**2018008304 컴퓨터소프트웨어학부 박경하**

**Summary of your algorithm**

주어진 input.txt 을 pandas에 담은 뒤, 추가로 visited과 noise 열을 만들어주었습니다. Visited은 시간복잡도를 줄이기 위해 이미 방문한 데이터라면 재방문하지 않도록 하기 위한 열이며, noise는 noise로 판명된 데이터를 따로 분리하기 위한 열입니다.

Build\_dbscan() 함수에서, 데이터의 1행부터 차례대로 for문을 통해 corepoint인지 측정합니다. Corepoint을 계산하기 위해 각각의 데이터들을 차례대로 find\_neighbors() 함수에 넣어주며, is\_corepoint() 을 통해 corepoint인 것이 판명되면 해당 corepoint의 이웃들이 담긴 dataset을 clustering\_neighbors() 함수에 넘겨 이웃들의 data들도 corepoint인지 계산합니다. 만약 어떤 이웃 또한 corepoint임이 판명되면 재귀함수를 통해 확장해나가며 cluster을 확정시켜줍니다.

해당 과정을 모든 데이터셋에 적용한 뒤, write\_dataset()을 통해 output 형식에 맞게 file을 작성합니다.

**Detailed description of your codes (for each function)**

**- def load\_dataset():**

input.txt을 pandas에 할당해주는 함수입니다. Visited는 해당 데이터에 접근했는지를 나타내고, noise는 해당 데이터가 노이즈로 판명되었는지 나타내며, inx\_of\_cluster는 배정된 클러스터의 index을 나타냅니다.

**- def build\_dbscan():**

pandas에 담은 데이터셋을 1행부터 차례대로 corepoint인지 판단해주고 클러스터를 배정해주는 함수입니다. 각각의 데이터가 corepoint인지 판단하기 위해 find\_neighbors() 함수에 인자로 넣어주며, 만약 corepoin라면 이를 중심으로 클러스터를 배정해줍니다. 만약 코어포인트가 아니라면 noise라고 분류합니다.

**- def find\_neighbors(target\_index, target\_row):**

인자로 받은 행이 corepoint인지 계산하는 함수입니다. 각각의 x좌표와 y좌표를 통해 입실론 반경 내의 데이터 개수를 세어준 뒤, is\_corepoint(x) 함수를 통해 코어포인트인지 확정해줍니다. 만약, 해당 data가 corepoint라면 해당 데이터의 입실론 내에 있었던 이웃들의 데이터를 clustering\_neighbors(neighbors) 함수의 인자로 넣고, 이웃들의 데이터도 corepoint인지 계산합니다.

또한, 재방문을 막기 위해 visited은 true로 바꾸어줍니다.

**- def is\_corepoint(x):**

입실론 반경 내에 있는 데이터의 개수가 minpts 보다 크거나 같다면 corepoint 라고 리턴해주는 함수입니다.

**- def clustering\_neighbors(neighbors):**

find\_neighbors에서 코어포인트로 판명된 데이터의 이웃데이터도 코어포인트인지 계산해주기 위한 함수입니다. 만약 이웃의 어떤 데이터 또한 corepoint로 판정되었다면 이 재귀함수를 통해 corepoint가 아닌 데이터가 나올 때까지 반복하며 클러스터를 배정해주는 함수입니다.

**- def write\_dataset():**

클러스터를 배정한 pandas을 output 형식에 맞는 txt파일로 작성하기 위한 함수입니다.

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



맥북의 terminal 에서 위와 같이 실행해주었습니다.

**Any other specification of your implementation and testing**

구동완료 되는 데까지 시간이 다소 소요됩니다ㅠㅠ!!