Assignment-3 (DBSCAN)

2016025105
컴퓨터공학과
강재훈

(i) Summary of your algorithm

DBSCAN(Density Based Spatial Clustering of Applications With Noise)는 density based clustering 입니다. 즉 Eps와 MinPts를 이용해 directly connected한 점들을 찾고 이렇게 찾은 점들을 1개의 cluster로 만듭니다. Eps란 해당 점의 포함 반경을 결정합니다. MinPts는 해당 점이 core point가 되기 위해 포함 반경 안에 포함 해야 하는 최소한의 점의 개수를 의미합니다. DBSCAN 알고리즘은 존재하는 여러 점 중 한개의 점을 선택하고 기존에 정의된 Eps와 MinPts를 이용하여 선택한 점이 core point인지 검사합니다. 만약 core point라면 1개의 cluster를 만들고 그 안에 들어가는 점들 중 core point를 찾아서 위와 같은 방법을 계속 적용하며 directly connected 한 점들을 찾아가며 cluster의 크기를 키워나갑니다. 이후 더이상 core point가 될 수 있는 점이 없고 border point들만 남은 상황이 되면 아직 cluster를 형성하지 못한 점을 선택하여 위와 같은 과정을 반복해 clustering을 완성 시킵니다.
제가 작성한 코드에선 일단 각 obj의 cluster를 초기화 시키고 처음 obj부터 순차적으로 확인하며 아직 cluster가 배정되지 않은 점이 발견되면 clustering을 시작합니다.
유클리디안 거리를 이용해 이웃점을 찾았고 찾은 이웃점들을 queue 자료구조에 넣어 너비우선탐색 알고리즘과 유사하게 구현하였습니다.

.

(ii) Detailed description of your codes

- def find_objs_neighbors(data_set, obj)

Paramger로 들어온 Obj의 neighbors들을 구하는 함수입니다. Distance를 구하고 주어진 Eps 값보다 작다면 neighbor로 취급하여 deque에 넣어줍니다. Queue로 구현한 이유는 이후 DBSCAN 알고리즘이 너비우선탐색과 비슷하게 동작하기 때문입니다.

(데이터 구조를 살펴보니 2차원 데이터여서 DIM_OBJ는 2로 설정하였습니다.)

```
def find_objs_neighbors(data_set, obj):
    global EPS, DIM_OBJ
    ret = deque()
    for other_obj in data_set:
        distance = 0
        for dim in range(1,DIM_OBJ+1,1):
            distance += math.pow(obj[dim]-other_obj[dim],2)
        distance = math.sqrt(distance)
        if distance <= EPS:
            ret.append(other_obj)
    return ret
```

- def DBSCAN(data_set, obj_cluster)

Obj_cluster는 각각의 obj들이 어떤 cluster에 포함되어 있는지 나타내는 리스트입니다. 초기값은 None으로 초기화 되어 있습니다. 따라서 이 리스트를 처음부터 살피며 해당 obj가 core point면 해당 obj부터 시작해서 클러스터를 만들기 시작합니다. Cluster를 확장해 나가는 알고리즘을 너비우선탐색에서 영감을 받아 작성하였습니다.

```
def DBSCAN(data_set, obj_cluster):
    global EPS, MIN_PTS, OUT_LIER, cluster_id

    for obj_idx, obj in enumerate(obj_cluster):
        if obj != None:
            continue
        neighbors = find_objs_neighbors(data_set, data_set[obj_idx])

        if len(neighbors) < MIN_PTS:
            obj_cluster[obj_idx] = OUT_LIER
        else:
            obj_cluster[obj_idx] = cluster_id
            while neighbors:
                now_obj = neighbors.popleft()
                now_neighbors = find_objs_neighbors(data_set, now_obj)
                if len(now_neighbors) >= MIN_PTS:
                    for now_nei in now_neighbors:
                        now_obj_idx = now_nei[0]
                        if obj_cluster[now_obj_idx] == None or obj_cluster[now_obj_idx] == OUT_LIER:
                            neighbors.append(now_nei)
                            obj_cluster[now_obj_idx] = cluster_id

            cluster_id += 1
```

- n_obj_in_clusters, clusters

n_obj_in_cluster == 각각의 clusteer에 몇개의 obj가 들어가는지 가지고 있습니다. 이후 heap 자료구조로 관리해 가장 큰 n개의 cluster만 선택할 예정입니다.

clusters == key 값에 해당 cluster의 id가 value에는 해당 cluster에 속하는 obj의 id가 들어간 리스트가 저장되어 있습니다.

```
n_obj_in_clusters = [[0, i] for i in range(cluster_id)]
clusters = {i:[] for i in range(cluster_id)}

for idx, clu in enumerate(obj_cluster):
    if clu == OUT_LIER or clu == None:
        continue
    n_obj_in_clusters[clu][0] -= 1
    clusters[clu].append(idx)
```

- 결과 파일 작성

n_obj_in_clusters 를 힙으로 만들어 크기가 큰 cluster부터 차례로 결과파일을 작성합니다.

```
heapify(n_obj_in_clusters)
cluster_counter = 0
while n_obj_in_clusters:
    if cluster_counter == N:
        break
    count, clu_idx = heappop(n_obj_in_clusters)
    if count == 0:
        continue
    file_name = input_file.replace(".txt","") + "_cluster_"+str(cluster_counter)+".txt"
    with open(file_name, 'w') as f:
        for obj_idx in clusters[clu_idx]:
            f.write(str(obj_idx)+"\n")
    cluster_counter += 1
```

(iii) Instructions for compiling your source codes at TA’s computer

python version == Python 3.9.5
실행법 == python clustering.py input#.txt N Eps MinPts
테스트실행법 == PA3.exe input#

(iii) Any other specification of your implementation and testing