Data Science (ITE4005)

< Programming Assignment #3 >

DBScan Algorithm

2016025732 이영석

2021-06-01

1. Summary

주어진 데이터셋을 클러스터링하는 DBScan Algorithm을 구현한다.

Hyper Parameter : n(몇 개의 클러스터 형성할건지), Eps(이웃이 되기위한 최대거리), MinPts(최소한의 이웃의 수)

point를 골랐을 때 core point가 되어야 클러스터를 형성할 수 있는데, 조건은 다음과 같다.

1. (neighbors 수) >= MinPts
2. (각 neighbor와 해당 point의 distance) <= Eps

```
if __name__=='__main__':
    input_name = sys.argv[1]

    set_data()
    dbScan= DBScan(n= sys.argv[2], eps= sys.argv[3], min_pts= sys.argv[4])

    t1 = time.time()
    top_n_label = dbScan.clustering()
    t2 = time.time()

    dbScan.print_output(top_n_label)
    print('Clustering time : ',t2-t1)
```

주 알고리즘의 흐름은 다음과 같다.

Setting Data -> Clustering -> Printing Output(해당 클러스터에 속해있는 id)

Clustering은 다음과 같은 모듈로 구성하였다. (자세한 사항은 2. Detailed Description 참고)

A. clustering()

i. set_label(id, cluster_id)

ii. get_neighbors(id)

B. get_distance(point1,point2)

2. Detailed Description

A. set_data():

```
def set_data():
    global data_origin

    df = pd.read_csv(input_name, sep='\t', engine='python', encoding="cp949", header=None)
    df.rename(columns={0:'index',1:'x', 2:'y',3:'cluster'},inplace=True)

    df['cluster'] = None # Class label 생성
    data_origin = df
```

해당 txt를 df로 가져와 행에 이름을 매기고, 'cluster'행을 추가한다

B. get_distance(point1,point2) :

```
def get_distance(point1, point2):
    x_change = data_origin.loc[point2, 'x'] - data_origin.loc[point1, 'x']
    y_change = data_origin.loc[point2, 'y'] - data_origin.loc[point1, 'y']

    return(np.sqrt(np.square(x_change)+ np.square(y_change)))
```

두 점 사이의 거리를 Euclidean distance를 통해 가져온다. neighbors를 구할 때 사용한다.

C. Class DBScan – 핵심 함수들 구현

i. clustering() :

```
def clustering(self):
    cluster_id = 0
    cluster_size = []
    for i in range(len(data_origin)):
        if data_origin.loc[i, 'cluster'] is None:
            flag = self.set_label(i, cluster_id) # flag : outlier detection
            if flag is not False :
                cluster_size.append(flag)
                print('cluster #'+str(cluster_id)+' labeled')
                cluster_id += 1
    top_n_count = np.argsort(cluster_size)[-self.n:]
    print('Top n Cluster:\t', top_n_count)
    print('Each size:\t', np.sort(cluster_size)[-self.n:])
```

주어진 데이터 '순서'대로 cluster label이 None인 것을 set_label()을 통해 labeling한다. 이 때, outlier인 경우를 제외하곤 해당 데이터의 neighbors들도 모두 labeling된다. for loop이 끝나면 각 클러스터의 사이즈만 return 받은 array에서 top-n-cluster를 도출한다.

ii. set_label(id, cluster_id)

```
def set_label(self, id, cluster_id):
    neighbors = self.get_neighbors(id)

    # Outlier setting / Outlier label(-1) can be changed anytime
    if len(neighbors) < self.min_pts :
        data_origin.loc[id, 'cluster'] = -1
        return False

    i = 0
    # Grow neighbors
    while(i< len(neighbors)):
        data_origin.loc[neighbors[i], 'cluster'] = cluster_id # Cluster label 삽입
        if len(self.get_neighbors(neighbors[i])) >= self.min_pts:
            neighbors += self.get_neighbors(neighbors[i])
            neighbors = list(dict.fromkeys(neighbors)) # 중복제거
        i+=1

    # print(neighbors)
    return len(neighbors)
```

해당 id를 가진 데이터가 core인지 판별하고, 맞다면 neighbors를 점진적으로 늘려나간다. 이후 neighbor의 cluster label에 해당 cluster_id를 부여한다. neighbors를 구할때는 get_neighbors()를 이용한다

i. get_neighbors(id) :

```
def get_neighbors(self,id):
    x = data_origin.loc[id,'x']
    y = data_origin.loc[id,'y']

    # 검사할 dataset 크기 줄이기
    x_min = x - self.eps
    x_max = x + self.eps
    y_min = y -self.eps
    y_max = y + self.eps
    tmp = list(np.where((x_min <=data_origin.to_numpy()[:,1]) & (data_origin.to_numpy()[:,1] <= x_max) & (y_min <= data_origin.to_numpy()[:,2]) & (data_origin.to_numpy()[:,2] <= y_max)))

    # distance > eps 인거 빼기
    to_remove = []
    for candidate in tmp :
        to_remove = list(np.where(get_distance(id, candidate) > self.eps))
    tmp = np.delete(tmp, to_remove)

    return list(tmp)
```

각 id에 해당하는 neighbors를 return한다. 데이터셋의 크기가 클 경우 distance를 구하는 과정에서 많은 연산이 소요되기 때문에(sqrt, square), x-eps < data < x+eps, y-eps < data < y+eps 에 해당하는 부분만 떼어와 dataset을 줄였다. 이후 해당 dataset의 distance를 구해 eps보다 작은 것들만 return하였다.

ii. print_output() :

```
def print_output(self, top_n_label):
    print(data_origin)
    for i in range(len(top_n_label)):
        output = DataFrame(data_origin[data_origin['cluster']==top_n_label[i]].index)
        output_name = input_name.split(sep='.')[0] + '_cluster_' + str(i) + '.txt'
        output.to_csv(output_name, index=False, sep='\n', header=False)
```

결과를 각 txt에 출력한다.

3. Instruction for compiling

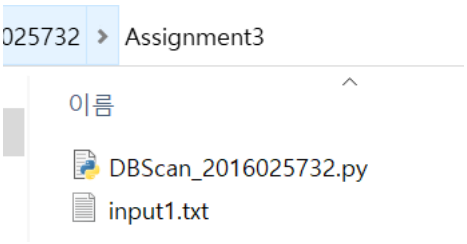
A. python 설치

B. Input 파일, DBScan_2016025732.py 같은 디렉토리내 위치

C. Execute the program with four arguments : input file, n, Eps, MinPts

4. Testing

A. 실행 전



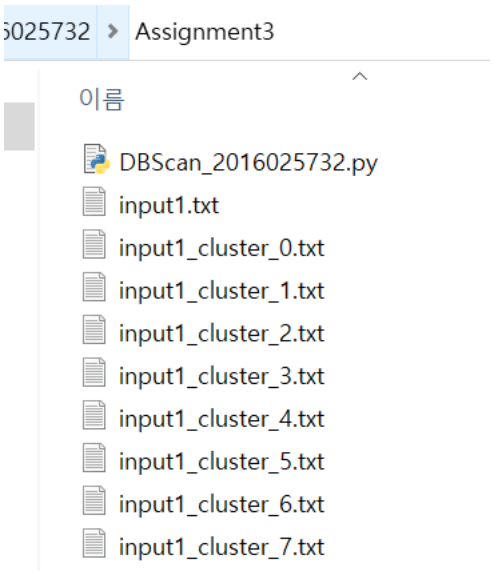
B. cmd 입력/출력

```
C:₩Users₩이영석₩Desktop₩git₩2021_ite4005_2016025732₩Assignment3>DBScan_2016025732.py input1.txt 8 15 22
cluster #0 labeled
cluster #1 labeled
cluster #2 labeled
cluster #3 labeled
cluster #4 labeled
cluster #5 labeled
cluster #6 labeled
cluster #7 labeled
cluster #8 labeled
cluster #9 labeled
cluster #10 labeled
Top n Cluster:   [7 8 5 3 2 0 1 4]
Each size:       [  34   34  179 1135 1458 1481 1593 1601]
      index          x           y  cluster
0         0   84.768997   33.368999        0
1         1  569.791016   55.458000        1
2         2  657.622986   47.035000        1
3         3  217.057007  362.065002        2
4         4  131.723999  353.368988        2
...     ...         ...         ...      ...
7995   7995  384.888000  273.808014        4
7996   7996  371.938995   70.299004        0
7997   7997   37.887001   79.575996        0
7998   7998  175.552994   76.314003        0
7999   7999  226.192001  115.615997        3

[8000 rows x 4 columns]
Clustering time :  76.28214168548584
```

input 1의 경우 1분대, input 2,3의 경우 10초대로 clustering 되었다.

C. 실행 후



D. 채점 프로그램 실행 결과

```
C:₩Users₩이영석₩Desktop₩test-3>PA3.exe input1
98.97691점
C:₩Users₩이영석₩Desktop₩test-3>PA3.exe input2
94.86023점
C:₩Users₩이영석₩Desktop₩test-3>PA3.exe input3
99.97736점
```