Weekly Report 4 - DBSCAN

Ganji Varshitha
AI20BTECH11009

Introduction

DBSCAN stands for Density-Based Spatial Clustering of Applications with Noise.
It is a density based unsupervised learning algorithm which clusters data into arbitrary shape.

Algorithm

It assumes clusters are dense regions separated by regions of low density. Hence, it identifies those regions of high densities.

Key terms

- **Density** : The number of points within a specific radius Eps.
- **Epsilon or Eps** : The radius of circle around each data point to check the density.
- **Core point** : A point is considered Core point if it has more than specific number of points(MinPts) within Eps.
- **Border point** : A point is considered Border point if it has less than MinPts but grater than 1 point within Eps.
- **Noise** : A point which does not have any points within Eps is considered noise.

Reachability and Connectivity

Reachability states if a data point can be accessed from another data point directly or indirectly, whereas Connectivity states whether two data points belong to the same cluster or not.
In the algorithm, any points can be referred as:

- **Directly density reachable** : An object q is directly density-reachable from object p if q is within the ϵ Neighbourhood of p and p is a core point.
- **Density reachable** : An object p is density-reachable from q w.r.t ϵ and MinPts if there is a chain of objects $p_1, \cdots, p_n$, with $p_1 = q, p_n = p$ such that p_{i+1} is directly density-reachable from p_i w.r.t ϵ and MinPts for all $1 \leq i \geq n$

- **Density Connectivity** : Object p is density-connected to object q w.r.t ϵ and MinPts if there is an object r such that both p and q are density-reachable from r w.r.t ϵ and MinPts.

Algorithm 1 DBSCAN algorithm

```
clusterIndex = 0
for p in dataset do
    if p has label then
        continue
    end if
    if neighboursCount(p,ε) ≥ MinPts then
        p is a core point
        p.clusterId = clusterIndex
        for neighbour in neighbours(p,ε) do
            if neighbour has no clusterId then
                neighbour.clusterId = clusterIndex
                if neighbour is core point then
                    Visit all neighbours
                end if
            end if
        end for
    else
        p is a noise point
    end if
end for
clusterIndex++
```

Key points

- It is density based and used euclidean distance as distance metric.
- The algorithm is very sensitive to the parameters Epsilon(Eps) and MinPts.
  Choosing MinPts
  Having domain knowledge is important to find MinPts. Also, its obvious that MinPts can't be 1 and should be atleast number of dimensions increased by 1.
  Choosing Epsilon:
  This is calculated using K-distance graph which means plotting the sorted distance between a point and its Kth nearest neighbour for all points in the dataset. The distance in the graph where the maximum curvature occurs is taken as Eps.
- The algorithm is robust to outliers
- Algorithm does not require to specify number of clusters.
- Algorithm may not work in high dimensional data and varying density clusters.

Python Implementation Example

```python
class DBSCAN():
  def __init__(self,min_samples,eps,dataset):
  '''Constructor'''
    self.min_samples = min_samples
    self.eps = eps
    self.dataset = dataset


  #Auxilary functions
  def neighbours(self,pt):
    neighbours= []
    point = self.dataset[pt]

    for y_idx,y_pt in enumerate(self.dataset):
      norm = np.linalg.norm(y_pt - point)
      if norm <= self.eps and y_idx != pt:
        neighbours.append(y_idx)
    return neighbours

  def check_neighbours(self,pt_id,cluster_idx,cluster_to_pt):
    for neighbour in self.neighbours(pt_id):
      if cluster_to_pt[neighbour]==-1:
        cluster_to_pt[neighbour] = cluster_idx
        if len(self.neighbours(neighbour))>=self.min_samples:
          self.check_neighbours(neighbour,cluster_idx,cluster_to_pt)


  #Actual DBSCAN Code
  def clustering(self):
    cluster_idx = 0
    #Initialising cluster indices to -1 for the whole dataset
    cluster_to_pt = [-1]*len(self.dataset)
    for pt_idx,pt in enumerate(self.dataset):
      if cluster_to_pt[pt_idx] != -1:
        continue
      if len(self.neighbours(pt_idx))>=self.min_samples:
        cluster_to_pt[pt_idx] = cluster_idx
        self.check_neighbours(pt_idx,cluster_idx,cluster_to_pt)
      cluster_idx +=1

    return cluster_to_pt
```

Running DBSCAN

```
#Parameters: MinPts:4,Eps: 0.06
dbs = DBSCAN(4,0.06,dataset1)
p = dbs.clustering()
fig, ax = plt.subplots()

scatter = ax.scatter(dataset1[:,0],dataset1[:,1], c=p, cmap='rainbow')
legend1 = ax.legend(*scatter.legend_elements(),
                    loc="lower left", title="Cluster")
ax.add_artist(legend1)

plt.show()
```

Running K-Means using sklearn

```
# FROM ABOVE PLOT WE HAVE OPTIMAL K = 2
kmeans = KMeans(n_clusters = 2,  random_state = 42)
label = kmeans.fit_predict(dataset1)
fig, ax = plt.subplots()

scatter = ax.scatter(dataset1[:,0],dataset1[:,1], c=label, cmap='
    rainbow')
legend1 = ax.legend(*scatter.legend_elements(),
                    loc="lower left", title="Cluster")
ax.add_artist(legend1)

plt.show()
```

Figure 1: DBSCAN Clustering

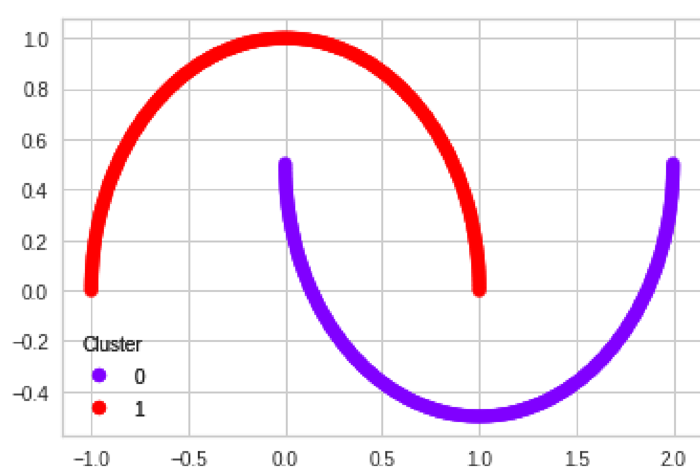
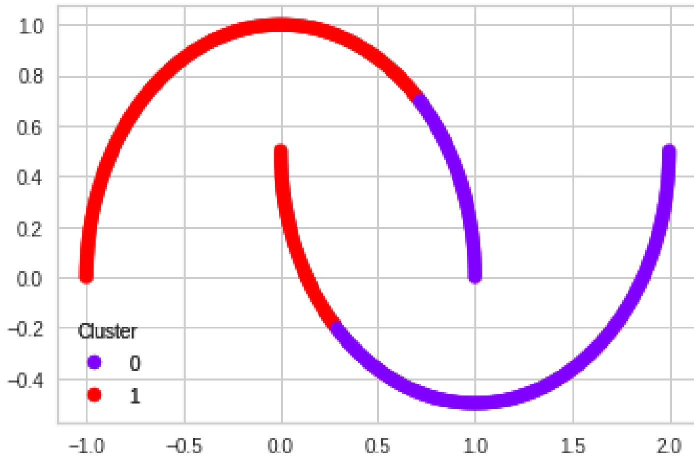


Figure 2: K-means Clustering



Questions

1. When will DBSCAN fail?

   A. For low dimensional data

   B. When density drops between clusters

   Solution:
   B. When density drops between clusters

2. State the two parameters of the model.
   Solution:
   Eps and MinPts

3. Which of the following is false?

   A. It is sensitive to parameters

   B. We need to specify the number of clusters

   Solution:
   B. We need to specify the number of clusters

4. Best case time complexity of the algorithm.
   Solution:
   If we use indexing to store the distances, time complexity turns out $\mathcal{O}$(nlogn)

5. What happens if MinPts is too large?
   Solution:
   Small clusters may combine to give big cluster which ignores the small clusters.