Assignment #3

DBSCAN

한양대학교 컴퓨터소프트웨어학부 2015004911 임성준
데이터사이언스(ITE4005, 12877) - 김상욱 교수님

---

1. 개요

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) 이란 **밀도 기반 클러스터링**으로 "동일한 클래스에 속하는 데이터는 서로 근접하게 분포할 것이다"라는 가정을 기반으로 동작하는 클러스터링 기법이다.
주어진 데이터를 DBSCAN을 활용하여 분류한다.

2. 목적

이 프로젝트의 목적은 주어진 data set을 DBSCAN을 이용하여 적절하게 분류하는 것이다.

3. 개발환경

```
OS : macOS Big Sur 11.3
Language : Python 3.9.1
Source code editor : Visual Studio Code
```

4. 프로그램 구조 및 알고리즘

1. input file, n, eps, minPts를 입력한다.
2. input file의 data set을 저장한다.
3. data set의 point를 확인하며 neighbor과의 길이가 eps를 만족하는지 확인한다.
4. Neighbor이 eps를 만족하면, 현재 cluster에 넣고, 다음 point로 expand한다.
5. 만약 point가 noise라면, 현재 cluster에 넣는다.
6. Expand가 완료되면, 분류되지 않은 point를 찾고 다음 cluster로 설정한다.
7. 모든 point의 분류가 완료되면 종료한다.

5. 코드 설명

```python
'''
Point class
Data set의 point를 선언한다.
'''
class Point:
    def __init__(self, id, x, y):
        self.id = int(id)
        self.x = float(x)
        self.y = float(y)
```

```
        ''' label
        ## Not classified: None
        ## Noise: 0
        ## Cluster id: 1 ~ n
        '''
        self.label = None
```

- `class Point` : *id, x, y, label*, 네 개의 변수를 가지고 있다.
- `__init__` : Point를 초기화하여 생성하는 함수이다.

```
'''
DBSCAN class
DBSCAN을 구현한다.
'''
class DBSCAN:
    def __init__(self, data, n, eps, minPts):
        self.data = data
        self.n = n
        self.eps = eps
        self.minPts = minPts
```

- `class DBSCAN` : *data, n, eps, minPts*, 네 개의 변수를 가지고 있다.
- `__init__` : DBSCAN을 구현하기 위한 변수를 선언한다.

```
    '''
    Get neighbors of point
    주어진 point의 eps를 만족하는 neighbors를 구한다.
    '''
    def get_neighbors(self, point):
        return [p for p in self.data if p != point and get_distance(point, p) <=
self.eps]

    '''
    Expand cluster
    Candidate point를 clustering한다.
    '''
    def expand_cluster(self, neighbors, cluster_id):
        for point in neighbors:
            if point.label == 0: ## Point is noise
                point.label = cluster_id

            if point.label is None: ## Point is unclassified
                point.label = cluster_id
                next_neighbors = self.get_neighbors(point)
                if len(next_neighbors) >= self.minPts:
                    neighbors.extend(next_neighbors)
```

- `get_neighbors` : 주어진 point에서의 근처의 모든 point를 순회하며, distance가 *eps*를 만족하는 point를 return한다.
- `expand_cluster` : 만약 point가 *noise*라면, 현재의 *cluster id*를 입력한다. 또한 만약 point가 *classified 되지 않았다*면,
  현재 cluster id를 입력하고, 해당 point에서의 neighbor을 구한 후, *expand*한다.
  DFS로 동작한다.

```
    '''
    Clustering
    DBSCAN의 clustering을 진행한다.
    '''
    def clustering(self):
        cluster_id = 1

        for point in self.data:
            if point.label is not None: ## Point is classified
                continue

            ## Get neighbors of point
            neighbors = self.get_neighbors(point)

            if len(neighbors) < self.minPts:
                point.label = 0 ## Set point is Noise
                continue

            ## Point is core point
            point.label = cluster_id
            ## Expand cluster
            self.expand_cluster(neighbors, cluster_id)
            cluster_id += 1

        ## Make cluster list from points
        clusters = [[] for _ in range(0, cluster_id-1)]
        for point in self.data:
            if point.label == 0: # Noise
                continue

            clusters[point.label - 1].append(point.id)

        ## Sort clusters
        clusters.sort(key=len, reverse=True)

        ## Select n clusters
        clusters = clusters[:self.n]
```

```
        return clusters
```

- `clustering` : 주어진 data set을 clustering 한다.
    - 전체 data set을 순회하며 point를 확인한다.
    - 해당 point와 *minPts*를 만족하는 neighbor을 찾는다.
    - *neighbor*을 현재 cluster에 넣고, 다음 point를 찾기 위해 *expand*한다.
    - 모든 point의 clustering이 완료되면, cluster list를 return 한 후, 종료한다.

```
'''
Get euclidean distance
Point a와 Point b 사이의 거리를 구한다.
'''
def get_distance(a: Point, b: Point) -> float:
    return math.sqrt(math.pow(a.x - b.x, 2) + math.pow(a.y - b.y, 2))
```

- `get_distance` : a와 b 사이의 거리를 *euclidean distance*로 구한다.

```
if __name__ == "__main__":
    ## Read argv
    input_file = sys.argv[1]
    n = int(sys.argv[2])        ## Number of clusters
    eps = float(sys.argv[3])    ## Epsilon of DBSCAN
    minPts = float(sys.argv[4]) ## MinPts of DBSCAN

    ## Save file name
    input_file_name = input_file.split(".")[0]

    ## Read file
    data = []
    f = open("data-3/"+input_file, "r")
    for line in f.readlines():
        p = line.split()
        point = Point(p[0], p[1], p[2])
        data.append(point)

    ## Run DBSCAN
    dbscan = DBSCAN(data, n, eps, minPts)

    ## Write file
    for idx, output in enumerate(dbscan.clustering()):
        output_file_name = input_file_name + "_cluster_%d.txt" % idx

        with open("test-3/"+output_file_name, "w") as output_file:
            for i in output:
                output_file.write("%d\n" % i)
```

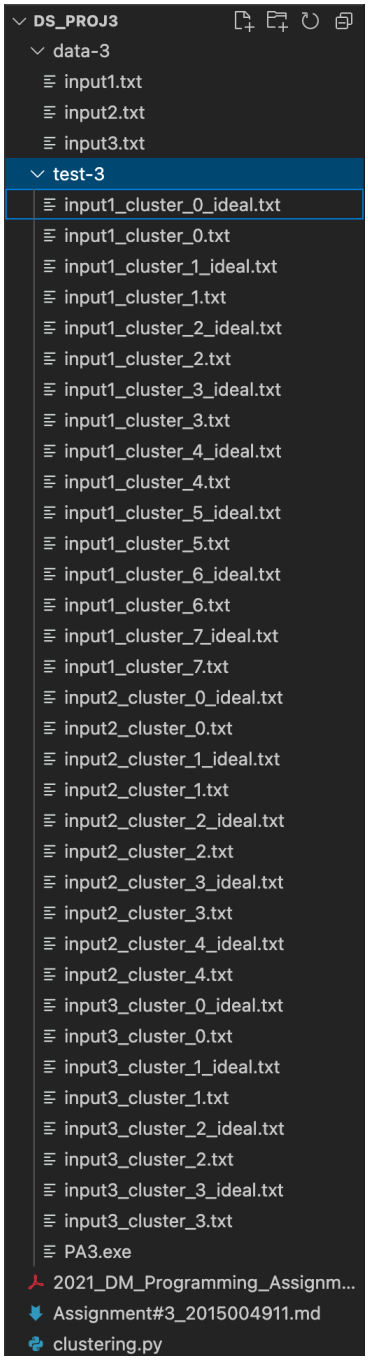
- `__main__` : 인자값을 받고, data를 읽고, clustering을 수행한다. Clustering이 완료되면, output file을 생성한다.

6. 실행 방법 및 실행 결과

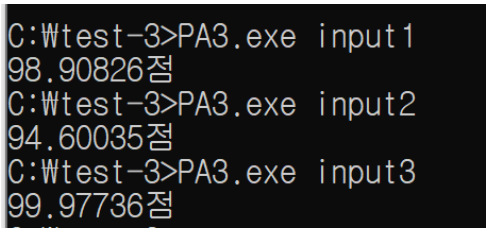
```
(base) limsungjun@Sungjuns-MacBook DS_proj3 % python3 clustering.py input1.txt 8 15 22
[(base) limsungjun@Sungjuns-MacBook DS_proj3 %
```

- `python3 clustering.py input1.txt 8 15 22`

python으로 구현했기에 terminal 상에서는 python을 실행하는 명령어 **`python3`** , **`input file`** , **`n`** , **`eps`** , **`minPts`** 순서로 입력해서 실행한다. `__main__` 코드에 *input file*과 *output file*의 경로를 설정했다. 그렇기 때문에, input file은 **`data-3`** 폴더에 있어야 하고, output file은 **`test-3`** 폴더에 생성된다.



구현한 코드의 file directory 구조이다.



구현한 코드의 실행 결과를 주어진 Testing program, `PA3.exe`으로 실행한 결과이다.

	Test score	My score
input1.txt	99	**98.90826**
input2.txt	95	**94.60035**
input3.txt	99	**99.97736**