Course name: Data Science(ITE4005)
Student id: 2016024793
Name: 김유진

Data Science
Programming Assignment #3

PROGRAMMING ASSIGNMENT #3 – DBSCAN

ENVIRONMENT

OS: Windows 10

Language: python 3.6.8

Used module: math, sys

```
import math
import sys
```

SUMMARY OF ALGORITHM

Given a set of points in some space to be clustered, the points are classified as *core points*, (*density-) reachable points* and *outliers*.

- A point *p* is a *core point* if at least **minPts** points are within distance **eps** of it.
- A point *q* is *directly reachable* from *p* if point *q* is within distance **eps** from the core point *p*.
- A point *q* is *reachable* from *p* if there is a path p_1, …, p_n with $p_1 = q$, where each p_{i+1} is directly reachable from p_i. (This implies that all points on the path must be core points, with the possible exception of *q.*)
- All points not reachable from any other point are *outliers*.

Now if *p* is a core point, then it forms a cluster together with all points (core and non-core) that are reachable from it.

Starting with an arbitrary point P that has not been visited, P's neighborhood is retrieved, and if it contains sufficiently many points, a cluster is started. Otherwise, the point is labeled as an outlier.

If a cluster is started, it begins to grow, considering reachable points as its member. Process that grows certain cluster are done only with the points that are not labeled. The process continues until the density-connected cluster is completely found. Then, a new unvisited point is retrieved and processed, leading to the discovery of a further cluster or noise.

DETAILED DESCRIPTIONS

1. **Read and save input data and return as a list of tuples**

readfile() is a function that reads the data and save it from given file. Given data is saved as a list of tuples. Each tuple contains object id and coordinates.

```
# Function that read inputfile and save the data.
def readFile(filename):
    f = open(filename, "r")
    D = list()
    while True:
        line = f.readline()
        if not line:
            break
        object_id, x_coord, y_coord = line.strip('\n').split('\t')
        tmp = (int(object_id), float(x_coord), float(y_coord))
        D.append(tmp)
    f.close()
    return D
```

2. **Functions for DBSCAN**

Following functions acts as the subprocess of DBSCAN.

```
# Function for calculating distance between two objects.
def distance(p1, p2):
    _, x1, y1 = p1
    _, x2, y2 = p2
    return math.sqrt((x1-x2)**2+(y1-y2)**2)
```

This function returns distance between two objects.

```
# Function for finding neighbors.
# Scans whole dataset and returns list of objects of which
# distance to object P is smaller than eps.
def findNeighbors(D, P, eps):
    neighbors = []
    for Pn in range(len(D)):
        if distance(D[P], D[Pn]) < eps:
            neighbors.append(Pn)
    return neighbors
```

This function returns list of objects id that are neighbors of object P. If distance to object P is smaller than **eps**, than it's added to the list.

```
# Function that finds members of cluster that object P is
# the core point.
def growCluster(D, labels, P, neighbors, C, eps, MinPts):
    # Mark object P as a member of cluster C
    labels[P] = C

    i = 0
    while i < len(neighbors):
        Pn = neighbors[i]           # check object Pn
        # If object Pn is marked as an outlier, mark it as cluster C
        if labels[Pn] == -1:
            labels[Pn] = C
        # Else if object Pn is not marked, mark it as cluster C
        elif labels[Pn] == 0:
            labels[Pn] = C
            PnNeighbors = findNeighbors(D, Pn, eps)
            # If object Pn is not a border point,
            # add Pn's neighbors to neighbors of object P(Grow cluster C).
            if len(PnNeighbors) >= MinPts:
                for tmp in PnNeighbors:
                    if tmp not in neighbors:
                        neighbors.append(tmp)
        i += 1
```

Lastly, this function adds objects to the given cluster when an object P and its neighbors are given. While iterating, it looks for points that are within **eps** (reachable from point P, so consider the point as a member of cluster C) and check if the point is a core point. If it is a core point, then its neighbors are also considered as neighbors of object P. The process stops when neighbors of P are completely checked. As a result, *growCluster()* tracks all points reachable from point P.

3. DBSCAN

Lastly, this function cluster given dataset using functions described above. *DBSACN()* returns a list which contains labels of objects. 0 means that the corresponding object is not labeled yet (unvisited), and -1 means that the object is an outlier.

```
# Density-based spatial clustering of applications with noise
# - Labels
#   0 : not labeled yet
#  -1 : outlier
#   1~: cluster numbering
def DBSCAN(D, eps, MinPts):
    labels = [0]*len(D)
    C = 0
    for P in range(0, len(D)):
        # If P is already labeled, continue.
        if not (labels[P] == 0):
            continue

        neighbors = findNeighbors(D, P, eps)
        # If number of P's neighbors are smaller then MinPts,
        # lable it as an outlier.
        if len(neighbors) < MinPts:
            labels[P] = -1
        # Else, grow cluster C. After this process, cluster C will
        # be fully grown!
        else:
            C += 1
            growCluster(D, labels, P, neighbors, C, eps, MinPts)
    return labels
```

4. Other utility functions

```
# Function that returns top n clusters with respect of its size.
def saveOutput(D, labels, n):
    clusters = dict()
    for i in range(len(D)):
        try:
            obj_id, _, _ = D[i]
            label = labels[i]
            clusters[label].append(obj_id)
        except:
            clusters[label] = [obj_id]
    tmp = list()
    for label, ids in clusters.items():
        # print("cluster {}: {}".format(label, len(ids)))
        if label == -1:
            continue
        tmp.append((label, len(ids)))
```

```
    tmp.sort(key=lambda tuple: tuple[1])

    output = list()
    for label, _ in tmp[-n:]:
        output.append(clusters[label])
    return output


# Function that prints the result
def printOutput(filename, output):
    outputfile = filename.split('.')[0]
    for i in range(len(output)):
        tmpfile = "{}_cluster_{}.txt".format(outputfile, i)
        f = open(tmpfile,"w")
        for o in output[i]:
            f.write("{}\n".format(o))
        f.close()
```

Since cluster labels are saved in 'labels' as a list, *saveOutput()* sort clusters by number of members and picks top n clusters. *printOutput()* takes the output and saves the result on output files.

5. Main function

```
if __name__=="__main__":
    if len(sys.argv) != 5:
        print("need 4 arguments")
    else:
        inputfile = sys.argv[1]
        n = int(sys.argv[2])
        eps = int(sys.argv[3])
        minPts = int(sys.argv[4])

        D = readFile(inputfile)
        labels = DBSCAN(D, eps, minPts)
        output = saveOutput(D, labels, n)
        printOutput(inputfile, output)
```

COMPILATION

Please use the command below.

python clustering.py [*input file name*] [*n*] [*eps*] [*MinPts*]

For example, `>python clustering.py test/input3.txt 4 5 5`.

TEST RESULTS

```
C:\Users\Ho Kim\Dropbox\학교\2019-1\ds\assignment\assignment03\test>PA3.exe input1
98.97037점
C:\Users\Ho Kim\Dropbox\학교\2019-1\ds\assignment\assignment03\test>PA3.exe input2
94.86598점
C:\Users\Ho Kim\Dropbox\학교\2019-1\ds\assignment\assignment03\test>PA3.exe input3
99.97736점
```

REFERENCE

https://en.wikipedia.org/wiki/DBSCAN