Data Science Assignment#3

Perform clustering by using DBSCAN + Python3

Lee Eunah - 2018년 5월 20일

Course name: Data Science (ITE 4005)
Professor: Sang-Wook Kim
TAs: Jangwan Koo
Tae-ri Kim

Student name: Lee Eunah(이은아)
Student number: 2016025769
Major: Computer Science Engineering

Implementation environment

- OS: Mac OS 10.12.6
- Language: Python 3.6.3

Summary of my algorithm

DBSCAN is clustering method based on density. So it is important to define whether a point is my neighbor or not. First, the code needs to input four arguments : inputFile, n, eps, minPts. The inputFile is datas that is used for clustering. The n is number of clusters, eps is maximum radius of the neighbor and minPts is minimum number of points in an Eps-neighbor of a given point. If the number of neighbor within the eps is more than minPts, the point is called core point. If it is less than minPts, the point is called border point. And if it is none, the point is called noise point. In brief, the code implemented relationship of core point that consisted of one cluster. And the code can handle an exception when the number of cluster exceeds given n. It needs to remove excess clusters of small size to handle an exception.

Detailed description of my codes

• Module

```
import math
import sys
```

- Importing sys module for using command line arguments.
- Importing math module for using pow() and sqrt() functions.

• File I/O

```
def readInputFile(fileName):
    datas = []
    inputfile = open("./"+fileName, 'r')
    lines = inputfile.readlines()
    for line in lines:
        data = line[:-1].split('\t')
        datas.append(data)

    inputfile.close()
    return datas
```

- Read input file function.

- The file data is divided by attribute. ([object_id] [x_coordinate] [y_coordinate])

```
def writeFile(fileName, checkCluster, numCluster, datas):
    for i in range (len(checkCluster)):
        if checkCluster[i] == numCluster:
            word = ""
            word += "%s\n" % datas[i][0]
            fileName.write(word)

    return fileName
```

- Write output file function.
- Create file for each cluster.

• Key functions

```
# euclidean distance formula
def euclideanDistance(x, p):
    point = 0
    distance = math.sqrt(math.pow(float(datas[x][1]) - float(datas[p][1]), 2) +
                         math.pow(float(datas[x][2]) - float(datas[p][2]), 2))

    return distance
```

- Calculate euclidean distance of two points.
- Formula:

$$Distance(X, P) = \sqrt{(x-a)^2 + (y-b)^2}$$

(let $X(x, y)$, $P(a, b)$)

```
# find out neighbors of point within given epsilon
def checkNeighbor(datas, i, eps):
    neighbor = []

    for p in range (len(datas)):
        if euclideanDistance(i, p) <= eps:
            neighbor.append(p)

    return neighbor
```

- This function is for searching neighbors of point.

- The criteria decided neighbors is within given epsilon. (The epsilon is maximum radius of the neighborhood.)

```
## collect datas for each cluster
def DBScan(datas, n, eps, minPts):
    ## -2: undefined
    ## -1: noise
    checkCluster = [-2 for undefined in range (len(datas))]
    numCluster = 0

    for i in range (len(datas)):
        # skip defined datas
        if checkCluster[i] != -2:
            continue

        # datas that are near the current data (= neighbor)
        neighbor = checkNeighbor(datas, i, eps)

        ## skip noise datas
        if len(neighbor) < minPts:
            checkCluster[i] = -1
            continue

        checkCluster[i] = numCluster

        ## set cluster number of neighbors
        for id in neighbor:
            checkCluster[id] = numCluster

       # search density-reachable in each neighbor
       while len(neighbor) > 0:
            currentNeighbor = neighbor[0]
            recurNeighbor = checkNeighbor(datas, currentNeighbor, eps)

            # core point
            if len(recurNeighbor) >= minPts:
                for i in range (len(recurNeighbor)):
                    point = recurNeighbor[i]
                    if checkCluster[point] == -2 or checkCluster[point] == -1:
                        neighbor.append(point)
                        checkCluster[point] = numCluster

            del neighbor[0]

       numCluster += 1

   return checkCluster, numCluster
```

- This function is for collecting datas for each cluster.
- Cluster labels are initialized to -2. -2 means undefined data, -1 means noise data and more than 0 means the number of cluster.
- If the point is not a core data, it gets out the loop. If it is a core data, keep looking for density-reachable neighbors.

- **Exception**

```
def excess(clusterLabel, num, n):
    count = {}
    for c in range (num):
        count[c] = clusterLabel.count(c)

    key = list(count.keys())
    value = list(count.values())

    # delete the small size of cluster
    for i in range (num - n):
        minimum = min(value)
        minIndex = value.index(minimum)
        minLabelIndex = key[minIndex]
        del value[minIndex]
        del key[minIndex]

    return key
```

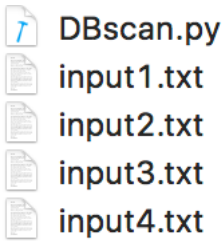
- This function is for deleting excess cluster of small size.

- **Main**

```
## main

# Check the command line arguments
if len(sys.argv) != 5:
    print('''Please fill in the command form.
Executable_file minimum_support inputfile outputfile''')
    exit()

inputFile = sys.argv[1]
n = int(sys.argv[2])
eps = int(sys.argv[3])
minPts = int(sys.argv[4])

split = inputFile.split('.')[0]

datas = readInputFile(inputFile)
clusterLabel, num = DBScan(datas, n, eps, minPts)

# when the number of cluster exceeds given n
if num > n:
    key = excess(clusterLabel, num, n)

    for numCluster in range (n):
        fileName = split + '_cluster_' + str(numCluster) + '.txt'
        outputFile = open(fileName, 'w')
        writeFile(outputFile, clusterLabel, key[numCluster], datas)
        outputFile.close()

# not exceed
for numCluster in range (n):
    fileName = split + '_cluster_' + str(numCluster) + '.txt'
    outputFile = open(fileName, 'w')
    writeFile(outputFile, clusterLabel, numCluster, datas)
    outputFile.close()

print('DBscan done')
```

- Main function
- Check command line exception

Compiling my code



Put all of files in same directory.

```
Eunahui-MacBook-Pro:data leeeunah$ ls
DBscan.py       input1.txt      input2.txt      input3.txt      input4.txt
Eunahui-MacBook-Pro:data leeeunah$
```

- move to that directiory.

```
Eunahui-MacBook-Pro:data leeeunah$ python DBscan.py input1.txt 8 15 22
DBscan done
Eunahui-MacBook-Pro:data leeeunah$ python DBscan.py input2.txt 5 2 7
DBscan done
Eunahui-MacBook-Pro:data leeeunah$ python DBscan.py input3.txt 4 5 5
DBscan done
```

- Enter the command.

```
Eunahui-MacBook-Pro:data leeeunah$ ls
DBscan.py               input1_cluster_6.txt    input3.txt
input1.txt              input1_cluster_7.txt    input3_cluster_0.txt
input1_cluster_0.txt    input2.txt              input3_cluster_1.txt
input1_cluster_1.txt    input2_cluster_0.txt    input3_cluster_2.txt
input1_cluster_2.txt    input2_cluster_1.txt    input3_cluster_3.txt
input1_cluster_3.txt    input2_cluster_2.txt    input4.txt
input1_cluster_4.txt    input2_cluster_3.txt
input1_cluster_5.txt    input2_cluster_4.txt
```

- Output files are created.

```
C:₩Users₩Leeeunah>cd Desktop/assignment3

C:₩Users₩Leeeunah₩Desktop₩assignment3>PA3.exe input1
98.98721점
C:₩Users₩Leeeunah₩Desktop₩assignment3>PA3.exe input2
94.83162점
C:₩Users₩Leeeunah₩Desktop₩assignment3>PA3.exe input3
99.97736점
```

- Result of executing the test program.

Any other specification about my code

- My code can remove excess cluster of small size. It is important factor to improve accuracy.
- The result is almost accurate.
- Each point searches its neighbors which is density-reachable to classify clusters.
- Elements in checkCluster list, -2 means undefined point, -1 means noise point and more than 0 means the number of cluster.