DBSCAN Report

2016024893 오성준

1. **Summary of your algorithm**

* Data를 받은 뒤에 Clustering을 실행합니다. 다음과 같은 단계로 clustering을 실행합니다.

1. 전체 Data를 돌면서 Neighbor들을 찾아서 2중 list의 형태로 저장합니다. Neighbor의 기준은 두 점 사이의 거리가 EPS보다 작거나 같을 때 입니다.
2. 1번에서 저장한 Neighbor를 기준으로 다시 전체를 돌면서 Neighbor의 수가 Min Points 값보다 같거나 크다면 Core Point라 생각하고 그 Key값을 list에 저장합니다.
3. 마지막으로 저장한 Core point에 대해서 BFS를 실행해서 cluster에 대해서 점들을 모아 둔 뒤에 저장합니다. 그 중에서 개수가 많은 N개의 cluster를 output file에 format에 맞게 write합니다.

* 구현에 대한 세부적인 사항은 2번에서 설명하겠습니다.

1. **Detailed description of your codes (for each function)**

* DBSCAN이라는 class를 만들어서 구현했습니다. 그리고 main function을 만들어서 DBSCAN object를 만들어서 과제를 수행했습니다. 다음부터 DBSCAN class function에 대해서 설명하겠습니다.

1. **Init function**

def \_\_init\_\_(self, data, n, eps, minpts, input\_f):

self.input\_string = input\_f

self.N = n

self.Eps = eps

self.Minpts = minpts

self.Data = {}

self.Data\_size = data.shape[0]

# dataframe is tooooooo slow to get neighbor

# so i use dictionary (hash)

for i in range(self.Data\_size):

d = data.iloc[i]

self.Data[d['key']] = (d['x'], d['y'])

self.Neighbors = {}

self.Neighbors = {int(key): [] for key,xy\_tuple in self.Data.items()}

self.get\_neighbor()

# find directly density reachable relationships

# it's like vector in cpp

self.Core = [key for key, neighbors in self.Neighbors.items() if len(neighbors) >= self.Minpts]

# list comprehension is used to pick up core point

self.Clusters = []

self.cluster()

self.write\_clusters()

* DBSCAN과 관련된 정보를 받은 다음 class에 선언된 함수들을 불러서 전체적인 프로젝트의 진행을 보여줍니다.
* 처음에는 data를 dataframe의 형태로 받아서 진행하려 했으나, 코드를 구현하면서 프로그램의 실행 시간이 너무 느려져서 dictionary를 사용해서 최대한 실행 시간을 줄여보았습니다.
* 그 이후에 각 key에 대해서 neighbor를 저장하는 get\_neighbor function을 사용했습니다. 그러고 난 후엔 list comprehension을 사용해서 core points의 key 값을 저장한 후에 clustering을 하고 각 cluster들을 정해진 file 형식으로 write하는 과정을 거쳐서 프로그램을 실행합니다.

1. **Get distance function**

def get\_distance(self,a,b):

return ((a[0]-b[0])\*\*2 + (a[1]-b[1])\*\*2)\*\*0.5

* 말 그대로 두 점 사이의 거리를 구하는 함수입니다. Neighbor를 구할 때 사용합니다.

1. **Get neighbor function**

def get\_neighbor(self):

for i in range(self.Data\_size):

point\_i = self.Data[i]

for j in range(i + 1, self.Data\_size):

point\_j = self.Data[j]

distance = self.get\_distance(point\_i, point\_j)

if distance <= self.Eps:

self.Neighbors[i].append(j)

self.Neighbors[j].append(i)

* N combinations 2 만큼 두 점의 조합을 보면서 두 점 사이의 거리를 구하고 그 값이 EPS 값보다 같거나 작으면 쌍방의 neighbor이기 때문에 서로의 neighbor에 등록했습니다.

1. **Cluster function**

def cluster(self):

# a = {} print(1 in a) -> return false

# check\_dict = {}

visit = [False for i in range(self.Data\_size)]

# in insert delete list is O(N) but queue is O(1) so i import queue

# q.put(self.Core[0])

for core\_point in self.Core:

if visit[core\_point]: continue

q = Queue()

q.put(core\_point)

tmp = []

while True:

if q.qsize() == 0: break

now = q.get()

if now not in self.Core: continue

# for문 안에서 계속 확인하는 것보다 이게 나을것이라 생각함

for point in self.Neighbors[now]:

if visit[point]: continue

q.put(point)

tmp.append(point)

visit[point] = True

self.Clusters.append(tmp)

self.Clusters.sort(key = len)

self.Clusters.reverse()

* Core point에 대해서 BFS를 실행해서 cluster들을 연결했습니다.
* 여기서 list를 사용하지 않고 외부 queue를 사용했는데, 그 이유는 list를 이용한 pop과 insert의 시간복잡도가 O(N)인데 비해서, 외부 queue는 시간복잡도가 O(1)이라서 시간을 줄여 줄거라 생각했기 때문입니다.
* 두번째로 visit list를 사용했는데, data가 많을 때는 문제가 있을 수 있다는 생각이 들었습니다. 우선은 구현했지만 나중에 좋은 방법을 찾아보고 싶었습니다. 아무튼 이 list는 queue에 넣거나 cluster에 등록할 때, 이미 방문했다면, 중복해서 처리하지 않도록 하는 list로 방문했다면 값이 True입니다.
* Cluster들을 class의 clusters list에 append를 해서 코드를 마무리 지었습니다.

1. **Write clusters function**

def write\_clusters(self):

for i in range(self.N):

file\_name = self.input\_string + "\_cluster\_" + str(i) + ".txt"

f = open(file\_name, 'w')

self.Clusters[i].sort()

for point in self.Clusters[i]:

f.write(str(point)+'\n')

f.close()

* File들을 format에 맞게 만들어서 저장합니다. Cluster를 개수만큼 sorting한 다음에 N개만큼만 file을 저장합니다.

1. **Instructions for compiling your source codes at TA's computer**

**1) requirements.txt**

이번 과제부터 python 가상 환경에서 진행하여 **requirements.txt** 파일을 첨부해서 함께 git lab에 올립니다. 보고서 3번 내용 자체를 지금에서야 이해하게 된 점에 대해 죄송합니다. 동시에 보고서에 사진도 첨부합니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

**2) 실행 명령어**

파일의 실행은 명세와 똑같이 구현했습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

**3) 채점**

채점 시스템이 잘 작동하지 않아서 친구에게 부탁해서 채점을 해보았습니다. 잘 채점이 된 모습입니다.

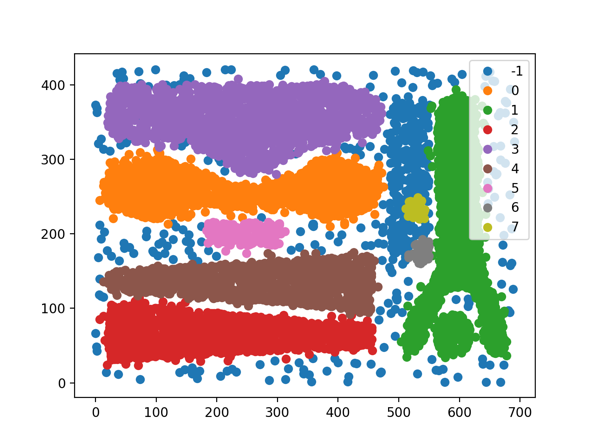
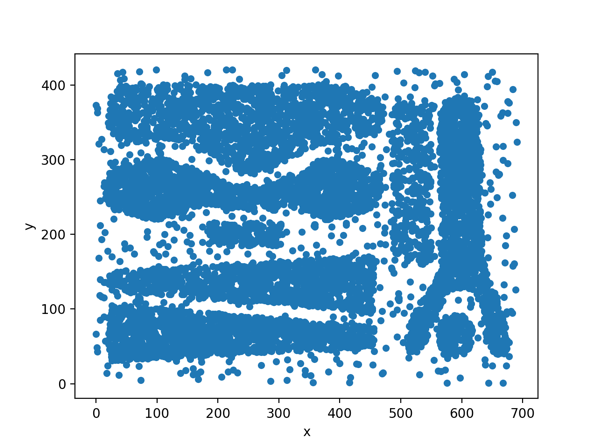
텍스트이(가) 표시된 사진

자동 생성된 설명

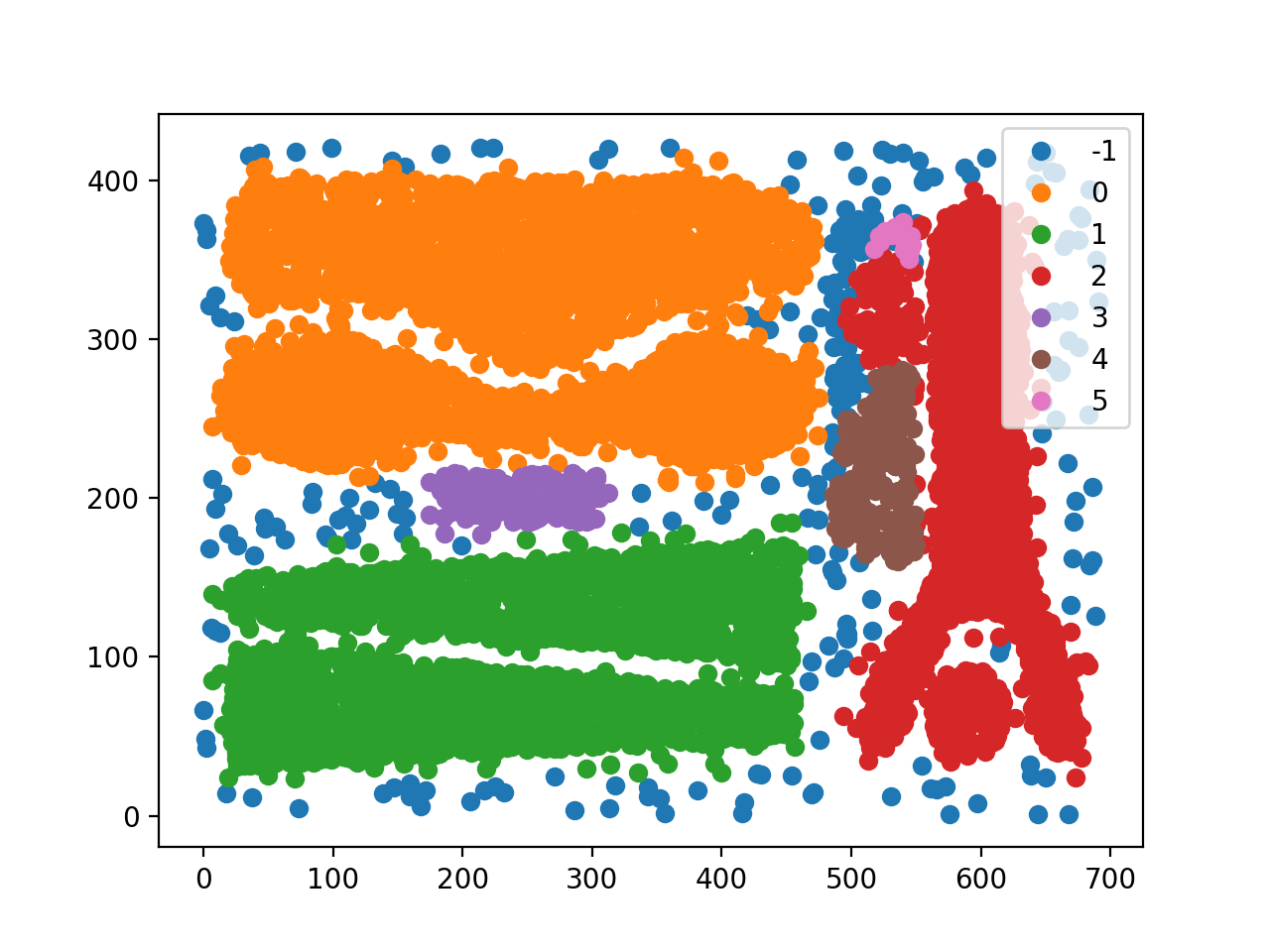
1. **Any other specification of your implementation and testing**

* 이번 과제를 진행하면서 data들을 시각화 하고 싶다는 생각이 들어서 **show.py**에 matplotlib을 이용해서 scatter로 점들을 시각화 했습니다. 동시에 clustering 이후에 색깔을 붙여서 다시 시각화 해서 전 후를 비교할 수 있도록 했습니다. 아래에 결과물을 첨부합니다.

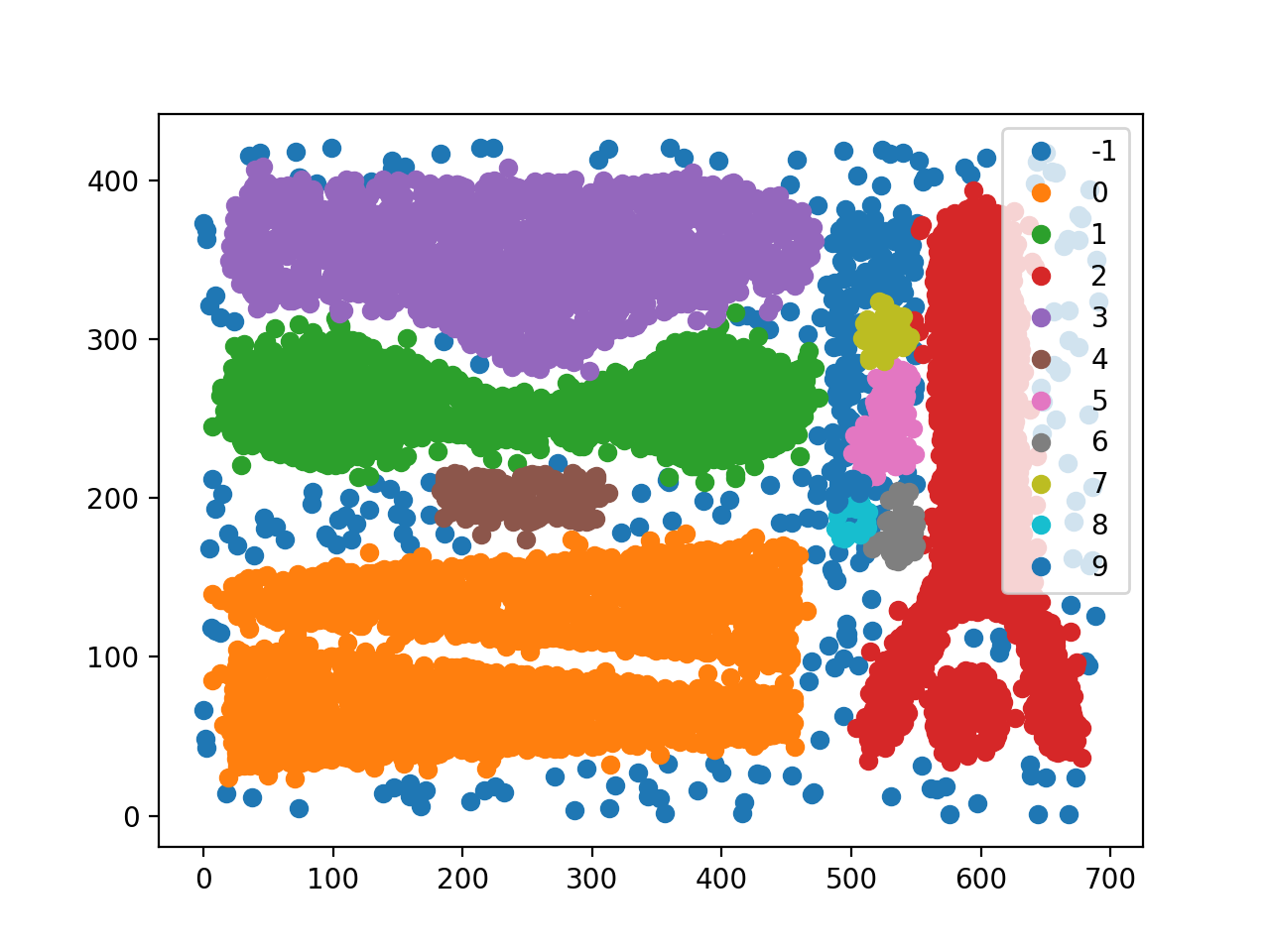
1. **Input1.txt**



* 아래로 긴 직사각형 모양의 cluster가 제대로 잡히지 않았는데, 우선 결과값은 대략 99점으로 제대로 나오긴 했습니다.
* Eps 값을 늘리거나 min points 값을 줄이면 cluster에 포함되곤 했지만, 단독으로 존재하기 보단, 주변의 cluster와 함께 묶이는 상황이 나타나곤 했습니다.
  + Eps 값이 17, min points 값이 22일 때,

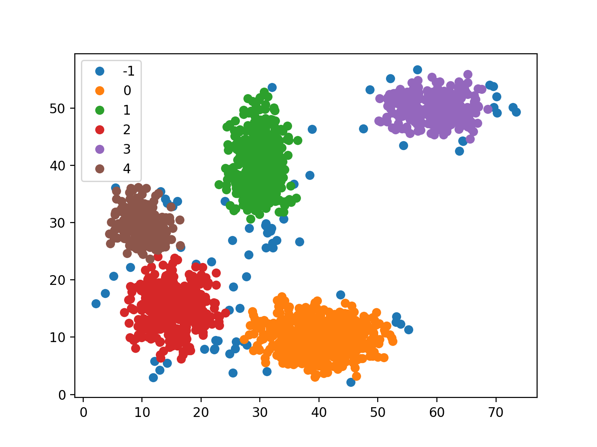
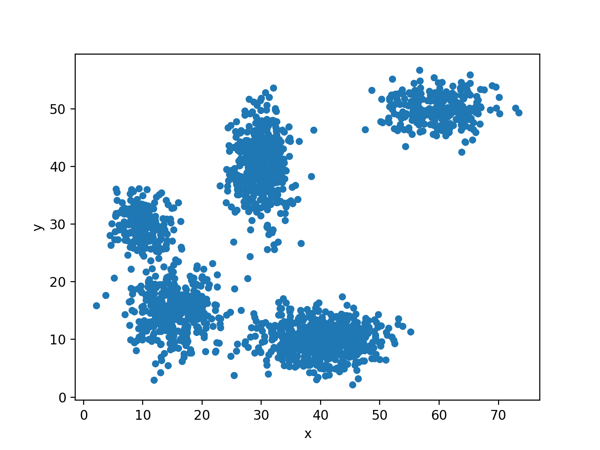


* + - Cluster의 수가 전체적으로 줄었고, 더 많은 점이 cluster에 포함된 모습이지만 결과가 만족스럽진 못하다고 생각합니다. 그 이유는 구분되어야 할 cluster가 합쳐졌기 때문입니다.
  + Eps 값이 15, min points 값이 20일 때,

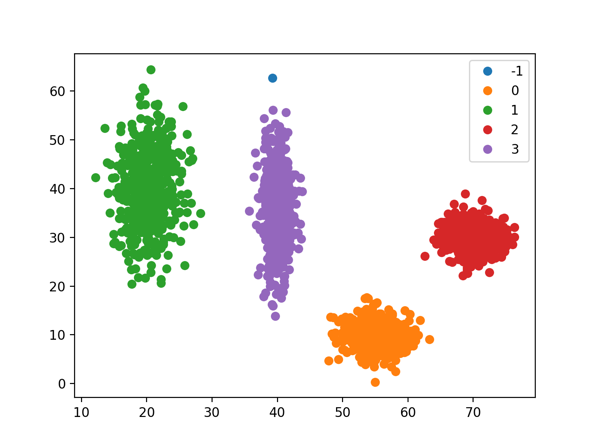
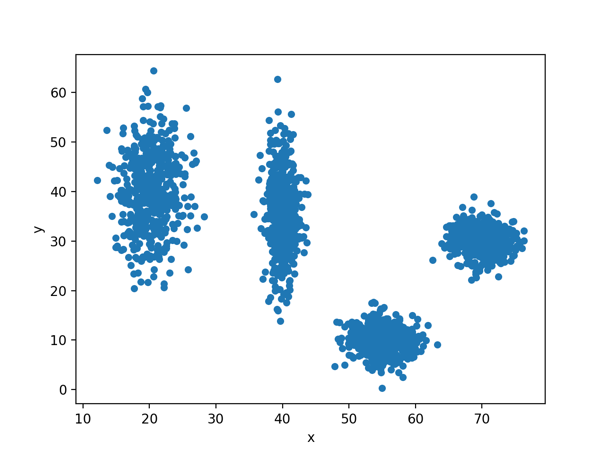


* + - 확실히, 아래로 긴 직사각형 부분에서 clustering되는 부분이 많아졌지만, 그럼에도 불구하고 각자가 쪼개진 모습입니다. 그리고 이번에는 주황색에 해당하는 부분이 합쳐졌기 때문에 마찬가지로 지금 시점이 이득이라고 볼 수 없을 겁니다.
* 이번 시도를 통해서 DBScan이 확실히 parameter에 민감하다는 것과, 시각적으로 완벽히 cluster라고 할 수 있는 것들이 제대로 clustering 되지 않을 수 있다는 것을 알 수 있었습니다.

1. **Input2.txt**



1. **Input3.txt**



* Input2와 input3에 관해선 특별한 점이 발견되지 않고 잘 clustering 됬다고 생각해서 다른 실험을 해보진 않았습니다.
* 전반적인 data의 분포와 결과를 분석할 수 있고, 여러가지 실험을 해볼 수 있어서 좋았습니다. Show.py의 구현 부분은 다음과 같습니다.
* Cluster를 나누기 전의 코드

data = pd.read\_csv(input\_f, header = None, sep = '\t', names = ['key', 'x, 'y'])

data.plot.scatter(x = 'x', y = 'y')

plt.show(block = True)

dbscan = DBSCAN(data ,int(n), int(eps), int(minpts), input\_f[:-4])

* Scatter와 plt를 이용해서 단순한 점들의 분포를 보여주도록 했습니다.
* Cluster를 나눈 이후의 코드

# for c in self.Clusters: print(len(c))

self.label = [-1 for i in range(self.Data\_size)]

self.write\_clusters()

self.D['label'] = self.label

groups = self.D.groupby('label')

fig, ax = plt.subplots()

ax.margins(0.05) # Optional, just adds 5% padding to the autoscalig

for name, group in groups:

ax.plot(group['x'], group['y'], marker='.', linestyle='', ms=12, label=name)

ax.legend()

plt.show()

* 각 점에 대해서 label을 받은 뒤에 label에 맞게 색깔을 다르게 해서 보여주도록 했습니다.

1. **이전 과제에 대해서**

* 앞서 말한 것처럼 저는 지금에서야 보고서 3번 문항에 대해서 알게 되었고, 모든 과제를 가상 환경에서 다시 실행했습니다. 그리고 현재의 requirements.txt 환경에서 잘 작동하는 것을 확인했습니다. 이전 과제들도 채점하실 때 참고 하셨으면 좋겠습니다.
* 2번째 과제에 대해서 말씀드릴 건 제가 구현한 dt.py 코드를 보면서 보고서를 쓰던 도중에 한글로 주석을 달아놓고 그대로 내는 바람에 아마도 dt.py가 잘 실행이 되지 않았을 수도 있을 것 같습니다.

# -\*- coding: utf-8 -\*-

이 주석을 가장 맨 위에 붙여놓으면 잘 작동하는 것을 확인했습니다. 참고해서 채점해 주셨으면 합니다.

* 여러모로 이런 부분들을 미리 확인하지 못하고 과제를 제출한 뒤에 뒤늦게 말하는 점 죄송합니다. 이번 과제와 recommender 과제는 주의해서 보고서를 잘 작성하도록 하겠습니다.