

# Assignment3

Data Science

컴퓨터소프트웨어학부 2017029589 류지범

## #0 Environment


- Python 3.7.11 에서 진행했다.
- 사용한 모듈은 `math` 이다.
- 실행 환경은 macOS Monterey 12.3 이다.
- 패키지 목록은 `requirements.txt` 에 담아두었고 `pip install -r requirements.txt` 로 설치하면 된다.

## #1 Run

- 패키지는 `pip install -r requirements.txt` 로 설치하면 된다.

◦ 

- 소스 코드와 input 파일이 있는 폴더에서 `python clustering.py input#.txt n Eps MinPts` 의 형태로 실행하면 된다.

◦ 

## #2 Code & Function Description

- global variable : DBscan 알고리즘을 돌리기 위해 선언한 전역변수들이다.

```
Eps = None
MinPts = None
n = None
DataSet = list()
isProcessed = None
isCore = None
DDRlist = list()
CoreList = list()
Cluster = list()
ClusterList = list()
```

- `check_core(idx)` : 해당 idx의 data가 core인지 아닌지 판별해주는 함수이다. core일 경우 Directly density-reachable를 list로 만들어 따로 관리하도록 한다.

```
def check_core(idx):
    global isCore, CoreList
    ddr = list()
    cnt = 0
    x = DataSet[idx][1]
    y = DataSet[idx][2]

    for data in DataSet:
        if data[0] == idx:
            continue

        dist = math.sqrt((x - data[1]) * (x - data[1]) + (y - data[2]) * (y - data[2]))
        if dist <= Eps:
            cnt += 1
            ddr.append(data[0])

    if cnt >= MinPts:
        isCore[idx] = True
        CoreList.append(idx)
        return ddr
    else:
        return list()
```

- `init_core()` : 모든 data에 대해 core임을 판별해주고 각 data마다 Directly density-reachable List를 만들어준다 (core일 경우만 존재함).

```
def init_core():
    global DDRlist
    for i in range(len(DataSet)):
        DDRlist.append(check_core(i))
```

- `retrieve_density_reachable(p)` : 재귀적으로 density-reachable한 data를 찾는 함수이다. 현재 보고 있는 data가 core일 경우 모든 directly density-reachable data를 재귀적으로 탐색하면서 core이면 그 과정을 반복하도록 했다. 이 과정에서 발견되는 모든 data들은 한 cluster에 속하도록 `Cluster`에 추가시켰다.

```
def retrieve_density_reachable(p):
    global isProcessed, Cluster
    if isCore[p]:
        if not isProcessed[p]:
            isProcessed[p] = True
            Cluster.append(p)
            for ddr in DDRlist[p]:
                if not isProcessed[ddr]:
                    Cluster.append(ddr)
                    retrieve_density_reachable(ddr)
    else:
        return
```

- `DBscan()` : 실제 DBscan을 실행하는 함수로 core list를 돌면서 density-reachable한 모든 data를 한 클러스터로 만들고 `ClusterList`에 추가시키는 함수이다.

```
def DBscan():
    global Cluster, ClusterList

    for p in CoreList:
        Cluster = list()
        retrieve_density_reachable(p)
        if (len(Cluster)) > 0:
            ClusterList.append(Cluster)
```

- `sort_cluster()` : n개가 넘는 cluster가 존재할 경우 내림차순으로 정렬해서 n개를 끊어야 하므로, 가장 큰 cluster부터 내림차순으로 index를 반환하도록 하는 함수이다.

```
def sort_cluster():
    idxArray = list()
    for i in range(len(ClusterList)):
        idxArray.append((len(ClusterList[i]), i))

    idxArray.sort(key=lambda x: -x[0])

    return idxArray
```

- `read_input(name)` : input.txt로부터 data를 읽어들이는 함수이다.

```
def read_input(name):
    global DataSet, isProcessed, isCore
    with open(name, "r") as f:
        while True:
            line = f.readline()
            if not line:
                break
            temp = line.split()
            data = [int(temp[0]), float(temp[1]), float(temp[2])]
            DataSet.append(data)
    isProcessed = [False] * len(DataSet)
    isCore = [False] * len(DataSet)
```

- `write_output(input_num)` : 양식에 맞게 output file을 작성하는 함수이다.

```
def write_output(input_num):
    fileName = "input" + input_num + "_cluster_"
    outArr = sort_cluster()

    for i in range(n):
        out = open(fileName + str(i) + ".txt", "w")
        idx = outArr[i][1]
        for data in ClusterList[idx]:
            line = str(data) + '\n'
            out.write(line)
        out.close()
```

- main : input file을 읽고 DBscan을 진행하고 output file을 작성한다.

```
if __name__ == '__main__':
    input = sys.argv[1]
    n = int(sys.argv[2])
    Eps = int(sys.argv[3])
    MinPts = int(sys.argv[4])

    input_num = input[5]

    read_input(input)
    init_core()
    DBscan()
    write_output(input_num)
```

## #3 Result

- 3개의 파일을 실행하고 채점한 결과는 다음과 같다.

```
Apple > ~/Des/H/2022/Da/Assignment3 · DataScience < at 05:01:28
> python clustering.py input1.txt 8 15 22
Apple > ~/Des/H/2022/Da/Assignment3
> python clustering.py input2.txt 5 2 7
Apple > ~/Des/H/2022/Da/Assignment3 · DataScience < at 05:02:50
> python clustering.py input3.txt 4 5 5
Apple > ~/Des/H/2022/Da/Assignment3 · DataScience < at 05:03:01
> mono PA3.exe input1
98.90825점 %
Apple > ~/Des/H/2022/Da/Assignment3 · DataScience < at 05:03:14
> mono PA3.exe input2
94.62901점 %
Apple > ~/Des/H/2022/Da/Assignment3 · DataScience < at 05:03:17
> mono PA3.exe input3
99.97736점 %
```

- input1.txt의 경우 한 cluster가 절반의 크기를 차지하고 있어, 실행시간이 약 33s 정도 걸렸다.
- input2.txt와 input3.txt는 1초 미만의 짧은 수행시간을 보여주었다.
- 목표한 결과와 거의 일치하는 모습을 보여주고 있다.