Assignment3 Report

(DBSCAN)

2016025478 서경욱

1. 개발 환경

- **파이썬 3.8.6**

```
C:₩Users₩user>python --version
Python 3.8.6
```

2. 실행 방법

```
C:\Users\user\Desktop\4-1\데이터사이언스\2021_ite4005_2016025478\Assignment3 (master)
(datascience) λ python clustering.py input1.txt 8 15 22
```

- **python clustering.py (Input data path) (cluster 개수) (Eps) (MinPts)**

3. Summary of Algorithm

본 과제에서는 각 object들의 x,y 좌표들이 주어진 data로 parameter로 받은Eps, MinPts를 이용한 DBSCAN을 통해 density-based spatial clustering을 수행한 뒤 각 cluster별로 parameter에서 받은 cluster 개수만큼의 output file을 cluster별로 index를 분리하여 작성하는 프로그램을 구현했다.

이 프로그램은 input data 경로, 산출할 cluster 개수, Eps, MinPts를 실행 parameter로 받으며 main에서 **parameter가 알맞은 개수**로 들어왔는지 확인한다. 다음으로 input data를 읽어온 뒤 좌표값들을 추출하여 data 리스트에 저장한다. 저장한 data, Eps, MinPts를 인자로 **DBSCAN class**의 object를 생성한다. 다음으로 DBSCAN class의 함수인 get_clusters 함수를 통해 clustering을 수행한다. DBSCAN class 내에는 모든 data에 대한 **visited 배열**이 있어서 clustering이 끝난 item들을 체크한다.

모든 data의 item들을 순회하며 visited 배열을 통해 방문하지 않은 item에 대해서 visited를 check해주고 clustering을 시작한다. 미리 계산한 item간의 dist 행렬을 이용해서 **Eps보다 작거나 같은 거리의 item**들을 neighbor로 뽑아낸다. 이 neighbor에

속한 **item의 개수가 MinPts보다 작으면** noise로 분류하기 위해 DBSCAN object의 label 배열에 -1로 기록해준다. 만약 neighbor의 item의 개수가 MinPts보다 크거나 같다면 현재 item은 **core point condition을 만족하므로 expand하게 된다**.

expand 함수에서는 cluster의 core였던 item의 neighborhood를 초기값으로 받고 이들을 순회하며 labeling이 되어있지 않은 neighbor을 core의 label로 지정해준다. core에서 **density-reachable한 점들을 찾아 cluster를 확장**하기 위해 neighbor들의 neighborhood(거리가 Eps 이하의 item들)을 구하고 이들의 개수가 MinPts보다 크거나 같다면 neighbor 배열에 추가해서 추후의 반복문에서 그 item으로부터 density-reachable한 점들을 찾아나가며 cluster를 확장하게 된다.

만약 더 이상 **neighbor에 core point condition을 만족하는 item이 없다면** 함수를 종료하고 나머지 visit하지 않은 item들에 대해 get_clusters 함수에서 clustering을 진행한다.

4. Detailed description of each function

```
class DBSCAN:
    def __init__(self, data, Eps, MinPts):
        self.data_cnt = len(data)
        self.epsilon = Eps
        self.MinPts = MinPts
        self.visited = np.full((self.data_cnt), False)
        dist = np.zeros((self.data_cnt, self.data_cnt))
        for i in range(self.data_cnt):
            print(str(i) + "  is done !!!")
            for j in range(self.data_cnt):
                dist[i][j] = np.sqrt((data[i][0]-data[j][0])**2 +
                                     (data[i][1]-data[j][1])**2)
        self.dist = dist
        self.label = np.full((self.data_cnt), 0)
        # 현재 cluster index
        self.index = 0
```

- **DBSCAN class**
- data_cnt = data에 들어있는 item의 개수
- epsilon = Eps (core point condition 확인을 위해 설정하는 거리 threshold)
- MinPts = item p로부터 Eps 이하의 거리 안에 있는 item q 개수의 threshold,

MinPts보다 같거나 많은 item이 Eps 거리내(neighbor)에 속하면 해당 item p는 core point이고 cluster를 형성할 수 있다.

- visited = 이미 clustering이 된 item들에 대해서 clustering을 진행하지 않기위해 체크하는 배열
- dist = 거리 행렬, dist[i][j]는 i에서 j까지의 거리(유클리드 거리)
- label = 몇번째 item이 몇번째 cluster에 속하는지 기록(-1은 noise)
- index = cluster index

```
if __name__ == '__main__':
    start = time.time()
    # check correct parameter
    if len(sys.argv) != 5:
        print("Insufficient arguments")
        sys.exit()

    data_path = sys.argv[1]
    input_num = int(data_path[5])
    n_clusters = int(sys.argv[2])
    Eps = float(sys.argv[3])
    MinPts = int(sys.argv[4])

    data_file = open(data_path, 'r')
    lines = data_file.readlines()
    data_file.close()

    data = []
    for line in lines:
        _idx, x, y = line.split('\t')
        data.append([float(x), float(y)])
        print(str(_idx) + "  is done !!!")
```

- main 함수
- sys.argv를 통해 실행 parameter를 받아온다. 만약 parameter 개수를 만족하지 않으면 프로그램을 종료한다.
- data_file을 data_path를 이용해 읽어온 뒤 readlines 함수를 통해 내용을 읽어온다. data라는 리스트에 delimiter '₩t'를 이용해 분리하여 x,y 좌표값을 저장한다. object_id는 input data의 id가 저장하는 순서대로라 따로 저장하지 않고 data의 index를 이용했다.

```
clustering_method = DBSCAN(data, Eps, MinPts)
clustering_method.get_clusters()
labels, counts = np.unique(clustering_method.label, return_counts=True)
label_cnt = sorted(list(zip(labels, counts)),
                   key=lambda x: x[1], reverse=True)
k = 0
for i in range(n_clusters+1):
    if label_cnt[i][0] == -1:
        continue
    else:
        output_file = open("input"+str(input_num) +
                           "_cluster_" + str(k) + ".txt", 'w')
        points = np.where(clustering_method.label == label_cnt[i][0])[0]
        for point in points:
            print(point, file=output_file)
        output_file.close()
        k += 1

print("time: ", time.time()-start)
```

- main 2번째 파트
- DBSCAN class clustering_method object를 생성하고 get_clusters 함수를 통해 data의 clustering을 한 뒤 object의 label배열에 clustering의 결과를 저장한다.
- numpy의 unique함수를 통해 label의 종류와 개수를 받는다.
- zip함수로 종류와 개수를 묶은 뒤 종류의 개수에 따라 내림차순으로 sort한다. **(Optional 구현)**
- 그 후 반복문을 돌며 input의 cluster개수만큼만 cluster의 큰 것부터 output_file로 명세에 맞게 작성하여 생성한다. 단, label이 **-1이라면 noise**이므로 명세에 따라 저장하지 않는다.

```
def get_clusters(self):
    for idx in range(self.data_cnt):
        if not self.visited[idx]:
            self.visited[idx] = True
            neighborhood = np.where(self.dist[idx] <= self.epsilon)[0]
            if len(neighborhood) < self.MinPts:
                self.label[idx] = -1
            else:
                # print(neighborhood)
                self.index += 1
                self.label[idx] = self.index
                self.expand(neighborhood.tolist())
```

- DBSCAN Clustering을 진행하는 get_clusters 함수이다
- 모든 data에 대해서 방문하지 않은 item에 대해 visited를 check하고 Eps를 이용해 거리 내의 neighbor들을 구한다.
- neighborhood의 item의 개수가 MinPts보다 작다면 noise로 간주하고 label을 -1로 설정한다.
- neigborhood의 item개수가 MinPts보다 크거나 같다면 index(cluster index)를 1 늘려 core의 label을 index로 설정해주고 neighborhood를 expand의 인자로 넘기며 cluster를 확장한다.

```
def expand(self, neighborhood):
    for n in neighborhood:
        if self.label[n] == 0:
            self.label[n] = self.index

        if not self.visited[n]:
            self.visited[n] = True
            cur_neighborhood = np.where(
                self.dist[n] <= self.epsilon)[0].tolist()
            if len(cur_neighborhood) >= self.MinPts:
                neighborhood.extend(cur_neighborhood)
```

- p가 core point일 때 p로부터 density-reachable한 item들을 찾아 cluster를 확장하기 위한 함수이다
- core point의 각 neighbor에 대해 label이 0(초기값)일 때, neighbor들은 p로부터 **directly density-reachable**하므로 core point의 label로 설정해준다.
- 방문하지 않은 neighbor에 대해 visited 배열에 check해주고 해당 neighbor에서의 거리가 Eps내의 item들을 뽑아 이들의 개수가 MinPts보다 크다면 core point가 될 수 있고 **그 item으로부터 더 확장해나가기 위해** neighbor에 extend함수로 추가한다.
- extend함수는 numpy array에서 작동할 수 없으므로 tolist함수를 통해 neighborhood, cur_neighborhood 함수 모두 list형태로 변환해서 extend를 통해 neighbor에 cluster에 속하는지, density-reachable한지 확인해야할 item들을 계속해서 늘려주었다.

5. 실행 결과

```
7992  is done !!!
7993  is done !!!
7994  is done !!!
7995  is done !!!
7996  is done !!!
7997  is done !!!
7998  is done !!!
7999  is done !!!
time:  139.03185772895813
```

(input1.txt time)

```
C:\Users\user\Desktop\4-1\데이터사이언스\2021_ite4005_2016025478\Assignment3 (master)
(datascience) λ PA3.exe input1
98.97277점
```

```
1992  is done !!!
1993  is done !!!
1994  is done !!!
1995  is done !!!
1996  is done !!!
1997  is done !!!
1998  is done !!!
1999  is done !!!
time:  11.40188455581665
```

(input2.txt time)

```
C:\Users\user\Desktop\4-1\데이터사이언스\2021_ite4005_2016025478\Assignment3 (master)
(datascience) λ PA3.exe input2
94.86598점
```

```
2092  is done !!!
2093  is done !!!
2094  is done !!!
2095  is done !!!
2096  is done !!!
2097  is done !!!
2098  is done !!!
2099  is done !!!
time:  12.752061367034912
```

(input3.txt time)

```
C:\Users\user\Desktop\4-1\데이터사이언스\2021_ite4005_2016025478\Assignment3 (master)
(datascience) λ PA3.exe input3
99.97736점
```