DBSCAN

for clustering

김상욱 교수님
2013002277 김재균
2017.05.20

1. 개요

군집화(Clustering)는 비슷한 개체끼리 한 그룹으로, 다른 개체는 다른 그룹으로 분류를 하여 하나의 집합으로 묶는 방법이다. 군집 분석은 각 개체의 유사성을 측정하여 유사성이 높은 개체들끼리 한 집단으로 분류하고, 군집에 속한 개체들의 유사성과 서로 다른 군집에 속한 개체 간의 상이성을 규명하는 통계 분석 방법이다. 군집화는 분류(Classification)와는 달리 비지도학습(unsupervised learning)이다.

2. 목적

이 프로젝트의 목적은 군집화의 알고리즘 중 밀도를 기준으로 군집화 하는 DBSCAN(Density-based spatial clustering of applications with noise) 방법을 이용하여 주어진 데이터를 바탕으로 군집 분석을 한다.

3. 개발환경

- Operating System: Windows 10 Pro
- Language: Python 3.6.4
- Source code editor: Visual Studio Code

4. 프로그램 구조 및 알고리즘

(1) 프로그램 실행 시 5가지 arguments(실행 파일, input data 파일, n, Eps, MinPts)를 입력한다. 여기서 n은 클러스터의 개수, Eps는 Epsilon이고 MinPts는 Core Point가 되기 위한 Eps 반경 내의 최소 neighborhood의 값에 대한 조건이다. Input 파일의 데이터를 [id, x_cor, y_cor] 형식에 맞게 가공 해주고 나머지 arguments들은 변수에 저장한다.

(2) Arguments에 대한 가공이 끝난 후, DBSCAN 알고리즘을 이용하여 군집화를 한다.

A. 먼저 ID 하나를 선택한 후, Eps 반경 내의 neighborhood들을 하나의 neighbors 집합으로 묶는다.

B. Neighbors의 개수가 주어진 minPts보다 크다면 이 ID는 Core Point로 간주하고 하나의 Cluster로 정의한다. 나머지 neighborhood들도 같은 Cluster Label로 정의한다.

C. 그 후, neighborhood를 하나씩 방문하며 Eps 반경 내의 neighborhood들을 Count 하고, 이 neighborhood 또한 Core Point라면 neighborhood의 neighborhood들을 모두 기존의 같은 Cluster로 정의한다.

D. 더 이상 방문할 neighborhood가 없다면 하나의 Cluster에 대해 군집화가 완성 되었고 방문하지 않은 다음 ID를 방문한다.

E. 방문하지 않은 ID가 없을 때까지 A, B, C, D 과정을 반복한다.

(4) 군집화 과정이 완성된 후, 주어진 클러스터 개수 n과 군집화 결과로부터 나온 개수를 비교하여 Optimization을 한다.

(5) 군집화된 각각의 클러스터를 Output File에 ID들을 출력한다.

5. 코드 설명

```
'''
Load input data and return in list
Data format: [ID, x_cor, y_cor]
'''
def load_data(argument):
    df = list()

    with open(argument, 'r') as f:
        input_data = f.read().split('\n')
        for data in input_data:
            data = data.split('\t')
            df.append(data)

    return df[:-1]
```

load_data()

- 주어진 input.txt 파일을 list의 형태로 저장한 후, 반환한다.

```
'''
Check if the data is neighbor::
If the distance between ID and compare ID less then eps, it is neighbor
'''
def is_neighbor(cor_1, cor_2, eps):
    ## Converting string to float
    x1 = float(cor_1[0])
    y1 = float(cor_1[1])
    x2 = float(cor_2[0])
    y2 = float(cor_2[1])

    distance = math.sqrt(math.pow(x1 - x2, 2) + math.pow(y1 - y2, 2))

    ## True if it is neighbor
    return distance < eps
```

is_neighbor()

- cor_1, cor_2 두 점의 거리를 계산하여 distance 변수에 저장한다.
- distance가 eps보다 작다면 True를 반환한다.

```
'''
Search neighbor based on input ID using eps
'''
def search_neighbor(total_data, ID, eps):
    neighbor_list = list()
    ## Only x_cor, y_cor is needed
    cor_1 = total_data[ID][1:]

    ## Search all data if it is neighbor or not
    for neighbor_id in range(len(total_data)):
        cor_2 = total_data[neighbor_id][1:]
        if is_neighbor(cor_1, cor_2, eps):
            ## If data is neighbor then append it to the neighbor list
            neighbor_list.append(neighbor_id)

    return neighbor_list
```

search_neighbor()

- 주어진 ID를 기준으로 eps 반경 내에 있는 모든 neighbor들을 찾는다.
- neighbor_list의 리스트에 neighbor들의 ID를 모두 저장한다

```
'''
Write data from each cluster label
'''
def write_data(input_name):
    input_name = input_name.replace('.txt', '')

    for label in range(len(set(cluster_labels))-1):
        file_name = input_name + '_cluster_' + str(label) + '.txt'

        ## Pull out the index numbers based on cluster labels
        ID_list = [i for i, j in enumerate(cluster_labels) if j == label]

        temp = ''
        for id in ID_list:
            temp += str(id) + '\n'
        with open(file_name, 'w') as f:
            f.write(temp)
```

write_data()

- 각각의 클러스터를 input#_cluster_#.txt 형식으로 파일 출력을 한다.
- 몇 개의 클러스터 Label이 있는지 확인하기 위해 enumerate 함수를 사용하였다.

```python
def classify_cluster(total_data, ID, label_type, eps, minPts):
    global cluster_labels

    ## Search neighbors
    neighbors = search_neighbor(total_data, ID, eps)

    ## Neighbor more than minPts -> Make cluster
    if len(neighbors) >= minPts:
        cluster_labels[ID] = label_type

        ## Give same cluster label type to neighbors
        for neighbor_id in neighbors:
            cluster_labels[neighbor_id] = label_type

        ## Loop all neighbors until there is no more core point
        while len(neighbors) > 0:
            neighbor_id = neighbors[0]
            ## Recursively expand cluster based on DFS method
            ## Search child neighbor
            child_list = search_neighbor(total_data, neighbor_id, eps)

            ## if child neighbor's count is more than minPts, it is core point
            if len(child_list) >= minPts:

                for i in range(len(child_list)):
                    index = child_list[i]
                    ## If child's neighbor not visited or is outlier
                    if cluster_labels[index] == None or cluster_labels[index] == -1:

                        ## Append it to neighbor list
                        neighbors.append(index)
                        cluster_labels[index] = label_type

            ## Go to next
            neighbors = neighbors[1:]

        return True

    ## Neighbor less than minPts
    else:
        cluster_labels[ID] = -1
        return False
```

classify_cluster()

- search_neighbor로 만들어진 neighbors가 minPts보다 크면 Core Point로 정의하고 클러스터를 만든다. 그리고 neighbors들도 같은 label_type 값으로 입력한다.
- neighbors들에 저장된 ID들을 다 방문할 때까지 while문을 이용하여 방문하면서 각자 Core Point인지 확인한다.
- Core Point이면 neighbors 리스트에 추가하고 그렇지 않으면 -1이라는 값을 넣어준다.

```python
if __name__ == '__main__':

    ## Load input data
    total_data = load_data(sys.argv[1])
    cluster_number = int(sys.argv[2])
    eps = int(sys.argv[3]) ## epsilon
    minPts = int(sys.argv[4])

    ## Cluster label starting from 0
    label_type = 0
    ## Initializing cluster labels with None
    cluster_labels = [None] * len(total_data)

    print("DBSCAN START:: \n")

    ## Loop every id in total data
    for ID in range(len(total_data)):
        ## If ID not visited then make cluster
        if cluster_labels[ID] == None:
            if classify_cluster(total_data, ID, label_type, eps, minPts):
                print("Cluster Label Type %s:: Classified..!" % label_type)
                label_type = label_type + 1

    ## Optimize cluster with given cluster number
    # optimization(cluster_number)

    print("Finished.")

    ## Write File
    write_data(sys.argv[1])
```

main 함수

- 주어진 arguments 중 input.txt는 가공하여 list로 반환하여 total_data 리스트에 저장하였다.나머지 n, eps, minPts는 각각의 변수에 저장하였다.
- 처음의 Cluster label을 0으로 정의하고 cluster_labels를 None으로 모두 초기화 시켜준 후, DBSCAN을 시작한다.
- 모든 ID를 반복하여 방문하면서 방문하지 않은 ID들은 classify_cluster 함수를 통해 군집화를 한다.
- 군집 분석이 끝난 후, write_data 함수를 통해 파일 출력을 한다.

6. 가이드 라인 및 실행 결과

- 가이드 라인

terminal(command) 명령어: python clustering.py input#.txt n eps minPts

- 실행 결과

```
C:\Users\Jkyun\Desktop\test1
λ PA3.exe input1
98.98721점
```

```
C:\Users\Jkyun\Desktop\test2
λ PA3.exe input2
94.83162점
```

```
C:\Users\Jkyun\Desktop\test3
λ PA3.exe input3
99.97736점
```

7. 에필로그

이번 프로젝트를 통해 군집화(Clustering)의 알고리즘 중 밀도(Density)를 이용하는 DBSCAN 방법을 통해 군집 방식을 배울 수 있었다. Optimization의 선택지가 여러 개가 존재하여 정답이 없어 어떤 방식이 최적화된 결과를 나올지 많은 고민을 하였다.

8. 참조

Data mining: concepts and techniques, Morgan Kaufmann