Report for hw3

2018000337 장호우

Environment

- Windows 10
- Python 3.7.7 64-bit

Requirements

- numpy

To install requirements

```
pip3 install numpy
```

Content

Summary of your algorithm

DBSCAN requires two parameters: eps and the minimum number of points required to form a dense region(minPts). It starts with an arbitrary starting point that has not been visited. This point's e-neighborhood is retrieved, and if it contains sufficiently many points, a cluster is started. Otherwise, the point is labeled as noise.

Detailed description of each function

```
# Point, type: -1: Noise; other num: visited
class Point:
    def __init__(self, args):
        self.id = int(args[0])
        self.x = args[1]
        self.y = args[2]
        self.type = None
```

```
# Main programming
class DBSCAN:
    def __init__(self, data, eps, minPts):
        self.data = data
        self.eps = eps
        self.minPts = minPts
        self.clusters = defaultdict(list)
    # return all points within current point's eps-neighborhood
    def regionQuery(self, curr_p):
        # calculate the distance of two points
        def dist(p1, p2):
            return sqrt((p1.x-p2.x)**2 + (p1.y-p2.y)**2)
        return [p for p in self.data if dist(p, curr_p) <= self.eps]
```

```
    def clustering(self):
        count_cluster = 0
        for p in self.data:
            if p.type: continue
            # mark Ppint p as visited
            p.type = count_cluster
            neighbors = self.regionQuery(p)
            if len(neighbors) < self.minPts:
                # mark Point p as Noise
                p.type = -1
            else:
                count_cluster += 1
                # expand the cluster
                self.clusters[count_cluster].append(p)
                for q in neighbors:
                    # if Point q is not visited, mark Point q as visited
                    if not q.type:
                        q.type = count_cluster
                        neighbor = self.regionQuery(q)
                        if len(neighbor) >= self.minPts:
                            [neighbors.append(n) for n in neighbor]
                    # if Point q is not yet member of this cluster, add Point q
to this cluster
                    if not (q in self.clusters[count_cluster]):
                        self.clusters[count_cluster].append(q)
        return self.clusters
```

```
if __name__ == "__main__":
    # parsing
    parser = argparse.ArgumentParser()
    parser.add_argument("input", type=str)
    parser.add_argument("n", type=int)
    parser.add_argument("eps", type=int)
    parser.add_argument("minPts", type=int)
    args = parser.parse_args()
    # load data to Points list
    data = list(map(Point, np.loadtxt(args.input).tolist()))
    clusters = DBSCAN(data, args.eps, args.minPts).clustering()
    # slice the dict from 0 to number of clusters requested without Noise
    dict_slice = lambda adict, start, end: { k:adict[k] for k in
list(adict.keys())[start:end] }
    clusters = dict_slice(clusters, 0, args.n)
    # save the data
    for key, value in clusters.items():
        np.savetxt(
            '{}_cluster_{}.txt'.format(args.input.split('.')[0], key-1),
            list(map(lambda x: x.id, value)),
            fmt='%d'
        )
```

How to run it

Running

For Example 1:

```
python clustering.py input1.txt 8 15 22
```

For Example 2:

```
python clustering.py input2.txt 5 2 7
```

For Example 3:

```
python clustering.py input3.txt 4 5 5
```

Testing

For Example 1:

```
./PA3.exe input1
```

output:

```
98.98035?
```

For Example 2:

```
./PA3.exe input2
```

output:

```
94.89474?
```

For Example 2:

```
./PA3.exe input3
```

output:

```
99.97736?
```