters:

a=pd.DataFrame(i)

b=np.array(a)

print(type(b))

B=len(b)

print("NUMBER OF item in this CLUSTER",B,"\n")

plt.scatter(b[:,0],b[:,1],s=5,c=c[j])

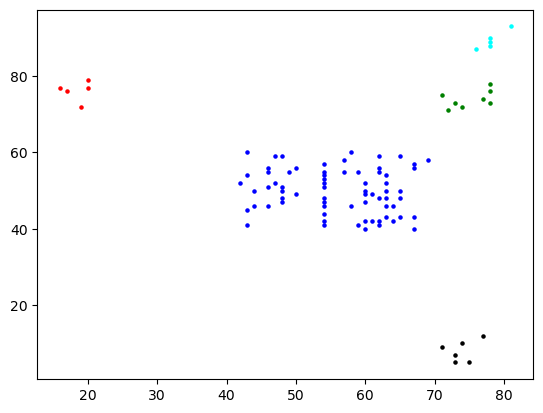
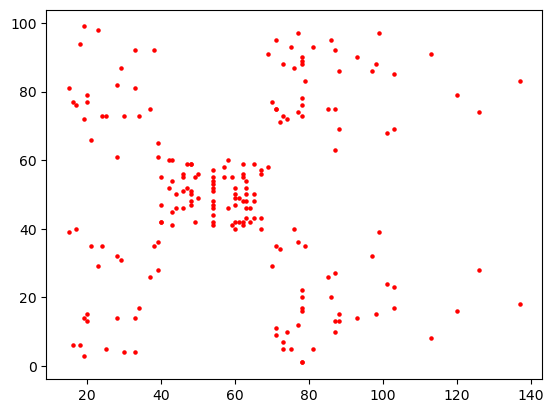
j=j+1

And here we print each cluster in different colour and Visualize Data by scatter plot.

1. **Testing Results:**

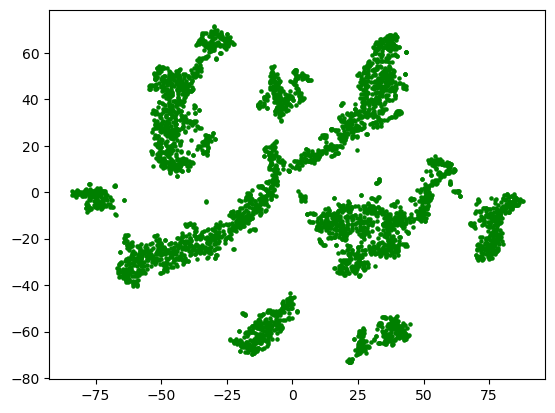
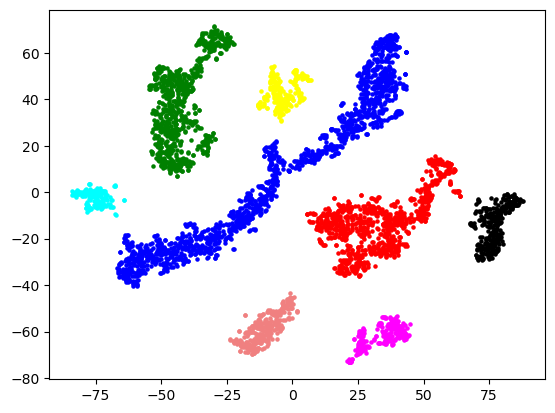
In our project, we use 2 data set which are collect from kaggle website. Our 1st data set name is “Mall\_Customers.csv” and another one is “tsne\_scores .csv”.

To pre-proposing our data we check null values in our data sets. Here I give some screenshots of my input and output in scatter plot.



Input dataset1 Output dataset1

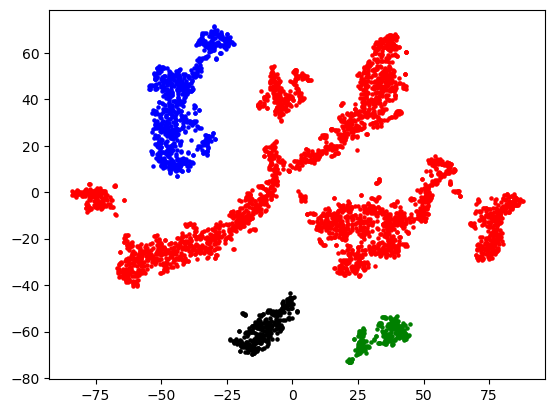
We used 200 rows, 2 columns, and a total of 200 points for our dataset1. Using our DBSCAN algorithm, we discovered 5 clusters that are represented by various colours. Number of clusters depend on eps and min\_points.



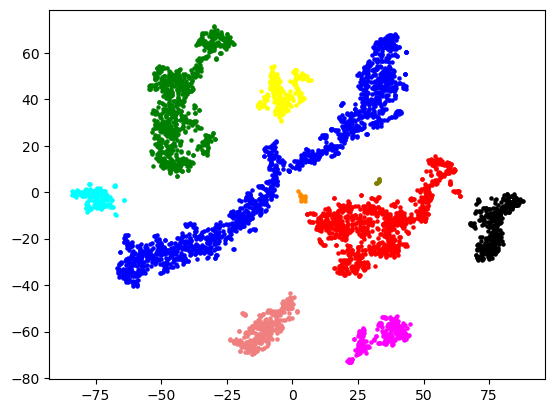
**Input dataset2 Output dataset2**

We used 4406rows, 2 columns, and a total of 4406 points for our dataset1. Using our DBSCAN algorithm, we discovered 8 clusters that are represented by various colours. Number of clusters depend on eps and min\_points. This output show when eps = 5 and min\_points = 15.

For the same dataset2, if I change eps and min\_point value then number of cluster will change. Here I give some example



Here, eps = 10, min\_points = 10 and clusters is 4



Here, eps = 5, min\_points = 5 and clusters is 10.

A larger value for "eps" will result in clusters that are more spread out and may include more noise, while a smaller value for "eps" will result in clusters that are more tightly grouped but may exclude some points that are actually part of the cluster.

The choice of eps is dependent on the data and should be made after understanding the data. Alternatively, we can test various values and pick the one that produces the best results.

In summary, eps determines the maximum distance between points in a cluster. Based on the features of the data and the desired cluster attributes, the value of eps must be selected.

1. **Future Scope:**

In this project, there is a limitation that prevents us from showing clusters in different colours if there are more than 12 of them. In future we will solve this limitation.