DBScan

2016059198 진승현

1. 프로그램 개요

2-D point 데이터를 입력받아 입력된 매개변수에 맞도록 clustering 된 n개의 2-D point 데이터를 출력하는 프로그램이다.

2. DBScan

DBScan은 밀도 기반으로 clustering을 하는 알고리즘이다.

Parameter는 다음과 같다.

n : n개의 cluster로 분할

Eps : 임의의 점 p로부터 다른 점 q를 이웃(neighborhood)으로 볼 수 있는 최대 거리(반지름)

MinPts : 임의의 점 p가 core point가 될 수 있는 최소 이웃의 수

일 때 모든 점은 다음의 3가지 종류 중 하나가 된다.

Core Point : 임의의 점 p에서부터 Eps 거리 안에 있는 neighborhood의 수가 MinPts보다 같거나 많은 점 p.

Border Point : 임의의 점 q에서부터 Eps 거리 안에 있는 neighborhood의 수가 MinPts보다 적으면서 neighborhood에 Core Point를 갖는 점 q.

Noise Point : 임의의 점 r에서부터 Eps 거리 안에 있는 neighborhood의 수가 MinPts보다 적으면서 neighborhood에 Core Point가 없는 점 r.

여기서 같은 Core Point를 공유하는 Core Point들과 Border Point들은 dencity-connected 되어있다고 하며 이를 하나의 cluster로 볼 수 있다.

이러한 방식으로 cluster를 확장하는 알고리즘이다.

3. 코드 설명(함수 기준)

1) Get Distance & Get Neighbors

```
def getDistances(p1, p2): #p1=point, p2=points
    d_type = ([('id','i4'), ('distance','f8')])
    distances = np.zeros(p2.shape[0], dtype=d_type)
    distances['id'] = p2['id']
    distances['distance'] = np.sqrt(np.power(p2['x'] - p1['x'],2) + np.power(p2['y'] - p1['y'],2))
    return distances


def getNeighbors(p):
    dist = getDistances(p, data[data != p])
    dist = dist[dist['distance'] <= Eps]
    return np.array([data[data['id'] == d['id']] for d in dist])
```

getDistances(p1, p2)

: 임의의 점 p1으로부터 임의의 점들 p2까지의 Euclidean 거리들을 반환하는 함수

getNeighbors(p)

: 임의의 점 p에서 거리가 Eps 안에 있는 neighbor들을 반환하는 함수

2) scan

```
def scan(p, neighbors, index):
    size = 0
    if neighbors.size < MinPts: #noise or border point
        points[points['id'] == p['id']] = (p['id'], index) # noise or border point
        if index == -1: #noise
            return 0
        else: #border
            return 1
    else: #core point
        size += 1
        points[points['id'] == p['id']] = (p['id'], c_index) # set cluster
        for neighbor in neighbors:
            if points[points['id'] == neighbor['id']]['cluster'] < 0: #not visited or noise
                size += scan(neighbor, getNeighbors(neighbor),c_index)
        return size
```

scan(p, neighbors, index)

: 임의의 점 p에서 density-connected인 점들을 재귀적으로 찾아 c_index번째 cluster에 할당하고 그 cluster의 size를 반환하는 함수

이웃의 수가 MinPts보다 작고 index가 c_index가 아니라면(-1이라면) noise point

이웃의 수가 MinPts보다 작지만 index가 c_index라면 border point

이웃의 수가 MinPts보다 크면 core point -> neighbors를 모두 재귀적으로 탐색해준다.

3) DBSCAN

```
def DBSCAN():
    global c_index
    cluster_size = []
    while -2 in points['cluster']: #모든 점을 다 순회했을 때까지
        pid = np.random.choice(points[points['cluster'] == -2]['id'], 1) #not visited points 중에서 하나 선택
        p = data[data['id'] == pid]
        size = scan(p, getNeighbors(p),-1)
        if size != 0:
            cluster_size.append(size)
            print(cluster_size[c_index])
            c_index += 1

    return np.array(cluster_size)
```

DBSCAN

: DBSCAN 본체 함수

모든 point들이 최소 1번 scan 될 때까지 반복(while, -2는 unvisited를 의미)

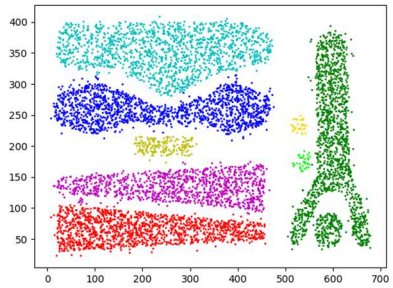
Size가 0이 아닌 경우(noise point가 아닌 경우) 하나의 cluster로 할당함.

4. 실행 예제

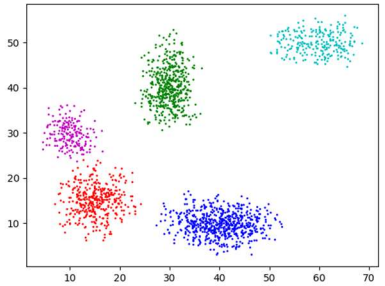
1)Argument Format

Python cluster.py [input data src] [cluster #] [Eps] [MinPts]

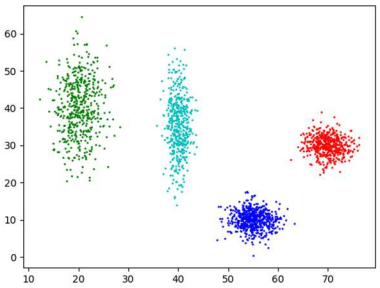
2) test 결과



Input1 : 98.90826점



Input2 : 94.56644점



Input3 : 99.95467점

5. Specification of Testing

OS : Windows 10 Home (x64)

Language version : Python 3.7.0