인공지능 과제 #2

Hierarchical Clustering Problem

2018008059 김은수

1. 코드 설명

(1) 각 좌표파일 마다 코드 실행방법

```
def run(plane: str):
    clus = Clustering(plane+".txt")

    with open(plane+'_output.txt', 'w') as f:
        f.write(str(clus.k_number))

    clus.single(plane+'_output.txt')
    clus.complete(plane+'_output.txt')
    clus.average(plane+'_output.txt')


if __name__ == '__main__':
    run('CoordinatePlane_1')
    run('CoordinatePlane_2')
    run('CoordinatePlane_3')
```

main문에 run함수를 적고, run함수의 인자로 입력 파일의 이름을 적는다. (입력 파일의 파일 형식을 제외한 이름만 적는다.)

run 함수를 통해 각 파일을 open하고, 각 파일에 대해 single clustering, complete clustering, average clustering이 진행된다.

(2) 각 좌표파일 마다 코드 동작

main에서 run함수를 부른다. run 함수에서 처음에 각 파일에 대해 class 객체를 만들어준다. 그 다음 output 파일을 만들어주고 해당 파일의 번호를 적는다. 만든 class 객체의 clustering 함수를 이용해서, single clustering, complete clustering, average clustering을 진행한다. 각 clustering의 함수에 output file을 인자로 주어서 output 파일에 출력을 할 수 있도록 한다.

2. 함수 설명

(1) 함수요약

Clustering 클래스 메서드 설명

Method	Description
Cosine_similarity	각 node 쌍의 cosine similarity를 구해서 matrix 형태로 저장한다. 저장하는 배열은 similarity_list다.
Single	Single clustering을 진행한다.
Complete	Complete clustering을 진행한다.
average	Average clustering을 진행한다.

(2) 함수 설명

- 초기화 함수

```
class Clustering:
    k_number = 0
    node_number = 0
    node = None
    node_list = []
    similarity_list =[]

    def __init__(self, filename):
        self.node_list.clear()
        self.similarity_list.clear()
        with open(filename, 'r') as f:
            self.k_number, self.node_number = [int(x) for x in f.readline().strip().split(' ')]
            self.similarity_list =[[0 for col in range(self.node_number)] for row in range(self.node_number)]
            for line in f:
                self.node= tuple(int(x) for x in line.strip().split(','))
                self.node_list.append(self.node)
```

File을 읽으면서 좌표평면의 번호는 클래스 변수 k_number에, 노드 개수는 클래스 변수 node_number에 할당한다. 다음으로 노드 개수만큼의 행과 열을 가지는 similarity_list를 만든다. Similarity list에는 각 노드 쌍의 cosine similarity가 할당된다. 다음으로 node를 tuple 형태로 입력 받은 뒤 node_list에 넣어준다.

- Cosine similarity

```
def cosine_similarity(self):
    for i in range(0,self.node_number):
        for j in range(0,self.node_number):
            mole = self.node_list[i][0]*self.node_list[j][0]+self.node_list[i][1]*self.node_list[j][1]
            deno = self.node_list[i][0]*self.node_list[i][0]+self.node_list[i][1]*self.node_list[i][1]
            deno1 = math.sqrt(float(deno))
            deno = self.node_list[j][0] * self.node_list[j][0] + self.node_list[j][1] * self.node_list[j][1]
            deno1 *=math.sqrt(float(deno))
            similarity = mole/deno1
            self.similarity_list[i][j] = similarity

    for i in range(0,self.node_number):
        self.similarity_list[i][i] = -100
```

전체 node 쌍끼리 cosine similarity를 구해줘서 해당되는 similarity_list 위치에 값을 넣어준다. cosine similarity 함수가 끝나면 각 노드끼리의 similarity를 담은 matrix인 similarity list가 나온다. Similarity list의 행과 열의 노드 순서는 node list에 저장된 노드 순서와 같다. 이때, 자신 노드끼리 similarity를 구하면 1이 나오지만, 자신의 노드끼리 합치는 경우는 없기 때문에 해당 부분은 -100으로 초기화해준다.

- Single

```
self.cosine_similarity()

group = [[] for row in range(self.node_number)]

for i in range(0,self.node_number):
    group[i].append(self.node_list[i])
```

우선 cosine_similarity()를 통해 similarity_list를 만들어준다. 또한, single cluster에 의해서 group이 되는 노드들을 저장하기 위해 group 배열을 만든다. Group list는 이차원 배열이다. 이 group list의 각 0번째 열에는 node list의 순서와 같이 node가 할당되어 있다. Cluster로 합치게 되면 뒤 순서에 있는 node가 더 앞 순서에 있는 node의 group list행에 append 되는 구조다.

이 single 함수의 number 변수는 초기 similarity_list의 크기이다. 그리고 while문이 돌면서 node가 합쳐지고 number는 1씩 감소하며, number가 3이 되면 이 while문이 종료된다. 그 이유는 다음과 같다. while문을 돌면서 similarity_list에서 similarity 값이 가장 높은 두 노드를 합친다. 그렇게 되면 while문을 돌 때마다 similarity_list의 크기는 1씩 작아지게 되고, 마지막으로 3개의 cluster로 구분될 때 matrix의 크기는 3이 된다. 따라서 matrix의 크기가 3이 됐다는 말은 cluster가 3개 생겼다는 얘기 이므로 while문을 멈추게 된다.

```
while number != 3:
    maxv = max(map(max,self.similarity_list))

    for i in range(0,self.node_number):
        for j in range(0, self.node_number):
            if self.similarity_list[i][j] == maxv:
                iv = i
                jv = j
                break

    if iv > jv: #iv가 항상 작다.
        tmp = iv
        iv = jv
        jv = tmp

    number -= 1
```

While문의 동작은 다음과 같다. Similarity_list에서 가장 큰 값을 구한다. 그리고 가장 높은 similarity를 가지는 node쌍을 찾기 위해 2중 for문을 돌린다. 2중 for문을 통해 node 쌍을 찾고 그 노드를 나타내는 index를 iv, jv로 나타낸다. 즉, node list의 iv번째와 jv번째 node가 합쳐진다. 여기서는 항상 node list에서 앞 순서에 있는 node group에 뒤 순서 node를 append할 것이므로 iv 와 jv 순서를 비교해서 iv가 더 작은 index를 가지도록 해준다.

```
for i in range(0,self.node_number):
    self.similarity_list[iv][i] = max(self.similarity_list[iv][i], self.similarity_list[jv][i])
    self.similarity_list[i][iv] = self.similarity_list[iv][i]
    self.similarity_list[jv][i] = -100
    self.similarity_list[i][jv] = -100
    self.similarity_list[i][i] = -100
```

다음으로 similarity_list를 변경해줘야 한다. Similarity_list에서 합쳐진 jv index를 가지는 node의 열과 행을 -100으로 바꿔준다. 이는 del해서 해당 행과 열을 없앤 것과 같은 효과이다. 실제로 없애지 않은 이유는 node_list나, group 배열과의 node index의 일관성을 주기 위해서다. 그리고 각 node들 마다 iv행과 jv행을 비교해서 더 큰 값을 similarity list에 업데이트해준다.

```
for i in range(len(group[jv])):
    group[iv].append(group[jv][i])

group[jv]=[]
similarity_ = max(map(max, self.similarity_list))
similarity_level.append(similarity_)
```

다음으로 group에 node를 합쳐 준다. 그리고 합쳐진 node에 해당하는 group은 빈 배열로 만들어준다. 그리고 similarity level 배열에 현재 가장 높은 similarity를 넣어준다. Similarity level은 합쳐진 상황에서의 가장 큰 similarity를 나타낸다.

```
for n in range(self.node_number):
    if not group[n]:
        continue
    else:

        f.write('[')
        for x in range(1,len(group[n])):
            f.write(str(group[n][x]))
            f.write(',')
            lst.append(group[n][x])
        f.write(str(self.node_list[n]))
        lst.append(self.node_list[n])
        f.write('] ')

for n in range(self.node_number):
    if self.node_list[n] not in lst:
        f.write('[')
        f.write(str(self.node_list[n]))
        f.write('] ')
```

Output 파일에는 다음과 같이 write한다. 만약 group이 비워져 있다면 그 group은 skip한다. Group의 요소가 있다면 group내 요소를 출력시켜준다. 요소 출력이 좀 복잡한데 이는 output 파일에 보기 편하게 적기 위함이다. 0말고 1부터 적은 이유는 마지막 요소를 출력하고도 ','가 출력되지 않기 하기 위해서이다. 그리고 0에 해당하는 요소를 출력해준다. 1부터 출력을 했기 때문에 길이가 1인 group은 나오지 않기 때문에 나오지 않았던 요소를 따로 출력하게 했다.

```
  f.write('\nspan: ')
  f.write(str(similarity_level[-2])+', ')
  f.write(str(similarity_level[-1]))

  f.close()
```

다음으로 similarity_level list의 마지막 요소와 그 전 요소를 출력해준다. 그 이유는 마지막 요소는 node들이 3개의 그룹으로 grouping이 완료된 이후의 similarity값이다. Similarity_level의 마지막에서 두번째 요소는 cluster가 4개일때의 similarity값이다. 따라서 cluster가 3개가 되려면 similarity가 이 사이에 있어야하기 때문에 span으로 이 두개의 값이 적절하다. 따라서 이 두 값을 출력해준다.

- Complete

```
for i in range(0,self.node_number):
    self.similarity_list[iv][i] = min(self.similarity_list[iv][i], self.similarity_list[jv][i])
    self.similarity_list[i][iv] = self.similarity_list[iv][i]
    self.similarity_list[jv][i] = -100
    self.similarity_list[i][jv] = -100
    self.similarity_list[i][i] = -100
```

Single 함수와 동일하지만 같지만 similarity_list를 업데이트 시킬 때 max값이 아니라 min값을 사용한다.

- Average

```
for i in range(0,self.node_number):
    self.similarity_list[iv][i] = (self.similarity_list[iv][i] + self.similarity_list[jv][i])/2
    self.similarity_list[i][iv] = self.similarity_list[iv][i]
    self.similarity_list[jv][i] = -100
    self.similarity_list[i][jv] = -100
    self.similarity_list[i][i] = -100
```

Single 함수와 모든 게 같지만 similarity_list를 업데이트 시킬 때 두 값의 평균을 이용한다.

3. 실험 결과 설명

- CoordinatePlane_1

```
1
---
single
cluster: [(1, 3),(1, 2)] [(-1, 2)] [(-20, -15),(-10, 1)]
span: 0.736327520755392, 0.7071067811865475
---
complete
cluster: [(1, 3),(1, 2)] [(-1, 2)] [(-20, -15),(-10, 1)]
span: 0.736327520755392, 0.5999999999999999
---
average
cluster: [(1, 3),(1, 2)] [(-1, 2)] [(-20, -15),(-10, 1)]
span: 0.736327520755392, 0.6535533905932737
```

- coordinatePlane_2

```
2
---
single
cluster: [(-438, 13),(-963, 752),(-752, 600),(-314, 461),(-445, 621),(-683, 888),(-519, 529),(-419, 854),
(-467, 178),(-504, -332),(-645, -450),(-312, -179),(-696, -409),(-546, -800),(-229, -469),(-347, -750),
(-98, -284),(-896, -120)] [(625, 384),(815, 144),(787, -17),(415, -953),(53, -169),(830, -954),(734, -
847),(768, -265),(388, -251),(602, 337)] [(512, 724),(444, 974)]
span: 0.9257682206228005, 0.9193678534664724
---
complete
cluster: [(-438, 13),(-504, -332),(-645, -450),(-312, -179),(-696, -409),(-546, -800),(-229, -469),(-347,
-750),(-98, -284),(-963, 752),(-752, 600),(-519, 529),(-467, 178),(-314, 461),(-445, 621),(-683, 888),
(-419, 854),(-896, -120)] [(625, 384),(815, 144),(787, -17),(444, 974),(512, 724),(602, 337)] [(53, -
169),(830, -954),(734, -847),(768, -265),(388, -251),(415, -953)]
span: -0.7049796316762562, -0.7441029700094789
---
average
cluster: [(-438, 13),(-467, 178),(-963, 752),(-752, 600),(-519, 529),(-314, 461),(-445, 621),(-683, 888),
(-419, 854),(-504, -332),(-645, -450),(-312, -179),(-696, -409),(-546, -800),(-229, -469),(-347, -750),
(-98, -284),(-896, -120)] [(625, 384),(815, 144),(787, -17),(444, 974),(512, 724),(602, 337)] [(53, -
169),(830, -954),(734, -847),(768, -265),(388, -251),(415, -953)]
span: 0.20423030410835533, 0.1270085714175628
```

- coordinatePlane_3

```
3
---
single
cluster: [(-306, -585),(-539, -816),(-525, -728),(-172, -221),(-460, -494),(-27, -76),(-285, -797),(-88, -442),(-188, -733),(9, -733),(-5, -599),(44, -791),(-200, -49),(-411, -97),(-788, -324),(-363, -245),(-698, -458),(-885, -34),(-437, 164),(-630, 113),(-556, 339),(-736, 475),(-387, 568),(-470, 571),(-899, 908),(859, 951),(664, 769),(581, 565),(885, 672),(520, 445),(916, 575),(400, 228),(994, 557),(83, 234),(378, 904),(409, 903),(194, 379),(-49, 823),(-61, 920),(14, 780),(15, 133),(-110, 421),(-249, 854),(-304, -539)] [(788, -290),(345, -163),(785, -475),(975, -577),(639, -250)] [(507, -855)]
span: 0.950986159367771, 0.8871449358866629
---
complete
cluster: [(-306, -585),(-460, -494),(-539, -816),(-525, -728),(-172, -221),(-363, -245),(-698, -458),(9, -733),(-5, -599),(44, -791),(-27, -76),(-285, -797),(-88, -442),(-188, -733),(639, -250),(788, -290),(345, -163),(785, -475),(975, -577),(507, -855),(-304, -539)] [(-411, -97),(-788, -324),(-885, -34),(-437, 164),(-630, 113),(-556, 339),(-736, 475),(-387, 568),(-470, 571),(-899, 908),(-200, -49)] [(664, 769),(581, 565),(916, 575),(400, 228),(994, 557),(885, 672),(520, 445),(83, 234),(378, 904),(409, 903),(194, 379),(-49, 823),(-61, 920),(14, 780),(15, 133),(-110, 421),(-249, 854),(859, 951)]
span: -0.5951602722234179, -0.9927284861861074
---
average
cluster: [(-306, -585),(-539, -816),(-525, -728),(-172, -221),(-460, -494),(9, -733),(-5, -599),(44, -791),(-27, -76),(-285, -797),(-88, -442),(-188, -733),(639, -250),(788, -290),(345, -163),(785, -475),(975, -577),(507, -855),(-304, -539)] [(-411, -97),(-788, -324),(-363, -245),(-698, -458),(-885, -34),(-437, 164),(-630, 113),(-556, 339),(-736, 475),(-387, 568),(-470, 571),(-899, 908),(-200, -49)] [(664, 769),(581, 565),(885, 672),(520, 445),(916, 575),(400, 228),(994, 557),(83, 234),(378, 904),(409, 903),(194, 379),(-49, 823),(-61, 920),(14, 780),(15, 133),(-110, 421),(-249, 854),(859, 951)]
span: 0.3253733303427031, -0.26897838169923316
```

모두 3개의 cluster로 나눠진 것을 볼 수 있다. Single의 경우 군집 간 similarity가 max, Complete의 경우 군집 간 similarity가 min, average의 경우 군집 간 similarity가 average로 계산되기 때문에 similarity가 complete < average < single 임을 알 수 있다.

또한, coordinatePlane 2에서 single은 cluster 요소 개수가 2,10,18로 나뉘며, complete는 18,6,6, average는 18,6,6으로 나뉜다. coordinatePlane 3에서 single은 1,5,44, complete는 21,11,18, average는 19,13,18로 나뉜다. 따라서 single보다 complete가, complete보다 average가 balance있게 grouping된다는 것을 알 수 있다.