Programming Assignment #3

컴퓨터소프트웨어학과 2017029470 김종원

1. Environment

- Mac OS (Monterey) M1 chip
- Python 3.10.3 (release March 16, 2022)

2. How to compile

Before compiling dt.py, python version 3 must be installed in your system.

```
wonnx@wonnx project_3_DBSCAN % python3 clustering.py ./data-2/input1.txt 8 15 22
wonnx@wonnx project_3_DBSCAN % python3 clustering.py ./data-2/input2.txt 5 2 7
wonnx@wonnx project_3_DBSCAN % python3 clustering.py ./data-2/input3.txt 4 5 5
```

- Execution file: clustering.py
- Input data file: input1.txt, input2.txt, input3.txt
- Output file: input#_cluster_i.txt
  - n: number of clusters for the corresponding input data. “i” in the output file is a number between 0 and n.

3. Summary of algorithm

DBSCAN, which is a density-based clustering, considers one cluster if there are n or more points within a radius x of a point. First, suppose that there is a point p, and if there are “minPts” points within a distance “Eps” from the point p, it is recognized as a single cluster. A point p having “minPts” points within a distance “Eps” is called core point.

- I initialize the visit array to false for all points.
- When a certain point is processed, the visit array is changed to true, and the point is checked if it is a core point.
- If the point is a core point, it finds its neighbors and put them in the same cluster.
- After checking whether the point in the neighbor are core points, and if they are core points, the neighbor of the point is also added to the current neighbor.
- The above process is repeated until there are no points left in the neighbor, and when empty, a new cluster is formed using the points whose visit array is false.

4. Detailed description of codes

A. check_core_point

```
4   def check_core_point(point, points):
5       Eps = int(sys.argv[3])
6       x_coordinate = float(point[1])
7       y_coordinate = float(point[2])
8       neighbor = list()
9       for obj in range(len(points)):
10          obj_x = float(points[obj][1])
11          obj_y = float(points[obj][2])
12          distance = np.sqrt(np.power(x_coordinate-obj_x, 2)+np.power(y_coordinate-obj_y, 2))
13          if distance <= Eps: neighbor.append(points[obj])
14      return neighbor
```

This function checks whether a given point a core point. If the distance to all other points is less than Eps, it is considered a neighbor. If the number of points included in the neighbor is MinPts or more, it is core point.

- Euclidean distance was used to find the distance from other points.

B. DBSCAN

```
16  def DBSCAN(points):
17      count = 0
18      cluster = list()
19      MinPts = int(sys.argv[4])
20      visited = [False] * len(points)
21      for point in points:
22          if visited[int(point[0])]==True: continue
23          neighbor = check_core_point(point, points)
24          if len(neighbor)>=MinPts:
25              cluster.append(list())
26              while True:
27                  if len(neighbor)==0: break
28                  next_point = neighbor.pop()
29                  if visited[int(next_point[0])]==True: continue
30                  visited[int(next_point[0])]=True
31                  cluster[count].append(next_point[0])
32                  new_neighbors = check_core_point(next_point, points)
33                  if len(new_neighbors)>=MinPts:neighbor+=new_neighbors
34              count += 1
35          else: visited[int(point[0])]=True
36      cluster.sort(key=len, reverse=True)
37      cluster = [sorted(list(set(list(map(int, l))))) for l in cluster]
38      return cluster
```

If a point is a core point, the points corresponding to its neighbors are added to the same cluster. For points included in the neighbor, if the point corresponds to the core point, a new neighbor is found and included in the same cluster. The newly obtained neighbor is merged with the existing neighbor.

- The **count variable** is used to check how many clusters are currently created.
- A **cluster variable** is a list containing a list, and each list means one cluster.
- Neighbor is in the form of a list containing points, and each point also contains object id, x coordinate, and y coordinate in the form of a list.
- Among the created clusters, the ones with the largest size were made as output, and sorting was performed to compare them with the test data.

C. predict

```
40  if __name__ == "__main__":
41      with open(sys.argv[1], 'r') as file:
42          lines = file.readlines()
43          points = [list(line.strip().split('\t')) for line in lines]
44      cluster = DBSCAN(points)
45      output = sys.argv[1].split('/')[2].split('.')[0]
46      for i in range(int(sys.argv[2])):
47          with open('./test-2/'+output+'_cluster_'+str(i)+'.txt', 'w') as f:
48              for obj in cluster[i]:
49                  f.write(str(obj)+'\n')
```

The points variable is in the form of putting a list in the list, and it receives the input file data in the form of a list and stores it in the list. It was exported according to the format of the output file specified in the task specification and saved in a directory called “test-2”.

5. Testing result

```
C:₩Users₩labor₩test₩test-2>PA3.exe input1
98.97277점
C:₩Users₩labor₩test₩test-2>PA3.exe input2
94.86598점
C:₩Users₩labor₩test₩test-2>PA3.exe input3
99.97736점
```

The testing results for each input file are as above. Since the appearance of the cluster may vary depending on the initial point, it seems to be different form the ideal result.