Data Science [ITE4005]

Programming Assignment #3

: Perform clustering on data set by using DBSCAN

2015004120
Software Engineering
Park Seonha

0. Environment

- OS : Windows
- Language : Python3 (version : 3.4.4)
- Executable file : clustering.py
  (../Programming_Assignment_3/project_clustering/clustering.py)

1. Summary of Algorithm

This programming assignment's goal is "Perform clustering on a given data set by using DBSCAN". Since we get the number of clusters for input data by system parameter and DBSCAN algorithm do not need that as parameter, we should match the number of clusters which the result of DBSCAN clustering and system parameter. So we can separate this assignment's goal by two parts, "perform clustering by using DBSCAN" and "reduce the number of clusters if the result of DBSCAN and the system parameter is different."

Doing Clustering, I checked each point if it is visited or classified using two list. At first, I set all points as UNVISIT and UNCLASSIFIED, and cluster id as zero. Next, check all points in data set if it was visited. If it marked as UNVISIT, mark it as VISIT and check it is noise or not. If it is noise, change its classify status UNCLASSIFIED to NOISE. If it is not, make a cluster and expand it.

While expanding cluster, check all eps-neighbors list of one point if it is UNVISIT or UNCLASSIFIED. If it is not visited, mark it as visited and check its eps-neighbor and if it is bigger than minPts, join its eps-neighbor to core point. If it is not classified or noise, add it in that cluster.

After Clustering, there can be created more clusters than expected in parameter input. So we should remove some clusters if the result of clustering and parameter input is different. While the number of cluster is larger than parameter input, choose the cluster whose size is smallest. And move all points in smallest cluster to nearest bigger cluster. After move all points in cluster, remove that cluster.

2. Description of codes

```
8   VISIT = 1
9   UNVISIT = 0
10  UNCLASSIFIED = 0
11  NOISE = -1
```

Before start program, define some constant. VISIT and UNVISIT is used for check the point is visited or not. UNCLASSIFIED and NOISE is used for list which contains classified data [cluster_id, UNCLASSIFIED, NOISE].

```
131 if __name__ == '__main__':
132     main()
```

```
67  def main():
68      # get command line argument
69      input_file = sys.argv[1]
70      num_of_cluster = int(sys.argv[2])
71      eps = int(sys.argv[3])
72      minpts = int(sys.argv[4])
73
74      with open(input_file) as f:
75          input_data = f.readlines()
76      input_data = [d.strip() for d in input_data]
```

This python program has main function. main function has the main routine of program like file I/O, form data, and also main() has the process to reduce the number of cluster.

At first, get command line argument with proper type casting. Open input file by file name, and get data as list of string.

```
78      # make train_data string to attribute list and get each attribute name and total number of columns
79      object_list = []
80      for line in input_data:
81          each_line = line.split("\t")
82          each_line[0] = int(each_line[0])
83          each_line[1] = float(each_line[1])
84          each_line[2] = float(each_line[2])
85          object_list.append(each_line)
```

object_list is the list of list. its element is the list which has 3 elements(object id, x coordinate, y coordinate).

```
87      # do clustering with DBSCAN and get clusters and number of cluster
88      clustered = DBSCAN(object_list, eps, minpts)
89      clusters = max(clustered)
```

call the function DBSCAN(dataset, eps, minPts) and get result as list whose element is cluster id. clusters is the variable which contains the number of total clusters.

```
14  # get distance between p[idx1, x1, y1] and q[idx1, x2, y2]
15  def dist(p, q):
16      return math.sqrt(abs(p[1] - q[1]) ** 2 + abs(p[2] - q[2]) ** 2)
```

The function dist(p, q) returns the distance between two point p and q. This function calcucates $\sqrt{|x_p - x_q|^2 + |y_p - y_q|^2}$.

```
19  # use in regionQuery(P,eps)
20  def checkEpsNeighbor(p, q, eps):
21      return dist(p, q) < eps
```

The function checkEpsNeighbor(p, q, eps) returns boolean value whether p and q are neighbor or not. This function uses dist(p, q) function and check that value is smaller than eps.

```
24  # return all points within P's eps-neighborhood (including P)
25  def regionQuery(p, all_point, eps):
26      returnPts = []
27      for q in all_point:
28          if checkEpsNeighbor(p, q, eps):
29              returnPts.append(q)
30      return returnPts
```

The function regionQuery(p, all_point, eps) returns the list of points which is P's eps-neighbor after check all points in data set.

```
33  # do DBSCAN algorithm
34  def DBSCAN(D, eps, minPts):
35      cluster_id = 0
36      visited = [UNVISIT] * len(D)
37      classified = [UNCLASSIFIED] * len(D)
38      #check all unvisited points
39      for p in D:
40          if visited[p[0]] == UNVISIT:
41              visited[p[0]] = VISIT
42              N = regionQuery(p, D, eps)
43              if len(N) < minPts:
44                  classified[p[0]] = NOISE
45              else:
46                  cluster_id += 1
47                  expandCluster(p, N, visited, classified, cluster_id, D, eps, minPts)
48      return classified
```

The function DBSCAN(D, eps, minPts) is the implementation of DBSCAN algorithm. It returns the list which contains cluster_id or UNCLASSIFIED or NOISE. Set the list visited and classified as list of 0.

For all point in data set, check each point is visited or not. If it's not visited, mark it as VISIT, and get eps-neighbor of that point. And check it is NOISE or not. if it is not, add new cluster and expand it with the function expandCluster(P,N,visited, C-set, C, dataset, eps, minPts).

```
51  # expand [cluster_id]th cluster
52  def expandCluster(P, N, visited, C_set, C, dataset, eps, minPts):
53      C_set[P[0]] = C
54      #check all eps-neighbors of P
55      while len(N) > 0:
56          P_ = N[0]
57          if visited[P_[0]] == UNVISIT:
58              visited[P_[0]] = VISIT
59              N_ = regionQuery(P_, dataset, eps)
60              if len(N_) >= minPts:
61                  N = N + N_
62          if C_set[P_[0]] == UNCLASSIFIED or C_set[P_[0]] == NOISE:
63              C_set[P_[0]] = C
64          N.pop(0)
```

The function expandCluster(P,N,visited, C-set, C, dataset, eps, minPts) makes cluster with a point which is not NOISE. Using its eps-neighbor, Iterate all point in eps-neighbor and check whether it is NOISE or not. If not, Join its eps-neighbor list and P's

eps-neighbor point. And check the point is not member of other cluster, add that point in cluster. Repeat it while the eps-neighbor list is not empty.

```
91      # separate points by cluster
92      cluster_list = [[] for x in range(clusters)]
93      for o, c in zip(object_list, clustered):
94          if c > 0:
95              cluster_list[c - 1].append(o)
```

After clustering, using the list clustered, separate points as unit of cluster.

```
97      # reduce the number of cluster if it is bigger than that in parameter
98      while num_of_cluster < clusters:
99          # calculate which cluster to merge (smallest cluster)
100         size_of_cluster = [0] * clusters
101         for c in range(len(size_of_cluster)):
102             size_of_cluster[c] = len(cluster_list[c])
103         to_merge = size_of_cluster.index(min(size_of_cluster))
104         # move points to nearest bigger cluster
105         for p in cluster_list[to_merge]:
106             neighbors = regionQuery(p,object_list,eps)
107             distance = sys.maxsize
108             for neighbor in neighbors:
109                 if dist(p,neighbor) < distance and clustered[p[0]] != clustered[neighbor[0]]:
110                     distance = dist(p,neighbor)
111                     nearest = neighbor
112             cluster_list[clustered[nearest[0]]-1].append(p)
113             clustered[p[0]] = clustered[nearest[0]]
114         cluster_list.pop(to_merge)
115         clusters -= 1
116
```

If the number of cluster is bigger than parameter input, remove smallest clusters. Find the smallest cluster and move its points into nearest bigger cluster. After move all points, remove that smallest cluster.

```
117     # write clusters in output file
118     output_form = (input_file.split("."))[0] + "_cluster_"
119     output_files = []
120     for n in range(clusters):
121         output_files.append(output_form + str(n) + ".txt")
122
123     for idx in range(len(output_files)):
124         out_f = open(output_files[idx], "w")
125         cluster_list[idx].sort()
126         to_write = cluster_list[idx]
127         for c in to_write:
128             out_f.write(str(c[0]) + "\n")
```

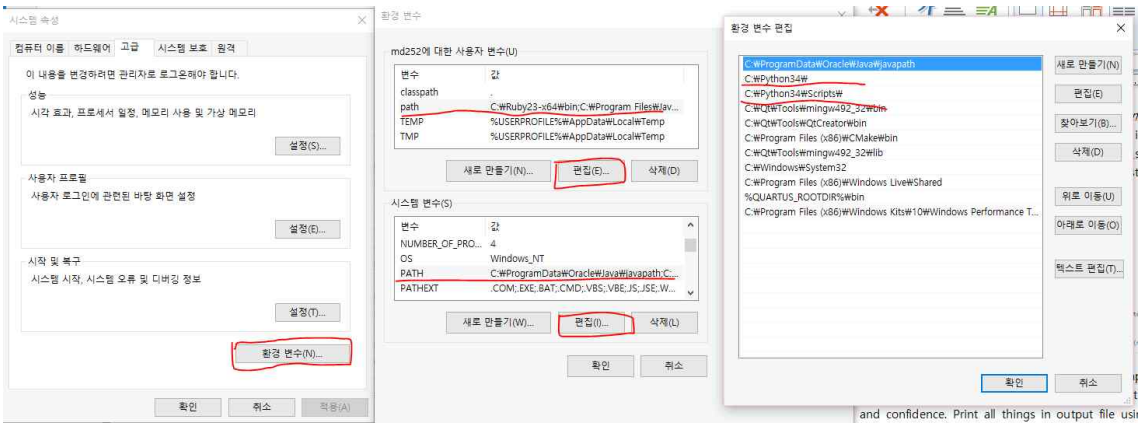
After make proper number of cluster, write its object_id to formed output file.
This program is end in this part.

3. Instructions for compiling this code

This code is written in Python3 and tested in Python 3.4.4. Since this code has **sqrt**(to calculate dist), and it exists in Python3 math module, to run this code, we need python3 interpreter.

We can get python 3.4.4 in (https://www.python.org/downloads/release/python-344/)

If python2 is already installed in PC, change path Python2x to Python34 in advanced system settings – environment variables.

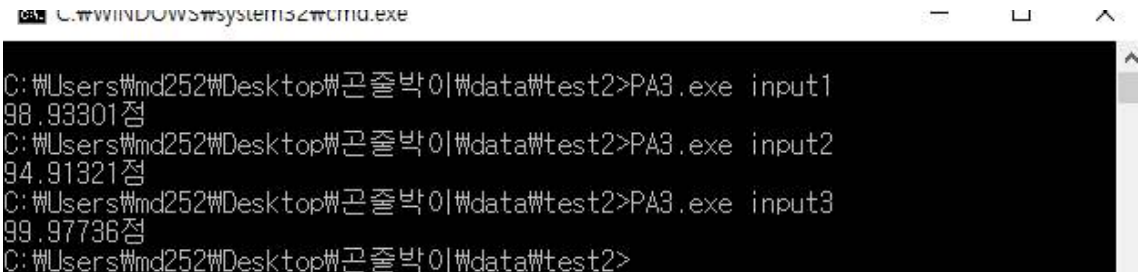


After environment variables setting, we can get python3 default in cmd.

In code directory(python file is in /project_clustering), we can run this program on cmd "python clustering.py [input file name] [# of clusters] [Eps] [minPts]".

The output file will be located in same directory with input file.

4. Other Specifications



In testing, some computer can take quite long time to perform clustering and reduce the number of cluster because of their performance. (more than 5 minutes in case input1) Since it can't check more than the number of all points in each input, it can't make infinite loop, just wait until it will finish.