**Programming Assignment #3 report**

한양대학교 컴퓨터전공

2012004087

이기준

**Summary of your algorithm**

1. 임의의 점 p를 선정한다. P의 eps 범위안의 있는 점들의 개수를 찾는다.
2. eps 범위안의 점의 개수가 minPts 이상이라면 core point로 판단한다.
3. Core point를 중심으로 cluster를 계속 확장해 나간다.
4. Core point의 eps 범위안의 모든 점에 대해 core인지 확인을 하는 작업을 동일하게 한다.
5. core라면 cluster 에 eps 범위안의 점을 추가하며, border이면 추가하지 않는다.
6. 위 과정을 반복하여 Cluster 개수가 사용자에게 입력받은 n값에 도달하면 프로그램을 종료한다.

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

**import** numpy **as** np  
**import** math  
**import** random  
**import** sys

넘파이를 통해 데이터처리를 쉽게 하였다.

Sqrt, pow 함수를 사용하기 위한 math

임의의 point p 를 찾기 위한 random

사용자 입력 input file , n, eps, minPts 를 받기위한 sys

numOfClassified = 0 현재 몇 개의 점들이 classified 되었는지 알려주는 전역변수

x축을 첫번째, y축을 두번째 순서로 정렬한다.

**def** SortDB (fp) :

---- 파일입력 부분 생략-----  
  
 list.sort(key = **lambda** x: (x[1], x[2]))  
 **return** list

근처에 있는 point를 찾기 쉽게 하기 위한 작업이다.

p의 index에서 점점 위로 올라가서 탐색

**def** IsCorePoint (DB, p\_idx, eps, minPts, cluster\_idx) :  
 **global** numOfClassified  
 neighbors = []  
 count = 0  
target\_idx = np.where(np.array(DB)[:, 0] == p\_idx)[0][0]  
 target = DB[target\_idx]  
idx = target\_idx  
 **while** DB[target\_idx][1] - eps <= DB[idx][1] :  
 idx -= 1  
 **if** idx < 0:  
 **break  
 if** math.sqrt(math.pow(DB[idx][1] - DB[target\_idx][1], 2) + math.pow(DB[idx][2] - DB[target\_idx][2], 2)) <= eps:  
 count+=1  
 **if** DB[idx][3] == **None** :  
 neighbors.append(DB[idx][0])  
  
idx = target\_idx  
 **while** DB[idx][1] <= DB[target\_idx][1] + eps:  
 idx += 1  
 **if** idx == len(DB):  
 **break  
 if** math.sqrt(math.pow(DB[idx][1] - DB[target\_idx][1], 2) + math.pow(DB[idx][2] - DB[target\_idx][2],2)) <= eps:  
 count += 1  
 **if** DB[idx][3] == **None** :  
 neighbors.append(DB[idx][0])  
  
**if** minPts - 1 <= count :  
 **if** cluster\_idx == -1 :  
 neighbors.append(DB[target\_idx][0]) **return** [**"core"**, neighbors]  
 **elif** count == 0:  
 numOfClassified += 1  
 target[3] = **"outlier"  
 return** [**"outlier"**]  
 **else** :  
 **return** [**"border"**, neighbors]

또 p의 index에서 점점 아래로 탐색

탐색 범위는 x - eps<=x <= x+ eps && y - eps <= y <= y +eps

Eps 범위에 해당하려면 유클리드 거리가 eps 이내여야 한다.

eps 범위에 해당되는 점 들을 리스트로 저장

eps 범위에 해당되는 점 들의 개수를 count

minPts 보다 count 값이 크면 core라고 반환하며

minPts 보다 count값이 작으면 border라고 반환

또한 count값이 0이면 outlier라고 반환

시작점 p 가 core라고 판단 났을 떄 확장하는 함수

**def** ExpandCluster(DB, cluster, eps, minPts, cluster\_idx) :  
  
 new\_cluster = []  
 new\_cluster += cluster  
  
 **while** len(cluster) != 0:  
 *# cluster에 담은 point들이 core인지 border인지 IsCorePoint로 체크* tempList = IsCorePoint(DB, cluster.pop(), eps, minPts, cluster\_idx)  
 *# print(cluster)  
 # core면 core리스트를 추가해줘야 하는데 original index를 통해 new\_cluster와 중복체크를 함* **if** tempList[0] == **"core"** :  
 **for** item **in** tempList[1] :  
 **if** item **not in** new\_cluster:  
 new\_cluster.append(item)  
 cluster.append(item)  
 *# stack이 비면 종료* **return** new\_cluster

점 p를 포함한 neighbors가 들어 있는 list자료형 cluster 변수를 stack 자료구조처럼 이용한다.

가장 최근에 들어온 점을 pop 하여 core인지 아닌지 확인한다.

core라고 판단 되었을 시 cluster 변수안에 neighbors들을 추가한다. 이 때 자기자신은 추가하지 않는다.

cluster변수안에 어느 원소도 남지 않았을 때 함수를 종료하며 이 과정에서 기록된 모든 클러스터를 담은 리스트 new\_cluster를 반환한다.

**def** DBSCAN (DB, numOfCluster, eps, minPts) :  
 **global** numOfClassified  
*# 총 클러스터 갯수* m = 0  
*# SortDB (DB)* cluster = []  
 **while** numOfClassified != len(DB) **and** m < numOfCluster :  
p\_idx = random.randint(1, len(DB) -1)  
 tempList = IsCorePoint(DB, p\_idx, eps, minPts, -1)  
**if** tempList[0] == **"core"** :  
 cluster.append([])  
cluster[m] =ExpandCluster(DB, tempList[1], eps, minPts, m)  
 **else** :  
 **continue** m += 1  
  
 **return** cluster

1. 임의의 점 p를 선정한다. P의 eps 범위안의 있는 점들의 개수를 찾는다.
2. eps 범위안의 점의 개수가 minPts 이상이라면 core point로 판단한다.
3. Core point를 중심으로 cluster를 계속 확장해 나간다.
4. Core point의 eps 범위안의 모든 점에 대해 core인지 확인을 하는 작업을 동일하게 한다.
5. core라면 cluster 에 eps 범위안의 점을 추가하며, border이면 추가하지 않는다.
6. 위 과정을 반복하여 Cluster 개수가 사용자에게 입력받은 n값에 도달하면 프로그램을 종료한다.

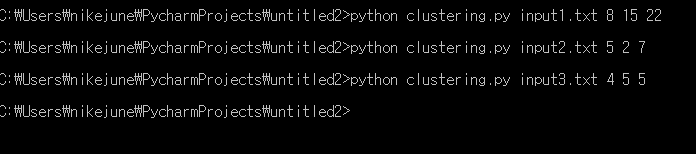
**Instructions for compiling your source codes at TA's computer (e.g. screenshot)**

OS : Windows10, 개발환경: pycharm 2017.2 사용언어:python (ver 3.6.4)

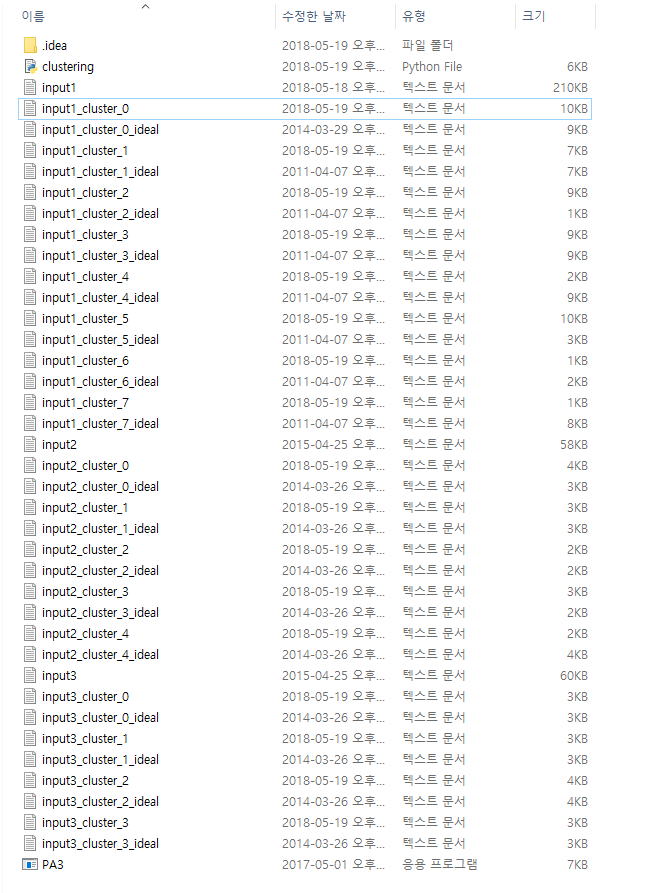
Numpy를 사용하기 때문에 numpy가 설치되어 있지않으면

Pip3 install numpy

를 통해 설치를 해야한다.



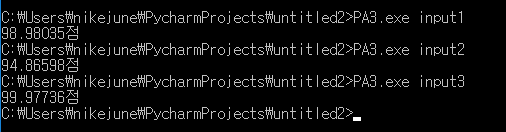
Python clustering.py [input\_file] [n] [eps] [minPts] 순으로 입력



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

**테스트결과**.

(윈도우10에서 실행)



Input1.txt같은 경우는 clustering 과정도 시간이 오래 걸리고 test도 다소 시간이 걸린다.

98.98점으로 거의 99점에 근사하다.

Input2.txt 는 95점에 근사한 94.86점이다.

input3.txt는 100점에 근사한 99.97점이다.