**Data tools Assignment 4**

Q1]

a)

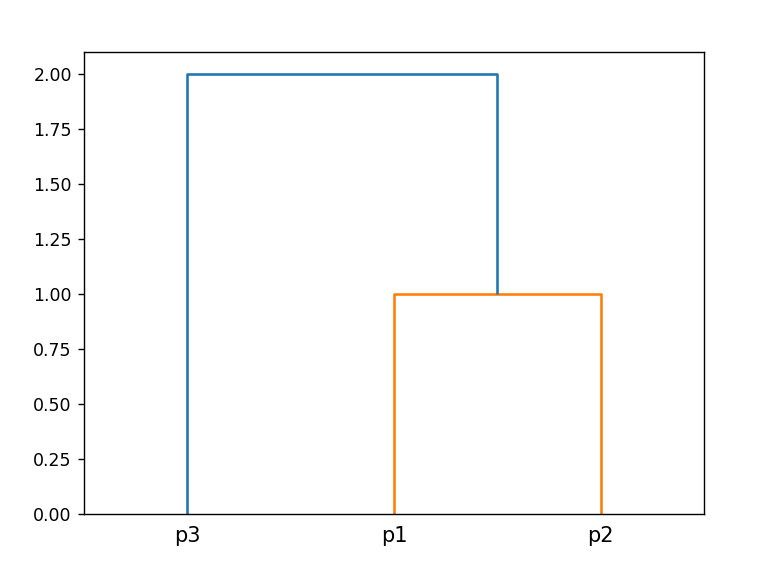
we assume our objects are (p1,p2,p3)

First iteration: the closest objects are 0,1 as they have the smallest dissimilarity = 1

Clusters are {p1,p2},{p3}

Second iteration: we take the minimum distance between the cluster {p1,p2} and p3 which is 4

Final clusters are {{p1,p2},p3}



b)

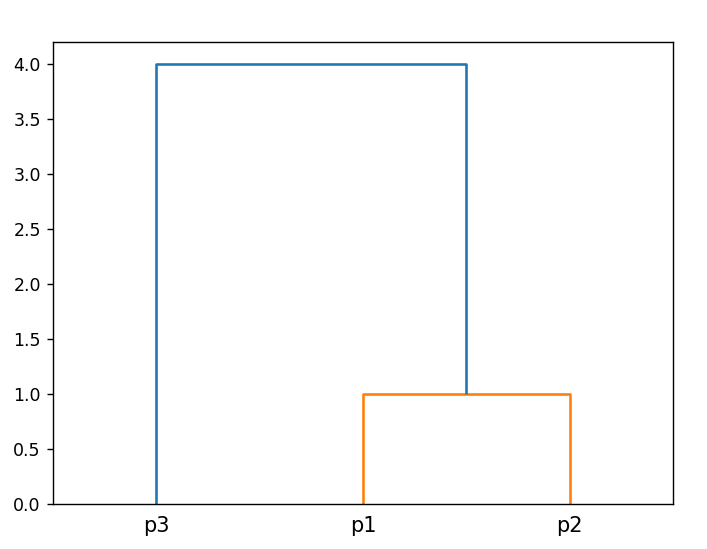
we assume our objects are (p1,p2,p3)

First iteration: the closest objects are p1,p2 as they have the smallest dissimilarity = 1

Clusters are {p1,p2},{p3}

Second iteration: we take the maximum distance between the cluster {p1,p2} and p3 which is 4

Final clusters are {{p1,p2},p3}



c)

**from** cmath **import** inf

**import** numpy **as** np

**import** pandas **as** pd

**import** matplotlib.pyplot **as** plt

**from** scipy.cluster.hierarchy **import** dendrogram

**from** scipy.spatial.distance **import** cdist

# to get each element in a cluster to compare ditances between each combination of clusters

***def*** flattenCluster(*c*):

    flat **=** []

**if** *type*(c) **!=** *tuple*:

        c **=** (c,)

**for** x **in** c:

**if** *type*(x) **==** *tuple*:

            flat.extend(flattenCluster(x))

**else**:

            flat.append(x)

**return** *tuple*(flat)

# calculate minimum ditances between clusters

***def*** clusterD(*c1*,*c2*,*d*):

**if** *type*(c1) **!=** *tuple*:

        c1 **=** (c1,)

**if** *type*(c2) **!=** *tuple*:

        c2 **=** (c2,)

    c1 **=** flattenCluster(c1)

    c2 **=** flattenCluster(c2)

# we start with distance = infinity and try to minimize it

# if we wanted to make complete linkage we would start at distance = 0 and try to maximize it

    distance **=** inf

**for** i **in** c1:

**for** j **in** c2:

**if** d.loc[i,j]**<**distance:

                distance **=** d.loc[i,j]

**return** distance

# dissimilarity matrix

d **=** pd.DataFrame([[0,1,4],[1,0,2],[4,2,0]])

# linkage matrix

l **=** []

index **=** []

# we give each cluster an index to construct linkage matrix

all\_indexes **=** {}

# giving points names

**for** x **in** range(1,len(d)**+**1):

    index.append(***f***'p{x}')

    all\_indexes[***f***'p{x}'] **=** x**-**1

d.index **=** index

d.columns **=** index

# setting diagonal with infinity so we can get minimum distance that is not between a point with itself

d[d**==**0]**=**inf

# saving every iteration and counting them

iterations **=** [d]

it**=**0

**while** len(iterations[**-**1])**>**1 **and** it **<** 10:

    d2 **=** iterations[**-**1]

    mini **=** np.min(d2.values)

    # indexes where the minimum distance is

    i **=** np.where(d2 **==** mini)

    clustered **=** []

    new\_clusters **=** []

    # merge clusters that has minimum distances if they have not been merged yet

**for** x **in** range(len(i[0])):

**if** i[0][x]**>**i[1][x] **and** i[0][x] **not** **in** clustered **and** i[1][x] **not** **in** clustered:

            i0 **=** d2.index[i[0][x]]

            i1 **=** d2.index[i[1][x]]

            new\_clusters.append((i1,i0))

            all\_indexes[(i1,i0)]**=**len(all\_indexes.keys())

            # add a row in linkage matrix that contains the index of first cluster and second, the distance between them, and the total number of points in both of them

            l.append([

*float*(all\_indexes[i0]),

*float*(all\_indexes[i1]),

*float*(clusterD(i0,i1,d)),

*float*(len(flattenCluster(i0))**+**len(flattenCluster(i1)))

                ])

            clustered.append(i[0][x])

            clustered.append(i[1][x])

    # add the unmerged clusters

**for** x,\_ **in** enumerate(d2.index):

**if** x **not** **in** clustered:

            new\_clusters.append(d2.index[x])

            clustered.append(x)

    iterations.append(

        pd.DataFrame(

            np.zeros(

                (len(new\_clusters),len(new\_clusters))

                )

            )

        )

    iterations[**-**1].index **=** new\_clusters

    iterations[**-**1].columns **=** new\_clusters

    # calculate distances after mergin

**for** i,ii **in** enumerate(iterations[**-**1].index):

**for** j,jj **in** enumerate(iterations[**-**1].index):

**if** i **==** j:

                iterations[**-**1].iloc[i,j] **=** inf

**else**:

                iterations[**-**1].iloc[i,j] **=** clusterD(ii,jj,d)

    print('-'**\***13,***f***"Iteration: {it**+**1}",'-'**\***13)

    print(iterations[**-**1])

    it**+=**1

print('-'**\***50)

c **=** []

print("Clusters: ",*end***=**'')

**for** x **in** iterations[**-**1].index:

    c.append(x)

print(c)

dendrogram(np.array(l),*labels***=**d.index)

plt.show()

d) to get the cluster of object 3 we calculate minimum distance between centroids which is 2. The 3rd object is in cluster c2

e) c1 contains only x1 so new cluster will be the same = x1

to calculate new cluster for c2

new clusters 🡪 c1 = [1,3] , c2 = [2.5,6]

Q2]

a) to compute the new clusters we calculate the euclidean distance between each points and each centroid then assigne the closest centroid

|  |  |  |  |
| --- | --- | --- | --- |
|  | M2(3,3) | M1(2,2) | Cluster |
| 1,1 | 2.83 | 1.41 | M1 |
| 1,2 | 2.24 | 1 | M1 |
| 1,3 | 2 | 1.41 | M1 |
| 2,2 | 1.41 | 0 | M1 |
| 3,3 | 0 | 1.41 | M2 |
| 3,4 | 1 | 2.4 | M2 |
| 4,3 | 1 | 2.4 | M2 |
| 4,4 | 1.41 | 2.82 | M2 |

The cost will be the summation of all those numbers = 24.44

b)

Dissimilarity matrix

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | P1(1,1) | P2(1,2) | P3(1,3) | P4(2,2) | P5(3,3) | P6(3,4) | P7(4,3) | P8(4,4) |
| P1(1,1) | 0 | 1 | 2 | 1.41 | 2.83 | 3.6 | 3.6 | 4.24 |
| P2(1,2) |  | 0 | 1 | 1 | 2.24 | 2.83 | 3.16 | 3.6 |
| P3(1,3) |  |  | 0 | 1.41 | 2 | 2.24 | 3 | 3.16 |
| P4(2,2) |  |  |  | 0 | 1.41 | 2.24 | 2.24 | 2.83 |
| P5(3,3) |  |  |  |  | 0 | 1 | 1 | 1.41 |
| P6(3,4) |  |  |  |  |  | 0 | 1.41 | 1 |
| P7(4,3) |  |  |  |  |  |  | 0 | 1 |
| P8(4,4) |  |  |  |  |  |  |  | 0 |

First iteration:

Take minimum distances = 1 as clusters then we’re left with {p1,p2},{p3},{p4},{p5,p6},{p7,p8}

We calculate the maximum distances between clusters

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | P1,P2 | P3 | P4 | P5,P6 | P7,P8 |
| P1,P2 | 0 | 2 | 1.41 | 3.6 | 4.24 |
| P3 |  | 0 | 1.41 | 2.24 | 3.16 |
| P4 |  |  | 0 | 2.24 | 2.83 |
| P5,P6 |  |  |  | 0 | 1.41 |
| P7,P8 |  |  |  |  | 0 |

Second iteration:

After taking the closest clustures = 1.41, the clusters are 🡪 {p1,p2,p4},{p3},{p5,p6,p7,p8}

Maximum distances between clusters

|  |  |  |  |
| --- | --- | --- | --- |
|  | P1,P2,P4 | P3 | P5,P6,P7,P8 |
| P1,P2,P4 | 0 | 2 | 4.24 |
| P3 |  | 0 | 3.16 |
| P5,P6,P7,P8 |  |  | 0 |

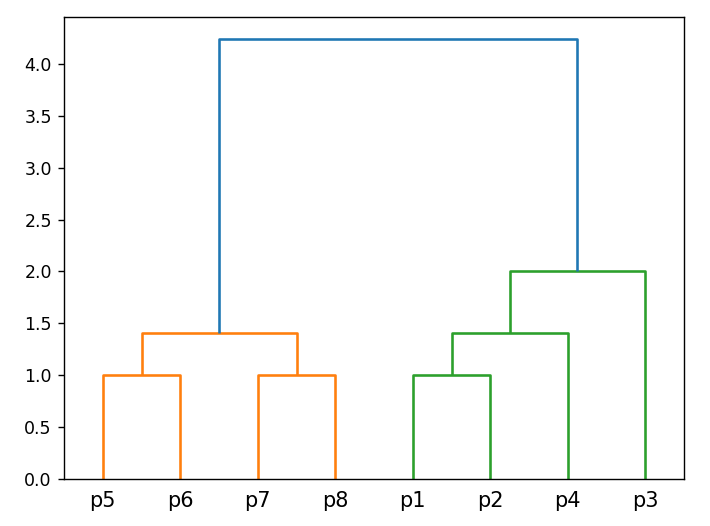
Third iteration:

Minimum = 2

The clusters are 🡪 {p1,p2,p4,p3},{p5,p6,p7,p8}

Maximum distances

|  |  |  |
| --- | --- | --- |
|  | P1,P2,P3,P4 | P5,P6,P7,P8 |
| P1,P2,P3,P4 | 0 | 4.24 |
| P5,P6,P7,P8 |  | 0 |



c)

we calculate distances between each point and all other points then select the k-th (3rd) smallest value

for p1 🡪 1, 1.41, 2, 2.83, 3.61, 3.61, 4.24 🡪 2

for p2 🡪 1, 1, 1, 2.24, 2.83, 3.16, 3.61 🡪 1

for p3 🡪1.0, 1.41, 2.0, 2.0, 2.24, 3.0, 3.16 🡪 2

for p4 🡪 1.0, 1.41, 1.41, 1.41, 2.24, 2.24, 2.83 🡪 1.41

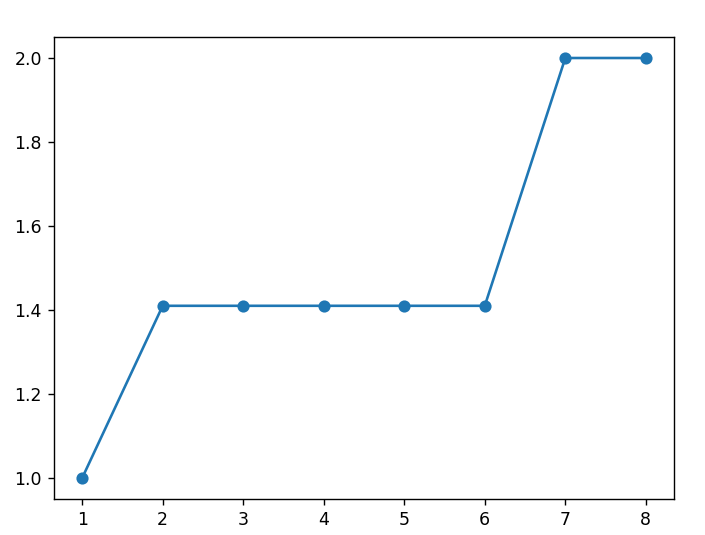
for p5 🡪 1.0, 1.0, 1.41, 1.41, 2.0, 2.24, 2.83 🡪 1.41

for p6 🡪 1.0, 1.0, 1.41, 2.24, 2.24, 2.83, 3.61 🡪 1.41

for p7 🡪 1.0, 1.0, 1.41, 2.24, 3.0, 3.16, 3.61 🡪 1.41

for p8 🡪 1.0, 1.0, 1.41, 2.83, 3.16, 3.61, 4.24 🡪 1.41

sort the values ascendingaley, plot them, then choose the elbow or knee to be the value of



We notice is 1.41

Starting from point (4,4) p8 we get all the points in radius 1.41 – points that has at least k points in its radius will be considered core points 🡪 p5, p6, p7 🡪 p8 is a core point

Visited = p8

Look for the rest of the points starting from the neighbours and repeat

P5 🡪 p4,p6,p7 🡪 p5 is a core point

Visited = p8,p5

P6 🡪 p5,p7,p8 🡪 p6 is a core point

Visited = p8,p5,p6

P7 🡪 p5,p6,p7 🡪 p7 is a core point

Visited = p8,p5,p6,p7

P4 🡪 p1,p2,p3,p5 🡪 p4 is a core point

Visited = p8,p5,p6,p7,p4

P3 🡪 p2,p4 🡪 p3 is a core point

Visited = p8,p5,p6,p7,p4,p3

P2 🡪 p1,p3,p4 🡪 p2 is a core point

Visited = p8,p5,p6,p7,p4,p3,p2

P1 🡪 p2,p4 🡪 p1 is a core point

Visited = p8,p5,p6,p7,p4,p3,p2,p1

This means all the data is in one cluster

d) since the data is all in a single cluster we will cut the dendrogram at the highest point wich is a vertical line at 4.24