Assignment 2

Hierarchical Clustering Problem

2020-2 인공지능
최용석교수님

컴퓨터소프트웨어학부
2015004911 임성준

● 코드 및 함수 설명

0. 과제 이해

1) 3가지 좌표 평면의 점들에 데이터가 주어진다.
2) 이 데이터들을 각각 3개의 군집으로 묶일 수 있도록 하는 similarity level을 구해야 한다.
3) 점들 사이의 similarity는 Cosine Similarity로 정의한다.
4) 좌표 평면상의 점들을 single-link, complete-link, group average-link cluetering 총 세가지 방법으로, 각 방법마다의 similarity level 범위(span이라고 한다.)를 구한다.

1. 기본 동작 설명

1) 읽을 파일을 file_read 함수로 불러와 정보를 입력한다.
2) Clustering을 수행하는 clustering함수를 호출한다.
3) 각 좌표간의 cosine similarity값을 배열에 입력한다. 이때의 cosine similarity값은 cosine_similarity 함수로 구한다.
4) single_link_clustering, complete_link_clustering, average_link_clustering 함수 세가지를 모두 차례대로 호출하며 file_write로 output 파일에 쓴다.

2. 함수 및 코드 설명

1) file_read & file_write

```python
#file read
def file_read(file_name):
    file = open(file_name, "r")
    k, n = map(int, file.readline().split())
    xy = [[0 for x in range(2)] for y in range(n)]

    index = 0
    while True:
        line = file.readline().strip("\n")
        if not line: break
        xy[index] = list(map(int, line.split(",")))
        index += 1

    file_name = file_name[:-4]
    file.close()

    return xy, file_name

#file write
def file_write(file_name, clusters, span, k):
    file = open(file_name + "_output.txt", "a")
    file.write("---\n" + k + "\n" + "clusters: ")
    for i in range(len(clusters)):
        file.write("[" + "," .join(map(str, clusters[i])) + "]")
    file.write("\nspan: {}, {}\n" .format(span[0], span[1]))

    file.close()
```

file_read : 파일에서 주어진 data를 입력받는다. 그 후, data를 넘겨주며 실행을 종료한다.
file_write : file_name에 맞는 output 파일을 불러온다. 그 후, clusters 리스트에 저장된 좌표의 군집을 출력하고, span 리스트에 저장된 similarity level 범위를 출력한다. 모든 수행이 종료되면 파일을 닫는다.

2) cosine_similarity

```python
#cosine similarity
def cosine_similarity(A, B):
    multiple_xx = 0
    multiple_xy = 0
    multiple_yy = 0
    for i in range(len(A)):
        x = A[i]
        y = B[i]
        multiple_xx += x*x
        multiple_xy += x*y
        multiple_yy += y*y

    return multiple_xy / math.sqrt(multiple_xx * multiple_yy)
```

$$similarity(A,B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

A와 B 좌표 사이의 cosine similarity를 구한다. 제곱근의 경우 math 라이브러리를 import해, 내장 sqrt 함수로 구한다.

3) single_link_clustering

```
#single link clustering
def single_link_clustering(xy):
    arr = copy.deepcopy(xy)
    length = len(arr)
    span = [-2, -2]
    clusters = DisjointSet(length)

    while clusters.length() >= 3:
        max_span, obj1, obj2 = 0, 0, 0

        for i in range(len(arr)):
            for j in range(i+1, len(arr)):
                if max_span < arr[i][j]:
                    max_span = arr[i][j]
                    obj1, obj2 = i, j
        span[0] = span[1]
        span[1] = max_span

        if clusters.length() == 3: break
        clusters.union(obj1, obj2)

        for i in range(len(arr)):
            if obj1 != i:
                arr[obj1][i] = max(arr[obj1][i], arr[obj2][i])
                arr[i][obj1] = arr[obj1][i]
            arr[obj2][i] = -2
            arr[i][obj2] = -2

    return clusters, span
```

clusters를 disjoint set으로 선언한다. 군집이 3개 이상이면 함수를 종료한다. similarity 값을 while 문으로 비교하며 값이 변경될 때마다 span 리스트에 적용시킨다.
single link clustering은 각 군집별로 best case를 찾는 것이므로 max값을 적용해서 구한다.

4) complete_link_clustering

```
#complete link clustering
def complete_link_clustering(xy):
    arr = copy.deepcopy(xy)
    length = len(arr)
    span = [-2, -2]
    clusters = DisjointSet(length)

    while clusters.length() >= 3:
        max_span, obj1, obj2 = -2, 0, 0

        for i in range(len(arr)):
            for j in range(i+1, len(arr)):
                if max_span < arr[i][j]:
                    max_span = arr[i][j]
                    obj1, obj2 = i, j
        span[0] = span[1]
        span[1] = max_span

        if clusters.length() == 3: break
        clusters.union(obj1, obj2)

        arr[obj1][obj2] = -2
        arr[obj2][obj1] = -2

        for i in range(len(arr)):
            if obj1 != i:
                arr[obj1][i] = min(arr[obj1][i], arr[obj2][i])
                arr[i][obj1] = arr[obj1][i]
            arr[obj2][i] = -2
            arr[i][obj2] = -2

    return clusters, span
```

clusters를 disjoint set으로 선언한다. 군집이 3개 이상이면 함수를 종료한다. similarity 값을 while 문으로 비교하며 값이 변경될 때마다 span 리스트에 적용시킨다.

complete link clustering은 각 군집별로 worst case를 찾은 다음, 그 중 best case를 찾는 것이므로 min값을 적용해서 구한다.

5) average_link_clustering

```
#group average link clustering
def average_link_clustering(xy):
    arr = [[-2 for x in range(len(xy))] for y in range(len(xy))]
    for i in range(len(xy)):
        for j in range(len(xy)):
            if i == j: continue
            arr[i][j] = cosine_similarity(xy[i], xy[j])

    length = len(xy)
    span = [-2, -2]
    clusters = DisjointSet(length)

    while clusters.length() >= 3:
        max_span, obj1, obj2 = -2, 0, 0

        for i in range(len(arr)):
            for j in range(i+1, len(arr)):
                if max_span < arr[i][j]:
                    max_span = arr[i][j]
                    obj1, obj2 = i, j
        span[0] = span[1]
        span[1] = max_span

        if clusters.length() == 3: break
        clusters.union(obj1, obj2)

        arr[obj1][obj2] = -2
        arr[obj2][obj1] = -2

        for i in range(len(arr)):
            if obj1 != i and arr[obj1][i] != -2:
                if clusters.find(obj1) == clusters.find(i): continue

                list1 = clusters.get_list(obj1)
                list2 = clusters.get_list(i)
                arr[obj1][i] = 0

                for j in list1:
                    for k in list2:
                        arr[obj1][i] += cosine_similarity(xy[j], xy[k])
                arr[obj1][i] = arr[obj1][i] / (len(list1) * len(list2))
                arr[i][obj1] = arr[obj1][i]

            arr[obj2][i] = -2
            arr[i][obj2] = -2

    return clusters, span
```

clusters를 disjoint set으로 선언한다. 군집이 3개 이상이면 함수를 종료한다. similarity 값을 while 문으로 비교하며 값이 변경될 때마다 span 리스트에 적용시킨다.
average link clustering은 각 군집별로 평균값 case를 찾는 것이므로 평균값을 적용해서 구한다

6) make_xy

```python
#make_xy
def make_xy(clusters, xy):
    obj1, obj2, obj3, count = -1, -1, -1, 0

    for i in set(clusters.data):
        if count == 0: obj1 = i
        elif count == 1: obj2 = i
        elif count == 2: obj3 = i
        count += 1

    arr = [[], [], []]
    for i in range(len(clusters.data)):
        index = 0
        if clusters.data[i] == obj1: index = 0
        elif clusters.data[i] == obj2: index = 1
        elif clusters.data[i] == obj3: index = 2
        arr[index].append((xy[i][0], xy[i][1]))

    return arr
```

disjoint set인 clusters를 3개 군집의 리스트로 변환시키는 함수이다.
리스트에 입력이 완료되면 리스트를 반환한다.

7) clustering

```python
#clustering
def clustering(xy, file_name):
    file = open(file_name + "_output.txt", "a")
    k = file_name[file_name.find('_') + 1:]
    file.write(k + '\n')
    file.close()

    arr = [[-2 for x in range(len(xy))] for y in range(len(xy))]
    for i in range(len(xy)):
        for j in range(len(xy)):
            if i == j: continue
            arr[i][j] = cosine_similarity(xy[i], xy[j])

    cluster, span = single_link_clustering(arr)
    clusters = make_xy(cluster, xy)
    file_write(file_name, clusters, span, "single")

    cluster, span = complete_link_clustering(arr)
    clusters = make_xy(cluster, xy)
    file_write(file_name, clusters, span, "complete")

    cluster, span = average_link_clustering(xy)
    clusters = make_xy(cluster, xy)
    file_write(file_name, clusters, span, "average")
```

output 파일을 open한 후, write해준다.
각 함수를 실행시키며 결과값을 file_write로 써준다.

average_link_clustering의 경우, similarity를 저장하는 함수인 arr를 함수 내부에서 선언해서 사용하므로, arr이 아닌 xy를 인자값으로 전달한다.

3. 실험 결과 설명

```
1
---
single
clusters: [(1, 2),(1, 3)][(-1, 2)][(-10, 1),(-20, -15)]
span: 0.736327520755392, 0.7071067811865475
---
complete
clusters: [(1, 2),(1, 3)][(-1, 2)][(-10, 1),(-20, -15)]
span: 0.736327520755392, 0.6
---
average
clusters: [(1, 2),(1, 3)][(-1, 2)][(-10, 1),(-20, -15)]
span: 0.736327520755392, 0.6535533905932738
```

```
2
---
single
clusters: [(-896, -120),(-438, 13),(-504, -332),(-963, 752),(-314, 461),(-546, -800),(-229, -469),(-752, 600),
(-312, -179),(-467, 178),(-347, -750),(-645, -450),(-683, 888),(-98, -284),(-445, 621),(-519, 529),(-696, -409),
(-419, 854)][(602, 337),(815, 144),(415, -953),(625, 384),(53, -169),(787, -17),(830, -954),(768, -265),(734,
-847),(388, -251)][(444, 974),(512, 724)]
span: 0.925768220622805, 0.9193678534664724
---
complete
clusters: [(-896, -120),(-438, 13),(-504, -332),(-963, 752),(-314, 461),(-546, -800),(-229, -469),(-752, 600),
(-312, -179),(-467, 178),(-347, -750),(-645, -450),(-683, 888),(-98, -284),(-445, 621),(-519, 529),(-696, -409),
(-419, 854)][(415, -953),(53, -169),(830, -954),(768, -265),(734, -847),(388, -251)][(602, 337),(815, 144),(625,
384),(787, -17),(444, 974),(512, 724)]
span: -0.7049796316762562, -0.7441029700094789
---
average
clusters: [(-896, -120),(-438, 13),(-963, 752),(-314, 461),(-752, 600),(-467, 178),(-683, 888),(-445, 621),(-519,
529),(-419, 854)][(-504, -332),(-546, -800),(-229, -469),(-312, -179),(-347, -750),(-645, -450),(-98, -284),(-696,
-409)][(602, 337),(815, 144),(415, -953),(625, 384),(53, -169),(787, -17),(830, -954),(768, -265),(444, 974),(512,
724),(734, -847),(388, -251)]
span: 0.15424743167763194, 0.07954467761968255
```

```
3
---
single
clusters: [(-304, -539),(-200, -49),(9, -733),(859, 951),(83, 234),(378, 904),(581, 565),(-437, 164),(-788, -324),
(-387, 568),(-363, -245),(916, 575),(885, 672),(400, 228),(664, 769),(-49, 823),(520, 445),(-61, 920),(-460, -494),
(409, 903),(-539, -816),(-899, 908),(-27, -76),(-411, -97),(14, 780),(-556, 339),(44, -791),(194, 379),(-525, -728),
(-110, 421),(15, 133),(-306, -585),(-5, -599),(-88, -442),(-470, 571),(-249, 854),(994, 557),(-885, -34),(-736, 475),
(-285, -797),(-630, 113),(-188, -733),(-172, -221),(-698, -458)][(639, -250),(345, -163),(785, -475),(975, -577),(788,
-290)][(507, -855)]
span: 0.950986159367771, 0.887144935886663
---
complete
clusters: [(-304, -539),(639, -250),(9, -733),(-363, -245),(345, -163),(507, -855),(785, -475),(-460, -494),(-539,
-816),(-27, -76),(44, -791),(-525, -728),(-306, -585),(-5, -599),(-88, -442),(-285, -797),(-188, -733),(975, -577),
(-172, -221),(788, -290),(-698, -458)][(-200, -49),(-437, 164),(-788, -324),(-387, 568),(-899, 908),(-411, -97),(-556,
339),(-470, 571),(-885, -34),(-736, 475),(-630, 113)][(859, 951),(83, 234),(378, 904),(581, 565),(916, 575),(885,
672),(400, 228),(664, 769),(-49, 823),(520, 445),(-61, 920),(409, 903),(14, 780),(194, 379),(-110, 421),(15, 133),
(-249, 854),(994, 557)]
span: -0.5951602722234179, -0.9927284861861074
---
average
clusters: [(-304, -539),(-200, -49),(9, -733),(-788, -324),(-363, -245),(-460, -494),(-539, -816),(-27, -76),(-411,
-97),(44, -791),(-525, -728),(-306, -585),(-5, -599),(-88, -442),(-885, -34),(-285, -797),(-188, -733),(-172, -221),
(-698, -458)][(639, -250),(345, -163),(507, -855),(785, -475),(975, -577),(788, -290)][(859, 951),(83, 234),(378,
904),(581, 565),(-437, 164),(-387, 568),(916, 575),(885, 672),(400, 228),(664, 769),(-49, 823),(520, 445),(-61, 920),
(409, 903),(-899, 908),(14, 780),(-556, 339),(194, 379),(-110, 421),(15, 133),(-470, 571),(-249, 854),(994, 557),
(-736, 475),(-630, 113)]
span: 0.13312808488922226, -0.07281332932132702
```

처음엔 좌표 평면의 번호가 쓰여있다.
밑으로는 각 함수별 결과값 (clusters, span)이 쓰여져있다.