project 3 report

2017030464 한현진

OS : Window 10 **언어버전** : Python 3.10.1

IDE : Pycharm Community Edition 2021.2.2

ü Summary of your algorithm

DBSCAN의 clustering 알고리즘을 사용했습니다.

1. 데이터 상에 존재하는 아무 점 하나(P)를 랜덤하게 선택한다. (나의 코드의 경우엔 그냥 첫번째부터 끝까지 선택한다.)

2. 그 점의 Eps 거리 이내에 존재하는 점의 개수를 다 찾아본다.

3-1. 그 점의 개수가 MinPts 개수와 같거나 더 많다면 선택했던 점 P를 core point로 정하고 그 반경안에 들어온 점들과 cluster를 형성한다.

3-2. 만약 MinPts 개수보다 적다면 클러스터를 형성하지 못하므로 다른 랜덤한 점 P를 다시 뽑는다.

4. 만약 cluster가 형성되었다면 형성된 cluster 안에 점들을 확인하면서 그 점들이 또다른 core point가 되는지 확인하고 core point가 된다면 기존에 자신이 속해있던 cluster와 자신이 core point인 cluster를 합치고 그렇게 cluster의 크기를 키워간다.

5. 만들어진 cluster의 모든 점을 확인하고 더 이상 클러스터를 늘릴 수 없다고 판단하면, 나머지 데이터에서 또 랜덤하게 한 점 P를 선택하고 이 과정을 반복한다.

ü Instructions for compiling your source codes at TA's computer (e.g. screenshot) (Important!!)

```
C:₩workspace₩2022_ite4005_2017030464₩DBSCAN>python clustering.py input1.txt 8 15 22
C:₩workspace₩2022_ite4005_2017030464₩DBSCAN>python clustering.py input2.txt 5 2 7
C:₩workspace₩2022_ite4005_2017030464₩DBSCAN>python clustering.py input3.txt 4 5 5
```

파이썬 파일이라서, 실행폴더까지 들어가서
python clustering.py input파일 n Eps MinPts
로 명령어를 치시면 될 것 같습니다.

ü Any other specification of your implementation and testing

인풋파일은 DBSCAN 폴더안의 data_input 폴더안에 넣어주시고,
실행을 마치면, 아웃풋파일은 DBSCAN 폴더안의 data_output 폴더안에 생길것입니다.
그럼 data_input, data_output, data_test 폴더안의 파일을 필요에 맞게 복사해서
test 폴더안의 PA3.exe 파일로 테스트를 진행해보시면

```
C:₩workspace₩2022_ite4005_2017030464₩DBSCAN₩test>PA3.exe input1
98.97037점
C:₩workspace₩2022_ite4005_2017030464₩DBSCAN₩test>PA3.exe input2
94.86598점
C:₩workspace₩2022_ite4005_2017030464₩DBSCAN₩test>PA3.exe input3
99.97736점
```

위와 같은 결과를 얻을 수 있습니다.

ü Detailed description of your codes (for each function)

코드의 흐름대로 설명 드리도록 하겠습니다.

```
def main():
    # 입력 받아오는 부분
    args = sys.argv[1:]
    input_name = args[0]
    n, Eps, MinPts = map(int, args[1:])

    data = pd.read_csv("./data_input/" + input_name, delimiter='\t', names=['object_id', 'x_coordinate', 'y_coordinate'])

    index = 0
    cluster_candidates = []
    while index < len(data):
        if len(data) < MinPts:
            break

        select_data_x = data.iloc[index, 1]
        select_data_y = data.iloc[index, 2]

        cluster_candidate = data.loc[((data['x_coordinate'] - select_data_x) ** 2) +
                                     ((data['y_coordinate'] - select_data_y) ** 2) <= Eps ** 2, :]

        if len(cluster_candidate) >= MinPts:
            index = 0
            data = data.loc[((data['x_coordinate'] - select_data_x) ** 2) +
                            ((data['y_coordinate'] - select_data_y) ** 2) > Eps ** 2, :]
            cluster_candidate, data = clustering(Eps, MinPts, data, cluster_candidate)
            cluster_candidates.append(cluster_candidate)
        else:
            index += 1
```

우선, 입력을 받습니다.

받은 입력을 토대로, input#.txt의 데이터를 pandas로 불러와서 data 변수에 저장합니다.
n, Eps, MinPts도 각각 저장합니다.

그리고 while 문으로, data안의 point들을 끝까지 확인을 했을 때, 와일문을 빠져나옵니다.

근데 와일문 안에서0 애초에 data에 남은 point의 개수가 MinPts 보다 작다면 클러스터를 형성하지 못하므로, 그냥 if문의 break로 빠져나옵니다.

만약 if문에서 걸러지지 않았다면, index 번째의 data 행에서 x와 y좌표 값을 알아낸 뒤(랜덤하게 point 선택), data 안에 있는 모든 point의 x와 y좌표와 계산해서, Eps와 비교하여 그 좌표간의 거리가 Eps보다 작거나 같다면, 같은 클러스터로 묶을 수 있습니다.

이 때, 형성된 cluster 후보의 point 개수가 MinPts 개수보다 크거나 같다면, 선택된 point가 core point가 되었다고 생각하고, data에서 클러스터가 형성된 point들을 모두 삭제해줍니다. 그리고 clustering 함수에 Eps, MinPts, data, cluster_candidate를 매개변수로 넘겨주고 해당 함수의 결과로 나온 것을 cluster로 확정된 것이라 생각하고 cluster_candidates 리스트에 넣어줍니다. 그리고 이 때에는, data가 변환되었으므로, index 값을 0으로 돌려서 다시 와일문을 돌아줍니다.

만약, cluster 후보의 point 개수가 MinPts 보다 작다면, 해당 값은 border 포인트가 될 값이거나, 아니면 outlier 이므로, 그냥 두고 index를 1증가시키고 while문을 마저 돌아줍니다.

```
def clustering(Eps, MinPts, data, cluster):
    index = 0
    while index < len(cluster):
        select_data_x = cluster.iloc[index, 1]
        select_data_y = cluster.iloc[index, 2]
        in_data_candidate = data.loc[((data['x_coordinate'] - select_data_x) ** 2) +
                                     ((data['y_coordinate'] - select_data_y) ** 2) <= Eps ** 2, :]
        in_cluster_candidate = cluster.loc[((cluster['x_coordinate'] - select_data_x) ** 2) +
                                           ((cluster['y_coordinate'] - select_data_y) ** 2) <= Eps ** 2, :]
        if len(in_data_candidate) != 0:
            cluster_candidate = pd.merge(in_data_candidate, in_cluster_candidate, how='outer',
                                         on=['object_id', 'x_coordinate', 'y_coordinate'])

            if len(cluster_candidate) >= MinPts:
                data = data.loc[((data['x_coordinate'] - select_data_x) ** 2) +
                                ((data['y_coordinate'] - select_data_y) ** 2) > Eps ** 2, :]
                cluster = pd.merge(cluster, in_data_candidate, how='outer',
                                   on=['object_id', 'x_coordinate', 'y_coordinate'])
        index += 1

    return cluster, data
```

Clustering 함수는 우선 실행이 된다면, 무조건 cluster가 형성 되었다는 가정하에 실행됩니다.

함수가 한번 실행되면 그냥 무조건 끝까지 cluster를 확인해서 계속 범위를 늘려나가는 것입니다.

Cluster에 있는 point를 순서대로 선택해서 data와 cluster에서 Eps 범위 안에 드는 point가 MinPts개수를 넘어가면 그 또한 core point가 될 수 있는데, 이미 형성된 cluster가 있으므로 그 cluster와 합쳐서 생각해줍니다. Pandas의 merge를 이용해서 말입니다. 그럼 또 그 만큼 data에서 해당 merge된 point들은 없애고 생각해줍니다.

그렇게 그냥 끝까지 확인을 해주고 cluster와 data를 리턴해줍니다.

```
cluster_candidates.sort(key=len, reverse=True)
while len(cluster_candidates) != n:
    cluster_candidates.pop()

index = 0
for cluster in cluster_candidates:
    output = open('./data_output/' + input_name[:6] + "_cluster_" + str(index) + input_name[6:], 'w', encoding='utf-8')
    for i in range(len(cluster)):
        output.write(str(cluster.iloc[i, 0]) + '\n')
    output.close()
    index += 1
```

다시 main 함수로 돌아와서, 그렇게 와일문을 다 돌아서 cluster 후보군을 다 정했다면, cluster들이 들어있는 cluster_candidates 리스트를 각 클러스트의 크기 내림차순으로 정렬해서, n의 개수와 같을 때까지 클러스터를 삭제해줍니다.

그리고 선택된 클러스터들을 이용해서 명세서에서 요구한 대로 output 파일을 만들어줍니다.