Programming Assignment #3 Report

2016024766

김서현

1. 구현한 DBSCAN 알고리즘에 대한 설명

DBSCAN은 density를 기반으로 clustering을 하는 알고리즘입니다. Density-connected인 point들을 계속 cluster로 묶으며 확장해 나가는 것이 DBSCAN 알고리즘의 핵심입니다. 직접 구현한 DBSCAN의 알고리즘은 다음과 같습니다.

1) Input file을 열어 points를 배열에 넣고 하나씩 순회하며 아래의 과정을 진행한다.
2) 우선 point가 어디에도 속하지 않은 UNASSIGNED 상태인지 확인한다.
    A. UNASSIGNED 상태라면 아래 과정을 진행한다.
    B. 이미 cluster에 속해 있거나 NOISE라고 라벨링이 됐다면, 그 point를 스킵하고 다음 point로 넘어간다.
3) Point의 neighbor 수를 확인하고, MinPts보다 많다면 core point로 정하고, 새로운 cluster를 생성한다.
    A. 만일 neighbor의 수가 적어 core point가 아니라면 NOISE로 라벨링을 한 뒤 input file의 다음 point로 넘어간다.
4) Neighbor points를 모두 새로운 cluster에 넣고 해당 cluster_id로 라벨링을 한다.
5) Cluster의 point를 순회하면서 다음의 과정을 반복한다.
    A. Point가 NOISE로 라벨링된 상태이면 cluster_id로 라벨을 바꿔준다.
    B. Point가 UNASSIGNED 상태이면 cluster_id로 라벨링을 해주고, core point라면 해당 point의 neighbor 중 현재 cluster에 속하지 않은 것들을 현재 cluster에 넣어준다.
    C. Point가 다른 cluster에 속해 있는 상태이고, core point라면 그 cluster와 현재 확장중인 cluster를 합친다.
    D. Point가 이미 해당 cluster에 속해 있는 상태라면, core point인지 확인한 뒤 그 point의 neighbor 중 현재 cluster에 속하지 않은 것들을 현재 cluster에 포함시킨다.
6) 모든 point에 대해 라벨링이 끝나면 cluster의 크기가 큰 순으로 정렬한 뒤 output files에 clusters를 write한다.

2. 코드에 대한 설명

1) Main

```python
if __name__ == '__main__':
    if len(sys.argv) < 5:
        print(
            "USAGE: python clustering.py <input filename> <number of clusters> <Eps> <MinPts>")
        sys.exit()
    filename = f'./data-3/{sys.argv[1]}'
    n = int(sys.argv[2])
    Eps = int(sys.argv[3])
    MinPts = int(sys.argv[4])
    cluster_id = 0
    df = pd.read_csv(filename, sep='\t', header=None)
    data = df.values.tolist()
    label = [UNASSIGNED]*len(data)
    start_time = time.time()
```

우선 argument로 입력 받은 값들을 읽어오고, 입력이 잘못됐다면 USAGE를 띄우고 프로그램을 종료합니다. Input file을 열어서 그 안의 data를 읽어오고, cluster_id 혹은 NOISE나 UNASSIGNED로 상태를 저장할 label 배열을 생성합니다. Label은 UNASSIGNED로 초기화 합니다.

```python
# scan all data
for X in data:
    now_id = int(X[0])
    # unassigned point가 아니라면 continue
    if label[now_id] != UNASSIGNED:
        continue
    neighbor = get_neighbor(data, X, Eps)
    # if core point, then make new cluster
    if len(neighbor) >= MinPts:···
    # if not core point, then NOISE
    else:
        label[now_id] = NOISE
```

Data의 points를 순회하면서 해당 point가 UNASSIGNED 상태가 아니면 다음 point로 넘어갑니다. UNASSIGNED 상태이긴 하지만 core point는 아니라면, NOISE로 라벨링하고 다음 point로 넘어갑니다. Point가 UNASSIGNED 상태이면서 core point일 때의 코드에 대해서는 아래에서 설명합니다.

```
# if core point, then make new cluster
if len(neighbor) >= MinPts:
    now_cluster = []
    now_cluster.extend(neighbor)
    cluster_id += 1
    # 첫번째 neighbor points의 label들을 cluster_id로 assign함
    for nc in now_cluster:
        if label[int(nc[0])] == UNASSIGNED:
            label[int(nc[0])] = cluster_id
    # 더이상 확장할 수 없을 때까지 cluster를 계속 확장함
    for i in now_cluster:
        next_id = int(i[0])
        # if noise point
        if label[next_id] == NOISE:
            label[next_id] = cluster_id
        # if unassigned point
        elif label[next_id] == UNASSIGNED:
            label[next_id] = cluster_id
            new_neighbor = get_neighbor(data, i, Eps)
            # if core point
            if len(new_neighbor) >= MinPts:
                for nn in new_neighbor:
                    if label[int(nn[0])] != cluster_id:
                        now_cluster.append(nn)
```

해당 point가 core point일 때의 코드입니다. 우선 cluster를 새로 생성하고, cluster_id도 새로 만듭니다. now_cluster에 neighbor points를 삽입하고 그 points의 label을 cluster_id로 변경합니다.

now_cluster의 points를 순회하면서 만일 point의 라벨이 NOISE라면 라벨을 cluster_id로 변경하고 넘어갑니다. (border point라고 판단)

만일 point의 라벨이 UNASSIGNED라면 해당 point의 label을 cluster_id로 변경하고, core point인지 확인합니다. Core point라면 neighbor points들을 확인해서 해당 cluster에 속하지 않은 points들을 모두 cluster에 포함시킵니다.

```
# if point is already assigned in another cluster
elif label[next_id] != cluster_id:
    # if core point
    if len(get_neighbor(data, i, Eps)) >= MinPts:
        diff_cluster_id = label[next_id]
        for t in range(len(label)):
            if label[t] == diff_cluster_id:
                label[t] = cluster_id
                now_cluster.append(data[t])
# if point is already assigned in now_cluster
elif label[next_id] == cluster_id:
    new_neighbor = get_neighbor(data, i, Eps)
    # if core point
    if len(new_neighbor) >= MinPts:
        for nn in new_neighbor:
            if label[int(nn[0])] == cluster_id:
                continue
            else:
                label[int(nn[0])] = cluster_id
                now_cluster.append(nn)
```

해당 point가 이미 다른 cluster에 속해 있다면, 우선 core point인지 확인합니다. Core point가 맞다면, 그 point가 속한 cluster와 now_cluster를 합칩니다. (now_cluster가 다른 cluster를 포함하는 형태로 합침)

해당 point가 이미 now_cluster에 속해 있는 상태라면 core point인지 border point인지 확인합니다. Core point라면 해당 point의 neighbor points를 구해서 now_cluster에 속하지 않은 point들을 now_cluster에 포함시킵니다.

```
print(time.time()-start_time)
inputname = sys.argv[1].split('.')[0]
order = make_order(label)
print(order)
write_file(label, inputname, n, order)
```

DBSCAN을 하는 데에 걸린 시간을 계산하고, write_file 함수를 이용해 각 cluster들을 output file에 write하고 종료합니다.

2) Distance

```
# calculate distance between two points
def distance(x1, y1, x2, y2):
    return ((x1-x2)**2+(y1-y2)**2)**(1/2)
```

두 point 사이의 distance를 계산하는 함수입니다.

3) Get_neighbor

```
# make a list of neighbor points
def get_neighbor(data, X, Eps):
    neighbor = []
    for i in data:
        if (distance(i[1], i[2], X[1], X[2]) <= Eps):
            neighbor.append(i)
    return neighbor
```

현재 point X의 neighbor points를 모두 찾아 배열에 넣고 리턴합니다. 자기 자신도 neighbor에 포함됩니다.

4) Make_order

```
# cluster의 크기가 큰 순으로 order를 만듦
def make_order(label):
    cnt = {}
    # cluster의 크기를 알기 위해 cnt에 cluster당 point개수를 저장함
    for val in label:
        if val not in cnt:
            cnt[val] = 1
        else:
            cnt[val] += 1
    order = sorted(cnt.items(), reverse=True, key=lambda item: item[1])
    return order
```

Label을 보고 각 cluster의 개수를 센 뒤, order에 내림차순으로 정렬해서 cluster_id와 크기를 넣어 리턴합니다.

5) Write_file

```python
def write_file(label, inputname, n, order):
    cnt = 0
    # 크기가 더 큰 cluster의 순으로 file을 만들어 write함
    for key, val in order:
        # Noise이면 무시함
        if key == NOISE:
            continue
        # n개만큼의 file을 write하면 종료함
        if n == cnt:
            break
        id = key
        cluster = []
        for idx in range(len(label)):
            if label[idx] == id:
                cluster.append(idx)
        file_name = f'./test-3/{inputname}_cluster_{cnt}.txt'
        f = open(file_name, 'w')
        for value in cluster:
            f.write(str(value)+'\n')
        f.close()
        cnt += 1
    # if number of clusters < n, then make more empty files
    while cnt < n:
        file_name = f'./test-3/{inputname}_cluster_{cnt}.txt'
        f = open(file_name, 'w')
        f.close()
        cnt += 1
```

Label과 order를 이용해 output file에 각 cluster를 write하는 함수입니다. 전체 cluster 개수가 n보다 크다면, 크기가 큰 순서로 cluster n개를 저장합니다. 만일 전체 cluster 개수가 n보다 작다면, 빈 파일을 더 생성해서 총 n개의 파일을 생성하도록 합니다.

3. 컴파일 방법

Python 3.9.1 버전으로 작성한 프로그램입니다.
테스트는 windows10 내의 command prompt를 사용하여 진행했습니다.

1) Pandas library를 사용하기 때문에 pandas를 설치해야 합니다.
   - pip install pandas
2) clustering.py 소스파일, data-3 폴더, test-3 폴더를 같은 디렉토리 안에 넣습니다.
3) data-3 폴더 안에는 input 파일들을 넣습니다.
4) test-3 폴더 안에는 input 파일과 clustering한 결과 파일, 정답 파일, test 프로그램(PA3.exe)을 모두 넣습니다.

5) "python clustering.py <input filename> <number of clusters> <Eps> <MinPts>"의 형식으로 실행시킬 수 있습니다.

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

4. 실행 결과

1) Input1.txt

```
₩test-3>PA3.exe input1
98.97691점
```

실행시간: 41.5초

2) Input2.txt

```
₩test-3>PA3.exe input2
94.86023점
```

실행시간: 2.7초

3) Input3.txt

```
₩test-3>PA3.exe input3
99.97736점
```

실행시간: 3.0초

5. 특이사항

1) Clustering.py를 실행시키면 label 배열과 걸린 시간을 출력합니다.

2) 결과 파일은 바로 test-3 폴더 내에 생성되도록 구현했습니다.

3) 주어진 input 파일들로 clustering을 해서 생성된 결과 파일들도 git에 함께 제출하였습니다.