CI_Hamed-Mohammadzadeh_Unsupervised

April 7, 2022

[4]:
```
from sklearn.cluster import DBSCAN
from sklearn import metrics
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import numpy as np
from PIL import Image
import glob
from numpy import asarray
from sklearn.neighbors import NearestNeighbors
from matplotlib import pyplot as plt
from sklearn.cluster import AgglomerativeClustering
from sklearn.cluster import DBSCAN
```

1 Rand Index

[5]:
```
def ri(data_with_label, k = 41):
    pred_cluster_dict = {} # key is label predicted by kmeans, values are␣
 ↪pic_nums of pics that are labeled with the respected key
    for i, pred_cluster in enumerate(data_with_label[:, 3]):
        if pred_cluster in pred_cluster_dict.keys():
          pred_cluster_dict[pred_cluster].append(data_with_label[i][0])
        else:
          pred_cluster_dict[pred_cluster] = [data_with_label[i][0]]

    data_label_dict = {} # key is pic_num, values are true_label(original␣
 ↪label) of that key
    for i, data_id in enumerate(data_with_label[:, 0]):
        data_label_dict[data_id] = data_with_label[i][1]

    data_pop_dict = {} # key is original cluster nummber, value is population
    for i, label in enumerate(data_with_label[:, 1]):
        if label in data_pop_dict.keys():
          data_pop_dict[label]+=1
        else:
          data_pop_dict[label] = 1
```

```
    TP_FP = 0
    for key in pred_cluster_dict:
        Ni = len(pred_cluster_dict[key])
        TP_FP += int((Ni * (Ni-1))/2)

    TP = 0
    for key in pred_cluster_dict:
        item_count = {}
        # print(label_dict[key])
        for item in pred_cluster_dict[key]:
            if data_label_dict[item] in item_count.keys():
                item_count[data_label_dict[item]]+= 1
            else:
                item_count[data_label_dict[item]] = 1
        # print(item_count)

        for i in item_count.keys():
            same_items = item_count[i]
            TP += int(same_items * (same_items-1)/2)
            # print(TP)

    FP = TP_FP - TP

    FN = 0
    for data_type in data_pop_dict:
        Ni = data_pop_dict[data_type]
        TPi = 0
        for key in pred_cluster_dict:
            pop_in_curr_cluster = 0
            for data in pred_cluster_dict[key]:
                if data_label_dict[data] == data_type:
                    pop_in_curr_cluster += 1
            TPi += int(pop_in_curr_cluster * (pop_in_curr_cluster-1)/2)
        #print(TPi)
        FN += int((Ni * (Ni-1)/2) - TPi)

    N = 0
    for data_type in data_pop_dict:
        N += data_pop_dict[data_type]
    TN = int(N*(N-1)/2 - (FN + TP + FP))

    RI = (TP + TN) / (TP + TN + FP + FN)
    return RI
```

[6]:
```
def print_dict(data_with_label):
```

```
    pred_cluster_dict = {} # key is label predicted by kmeans, values are␣
↪pic_nums of pics that are labeled with the respected key
    for i, pred_cluster in enumerate(data_with_label[:, 3]):
        if pred_cluster in pred_cluster_dict.keys():
          pred_cluster_dict[pred_cluster].append(data_with_label[i][0])
        else:
          pred_cluster_dict[pred_cluster] = [data_with_label[i][0]]

    for keys,values in pred_cluster_dict.items():
        print(keys)
        print(values)
```

2 Data Preprocessing

[7]:
```
filelist = glob.glob('ORL/*.jpg')
list = []
for fname in filelist:
    pic_num = int(fname.split("\\")[-1].split(".")[0].split("_")[0])
    cluster_num = int(fname.split("\\")[1].split(".")[0].split("_")[1])
    img_arr = asarray(Image.open(fname).convert('L')).flatten()
    list.append([pic_num, cluster_num, img_arr])
    #print(img_arr)

#1, 2, 3(arr)
data = np.array(list)

data.shape, list[0]
```

[7]:
```
((410, 3), [100, 10, array([100, 118, 111, …,  21,  20,  21], dtype=uint8)])
```

3 KMEANS

[8]:
```
kmeans = KMeans(n_clusters=41, init = 'random', n_init = 1, random_state=11).
↪fit(np.stack(data[:, 2]))
output_labels0 = np.array(kmeans.labels_).reshape((len(kmeans.labels_), 1))
data_with_label0 = np.append(data, output_labels0, axis=1)
ri(data_with_label0), print_dict(data_with_label0)
```

```
38
[100, 266, 268, 269, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80]
2
[101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 139]
24
[10, 126, 171, 172, 174, 177, 179, 2]
18
```

```
[111, 116, 118, 129, 173, 175, 176, 178, 180, 392, 395, 45, 46, 48, 49, 50]
17
[112, 113, 114, 115, 117, 119, 120]
7
[11, 13, 151, 156, 15, 161, 17, 189, 18, 20, 272, 279]
10
[121, 122, 123, 124, 125, 127, 128, 130, 244, 245, 248, 25, 394, 397, 400, 41,
42, 43, 44, 47]
13
[12, 141, 143, 144, 146, 148, 14, 16, 19, 231, 235, 237, 316, 317, 319, 320]
35
[131, 132, 133, 134, 135, 136, 137, 138, 140, 362, 364, 365, 366, 368]
40
[142, 145, 147, 149, 150, 341]
21
[152, 153, 157, 158, 159, 160, 1, 3, 7]
23
[154, 155]
0
[162, 165, 222, 228, 85, 90]
26
[163, 164, 21, 241, 243, 246, 247, 249, 24, 250, 251, 252, 253, 256, 259, 26,
27, 28, 29, 30]
19
[166, 167, 168, 169, 170, 352]
34
[181, 182, 183, 184, 185, 186, 187, 188, 190, 261, 262, 263, 264, 265, 270, 274,
275, 277]
36
[191, 192, 193, 194, 195, 196, 197, 198, 199, 200]
20
[201, 204, 205, 206, 207, 208, 209, 210, 232, 233, 234, 236, 238, 239, 240, 325,
326, 329]
16
[202, 203, 281, 282, 283, 289, 290, 294, 295, 297, 386, 390]
32
[211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 381, 383, 384, 385, 389]
11
[221, 224, 226, 227, 229, 230, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380,
86, 87, 99]
5
[223, 225, 81, 82, 83, 84, 88, 89]
14
[22, 23, 254, 255, 258, 260]
30
[242, 32, 34, 35, 37, 39, 40]
6
[257, 271, 273, 276, 278, 280, 361, 363, 367, 369, 370]
```

```
33
[267, 31, 33, 356, 357, 360, 36, 38]
1
[284, 285, 286, 287, 288]
28
[291, 292, 293, 296, 298, 299, 300, 301, 306, 307, 308, 309]
29
[302, 303, 304, 305, 310]
31
[311, 312, 313, 314, 315, 318]
4
[321, 322, 323, 324, 327, 328, 330]
22
[331, 332, 333, 334, 335, 336, 337, 338, 339, 340]
27
[342, 344, 346, 347, 348]
39
[343, 345, 349, 350, 399]
3
[351, 353, 354, 355, 358, 359, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70]
12
[382, 387, 388]
37
[391, 393, 396, 398]
8
[401, 402, 403, 404, 405, 406, 407, 408, 409, 410]
15
[4, 5, 6, 8, 9]
9
[51, 52, 53, 54, 55, 56, 57, 58, 59, 60]
25
[91, 92, 93, 94, 95, 96, 97, 98]
```

[8]: (0.9754904884012165, None)

4 AGGLOMERATIVE

4.0.1 Average Link

[9]:
```
k = 41
agglo = AgglomerativeClustering(n_clusters = k, linkage = 'average').fit(np.
 ↪stack(data[:, 2]))
output_labels2 = np.array(agglo.labels_).reshape((len(agglo.labels_), 1))
data_with_label2 = np.append(data, output_labels2, axis=1)

ri(data_with_label2), print_dict(data_with_label2)
```

```
36
[100]
1
[101, 103, 106, 107, 108, 109, 110]
32
[102]
20
[104, 105]
30
[10, 5, 8, 9]
9
[111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 251, 252, 253, 254, 255, 256,
257, 258, 259, 260]
8
[11, 12, 13, 141, 143, 144, 146, 148, 14, 15, 16, 17, 18, 19, 20, 317]
3
[121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 142, 145, 147, 149, 150, 171,
172, 173, 174, 175, 176, 177, 178, 179, 180, 244, 245, 248, 24, 25, 391, 392,
393, 394, 395, 396, 397, 398, 399, 400, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50]
6
[131, 132, 133, 134, 135, 136, 137, 138, 139, 140]
40
[151, 156]
2
[152, 153, 157, 158, 159, 160, 1, 3, 7]
23
[154, 155]
0
[161, 162, 163, 164, 165, 21, 22, 23, 241, 242, 243, 246, 247, 249, 250, 26, 27,
28, 29, 30, 31, 32, 33, 342, 344, 346, 347, 348, 34, 35, 36, 37, 38, 39, 40]
7
[166, 167, 168, 169, 170, 352, 356, 357]
15
[181, 182, 183, 184, 185, 186, 187, 188, 190, 261, 262, 263, 264, 265, 270]
34
[189, 360]
14
[191, 192, 193, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 281, 282,
283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298,
299, 300, 301, 306, 307, 308, 309, 321, 322, 323, 324, 325, 326, 327, 328, 329,
330]
26
[194, 196]
37
[195, 197, 198, 199]
18
[211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 381, 382, 383, 384, 385, 386,
387, 388, 389, 390]
```

```
22
[221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 371, 372, 373, 374, 375, 376,
377, 378, 379, 380, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90]
35
[231, 232, 233, 234, 235, 236, 237, 238, 239, 240]
29
[266, 267, 268, 269]
10
[271, 273, 276, 278, 280, 361, 363, 367, 369, 370]
4
[272, 274, 275, 277, 279, 362, 364, 365, 366, 368]
33
[2]
38
[302, 303, 304, 305, 310]
28
[311, 312, 313, 314, 315, 316, 318, 319, 320]
5
[331, 332, 333, 334, 335, 336, 337, 338, 339, 340]
25
[341]
17
[343, 345, 349, 350]
13
[351, 353, 354, 355, 358, 359, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70]
39
[401]
31
[402, 405, 406]
12
[403, 404, 408, 409, 410]
27
[407]
24
[4, 6]
21
[51, 52, 53, 54, 55, 56, 57, 58, 59, 60]
11
[71, 72, 73, 74, 75, 76, 77, 78, 79, 80]
16
[91, 99]
19
[92, 93, 94, 95, 96, 97, 98]
```

[9]: (0.9560140735881686, None)

4.0.2 Single Link

[10]:
```
k = 41
agglo = AgglomerativeClustering(n_clusters = 60, linkage = 'single').fit(np.
 ↪stack(data[:, 2]))
output_labels3 = np.array(agglo.labels_).reshape((len(agglo.labels_), 1))
data_with_label3 = np.append(data, output_labels3, axis=1)

ri(data_with_label3), print_dict(data_with_label3)
```

```
37
[100]
0
[101, 102, 103, 106, 107, 108, 109, 110]
20
[104, 105]
40
[10]
13
[111, 112, 113, 114, 115, 116, 117, 118, 119, 120]
32
[11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
1
[121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 142, 145, 147, 149, 150, 163,
164, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180,
201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216,
217, 218, 219, 21, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 22, 230,
23, 241, 242, 243, 244, 245, 246, 247, 248, 249, 24, 250, 251, 252, 253, 254,
255, 256, 257, 258, 259, 25, 260, 26, 271, 273, 276, 27, 280, 281, 282, 283,
284, 285, 286, 287, 288, 289, 28, 290, 291, 292, 293, 294, 295, 296, 297, 298,
299, 29, 300, 301, 306, 307, 308, 309, 30, 31, 321, 322, 323, 324, 325, 326,
327, 328, 329, 32, 330, 33, 342, 344, 346, 347, 348, 34, 35, 361, 363, 367, 369,
36, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 37, 380, 381, 382, 383,
384, 385, 386, 387, 388, 389, 38, 390, 391, 392, 393, 394, 395, 396, 397, 398,
399, 39, 400, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 81, 82, 83, 84, 85,
86, 87, 88, 89, 90]
34
[131, 132, 133, 134, 135]
58
[136, 139]
54
[137, 138]
49
[140]
24
[141, 143, 144, 146, 148]
41
[151]
```

```
26
[152, 153, 160]
21
[154, 155]
45
[156]
47
[157]
35
[158]
38
[159]
19
[161, 162, 165]
8
[181, 182, 183, 184, 185, 186, 187, 188, 190]
42
[189]
27
[191, 192, 193, 200]
33
[194, 196]
10
[195, 197, 198, 199]
2
[1, 7]
29
[231, 232, 233, 234, 235, 236, 237, 238, 239, 240]
4
[261, 262, 263, 264, 265, 266, 267, 268, 269, 270]
59
[272, 275, 277]
51
[274]
46
[278]
30
[279]
50
[2]
17
[302, 303, 304, 305, 310]
6
[311, 312, 313, 314, 315, 316, 318, 319, 320]
55
[317]
15
[331, 332, 333, 334, 335, 336, 337, 338, 339, 340]
```

```
39
[341]
11
[343, 345, 349, 350]
9
[351, 355]
5
[352, 356, 357]
14
[353, 354, 358, 359]
52
[360]
23
[362, 364, 365, 366, 368]
36
[3]
31
[401]
44
[402]
22
[403]
18
[404, 408, 409, 410]
53
[405, 406]
48
[407]
3
[4, 6]
12
[51, 52, 53, 54, 55, 56, 57, 58, 59, 60]
57
[5]
28
[61, 62, 63, 64, 65, 66, 67, 68, 69, 70]
16
[71, 72, 73, 74, 75, 76, 77, 78, 79, 80]
56
[8]
7
[91, 92, 93, 94, 95, 96, 97, 98]
43
[99]
25
[9]
```

[10]: (0.7650187846621742, None)

4.0.3 Complete Link

[11]:
```
k = 41
agglo = AgglomerativeClustering(n_clusters = k, linkage = 'complete').fit(np.
 ↪stack(data[:, 2]))
output_labels4 = np.array(agglo.labels_).reshape((len(agglo.labels_), 1))
data_with_label4 = np.append(data, output_labels4, axis=1)

ri(data_with_label4), print_dict(data_with_label4)
```

```
2
[100, 189, 266, 267, 268, 269, 31, 33, 360, 36, 38]
34
[101, 102, 103, 106]
7
[104, 105, 136, 137, 138, 139, 140]
23
[107, 108, 109, 110]
30
[10, 5, 8, 9]
27
[111, 112, 113, 114, 115, 116, 117, 118, 119, 120]
1
[11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 311, 312, 313, 314, 315, 316, 317, 318,
319, 320]
5
[121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 244, 245, 248, 24, 25, 394,
400, 41, 42, 43, 44, 47]
25
[131, 132, 133, 134, 135]
10
[141, 143, 144, 146, 148, 151, 156, 231, 232, 235, 236, 237, 238]
4
[142, 145, 147, 149, 150, 241, 242, 243, 246, 247, 249, 250, 251, 252, 253, 254,
255, 256, 258, 259, 260, 342, 344, 346, 347, 348]
3
[152, 153, 157, 158, 159, 160, 1, 3, 7]
40
[154, 155]
37
[161, 162, 165, 32, 34, 35, 37, 39, 40]
26
[163, 164, 21, 222, 228, 22, 23, 26, 27, 28, 29, 30, 81, 82, 83, 84, 85, 86, 87,
88, 89, 90]
12
```

```
[166, 167, 168, 169, 170, 233, 234, 239, 240, 352, 356, 357]
16
[171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 392, 393, 395, 397, 399, 45,
46, 48, 49, 50]
15
[181, 182, 183, 184, 185, 186, 187, 188, 190, 274]
14
[191, 192, 193, 194, 196, 200, 284, 285, 286, 287, 288]
32
[195, 197, 198, 199]
17
[201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 301, 306, 307, 308, 309]
38
[211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 281, 282, 283, 289, 290, 381,
382, 383, 384, 385, 386, 387, 388, 389, 390]
8
[221, 223, 224, 225, 226, 227, 229, 230, 371, 372, 373, 374, 375, 376, 377, 378,
379, 380]
6
[257, 271, 273, 276, 278, 280, 361, 363, 367, 369, 370]
0
[261, 262, 263, 264, 265, 270, 272, 275, 277, 279, 362, 364, 365, 366, 368]
13
[291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 321, 322, 323, 324, 325, 326,
327, 328, 329, 330]
39
[2]
29
[302, 303, 304, 305, 310]
11
[331, 332, 333, 334, 335, 336, 337, 338, 339, 340]
33
[341, 391, 396, 398]
18
[343, 345, 349, 350]
22
[351, 353, 354, 355, 358, 359, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70]
28
[401]
21
[402, 405, 406]
36
[403, 404, 408, 409, 410]
31
[407]
35
[4, 6]
20
```

```
[51, 52, 53, 54, 55, 56, 57, 58, 59, 60]
24
[71, 72, 73, 74, 75, 76, 77, 78, 79, 80]
19
[91, 99]
9
[92, 93, 94, 95, 96, 97, 98]
```

[11]:
```
(0.973248255709941, None)
```

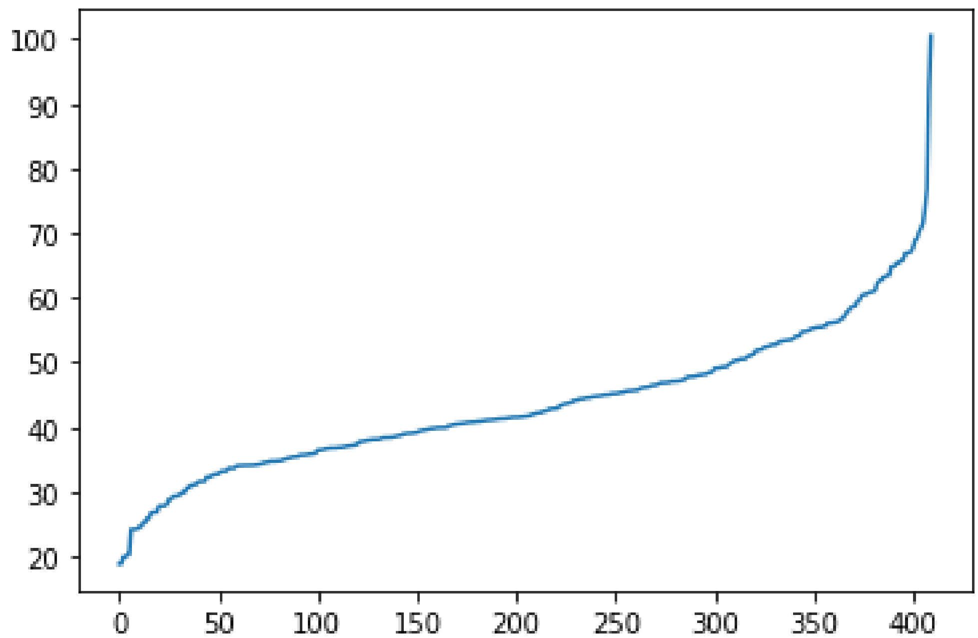
5 DBSCAN

Using KNN for choosing epsilon - https://medium.com/@tarammullin/dbscan-parameter-estimation-ff8330e3a3bd

5.0.1 Choosing epsilon

[12]:
```
from sklearn.neighbors import NearestNeighbors
from matplotlib import pyplot as plt

neighbors = NearestNeighbors(n_neighbors=4)
neighbors_fit = neighbors.fit(StandardScaler().fit_transform(np.stack(data[:,␣
 ↪2])))
distances, indices = neighbors_fit.kneighbors(StandardScaler().fit_transform(np.
 ↪stack(data[:, 2])))

distances = np.sort(distances, axis=0)
distances = distances[:,1]
plt.plot(distances)

# the average distance between each point and its k nearest neighbors,
# where k = the MinPts value you selected. The average k-distances are
# then plotted in ascending order on a k-distance graph.
# You'll find the optimal value for  at the point of maximum curvature
```

[12]:
```
[<matplotlib.lines.Line2D at 0x18add704948>]
```



[13]:
```
ep0 = 55
n0 = 4
db = DBSCAN(eps=ep0, min_samples=n0).fit(StandardScaler().fit_transform(np.
 ↪stack(data[:, 2])))
output_labels5 = np.array(db.labels_).reshape((len(db.labels_), 1))
k0 = max(db.labels_) + 1
# pic num, clust num, img arr, label
data_with_label5 = np.append(data, output_labels5, axis=1)
k0, ri(data_with_label5), print_dict(data_with_label5)
```

```
-1
[100, 102, 104, 105, 107, 108, 109, 10, 111, 113, 114, 116, 118, 119, 123, 124,
131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 148, 151, 152, 153, 154, 155,
156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 189, 194, 195, 196, 197, 198,
199, 1, 214, 222, 223, 225, 228, 235, 244, 245, 248, 24, 252, 257, 25, 266, 267,
268, 269, 26, 271, 272, 273, 274, 275, 276, 277, 278, 279, 27, 280, 28, 2, 301,
302, 306, 307, 308, 309, 311, 312, 313, 314, 315, 316, 317, 318, 319, 31, 320,
328, 334, 335, 337, 33, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351,
352, 353, 354, 355, 356, 357, 358, 359, 360, 368, 36, 381, 382, 383, 384, 385,
386, 387, 388, 389, 38, 390, 391, 393, 396, 399, 3, 401, 402, 403, 404, 405,
406, 407, 408, 409, 40, 410, 4, 5, 6, 7, 86, 87, 8, 91, 92, 93, 94, 95, 96, 97,
98, 99, 9]
0
[101, 103, 106, 110]
1
[112, 115, 117, 120]
```

2
[11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
3
[121, 122, 125, 126, 127, 128, 129, 130]
4
[141, 143, 144, 146]
5
[142, 145, 147, 149, 150]
6
[166, 167, 168, 169, 170]
7
[171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 392, 395, 397, 50]
8
[181, 182, 183, 186, 190]
9
[184, 185, 187, 188]
10
[191, 192, 193, 200]
11
[201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300]
12
[211, 212, 213, 215, 216, 217, 218, 219, 220]
13
[21, 22, 23, 29, 30, 32, 34, 35, 374, 37, 39]
14
[221, 224, 226, 227, 229, 230, 371, 372, 373, 375, 376, 377, 378, 379, 380, 81, 82, 83, 84, 85, 88, 89, 90]
15
[231, 232, 236, 237, 238]
16
[233, 234, 239, 240]
17
[241, 242, 243, 246, 247, 249, 250]
18
[251, 253, 254, 255, 256, 258, 259, 260]
19
[261, 262, 263, 264, 265, 270]
20
[281, 282, 283, 284, 285, 286, 287, 288, 289, 290]
21
[303, 304, 305, 310]
22
[321, 322, 323, 324, 325, 326, 327, 329, 330]
23
[331, 332, 333, 336, 338, 339, 340]
24
[361, 363, 367, 369, 370]

```
25
[362, 364, 365, 366]
26
[394, 398, 400, 41, 42, 43, 44, 45, 46, 47, 48, 49]
27
[51, 52, 53, 54, 55, 56, 57, 58, 59, 60]
28
[61, 62, 63, 64, 65, 66, 67, 68, 69, 70]
29
[71, 72, 74, 77, 79]
30
[73, 75, 76, 78, 80]
```

[13]:
```
(31, 0.8267398175204246, None)
```

6 Enhanced DBSCAN

Perform DBSCAN clustering on noises iteratively

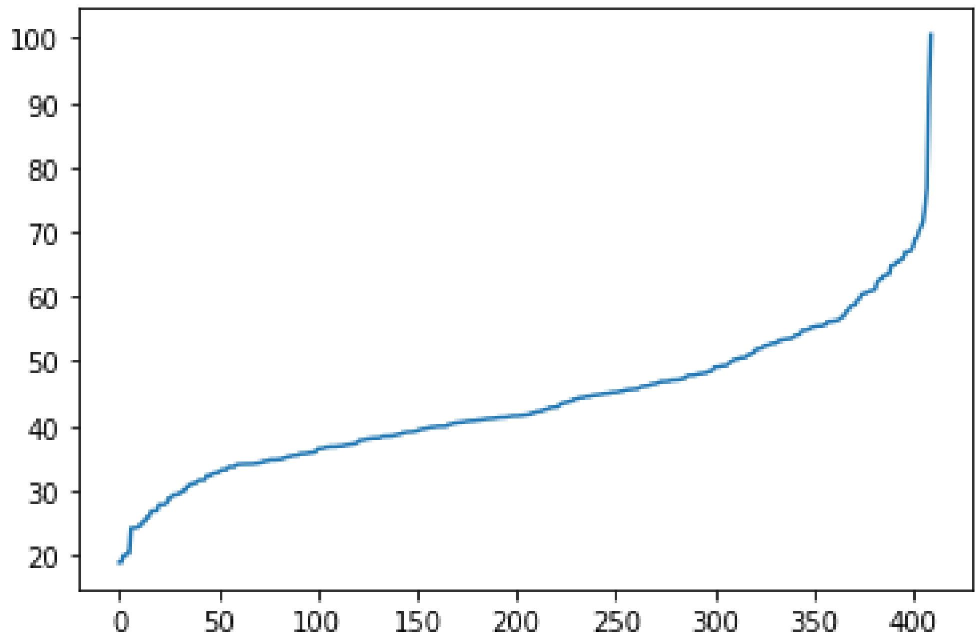
[14]:
```
print('n_neighbors for plotting...')
n_ne = input()

neighbors = NearestNeighbors(n_neighbors=int(n_ne))
neighbors_fit = neighbors.fit(StandardScaler().fit_transform(np.stack(data[:,␣
↪2])))
distances, indices = neighbors_fit.kneighbors(StandardScaler().fit_transform(np.
↪stack(data[:, 2])))

distances = np.sort(distances, axis=0)
distances = distances[:,1]
plt.plot(distances)
plt.show()

print('eps...')
eps = int(input())
print('min_points...')
n = int(input())

db = DBSCAN(eps=eps, min_samples=n).fit(StandardScaler().fit_transform(np.
↪stack(data[:, 2])))
output_labels = np.array(db.labels_).reshape((len(db.labels_), 1))
k = max(db.labels_) + 1
# pic num, clust num, img arr, label
data_with_label = np.append(data, output_labels, axis=1)
print('ri: ', ri(data_with_label))

print_dict(data_with_label)
```

```
while True:
    print('continue?')
    confirm = int(input())

    if confirm != 1:
        break

    noises1 = data_with_label[data_with_label[:, 3] == -1]

    print('n_neighbors for plotting...')
    n_ne1 = input()
    neighbors1 = NearestNeighbors(n_neighbors=int(n_ne1))
    neighbors1_fit = neighbors1.fit(StandardScaler().fit_transform(np.
↪stack(noises1[:, 2])))
    distances1, indices1 = neighbors1_fit.kneighbors(StandardScaler().
↪fit_transform(np.stack(noises1[:, 2])))

    distances1 = np.sort(distances1, axis=0)
    distances1 = distances1[:,1]
    plt.plot(distances1)
    plt.show()

    print('eps...')
    eps1 = int(input())
    print('min_points...')
    n1 = int(input())

    db1 = DBSCAN(eps=eps1, min_samples=n1).fit(StandardScaler().
↪fit_transform(np.stack(noises1[:, 2])))
    output_labels1 = np.array(db1.labels_).reshape((len(db1.labels_), 1))

    # pic num, clust num, img arr, label
    noises1[:, 3] = np.resize(output_labels1, len(output_labels1))

    print('Clustering for noiese: ')
    print_dict(noises1)


    # merging
    k = max(data_with_label[:, 3])
    for i, pred_cluster in enumerate(data_with_label[:, 3]):
        if pred_cluster == -1:
            for j, noise_pred_cluster in enumerate(noises1[:, 3]):
                if data_with_label[i][0] == noises1[j][0] and␣
↪noise_pred_cluster!= -1:
                    data_with_label[i][3] = noises1[j][3] + k + 1
```

```
    print('Clustering for  whole data: ')
    print_dict(data_with_label)


    print('ri: ', ri(data_with_label))
```

n_neighbors for plotting…

 4



eps…

 55

min_points…

 4

```
ri:  0.8267398175204246
-1
[100, 102, 104, 105, 107, 108, 109, 10, 111, 113, 114, 116, 118, 119, 123, 124,
131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 148, 151, 152, 153, 154, 155,
156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 189, 194, 195, 196, 197, 198,
199, 1, 214, 222, 223, 225, 228, 235, 244, 245, 248, 24, 252, 257, 25, 266, 267,
268, 269, 26, 271, 272, 273, 274, 275, 276, 277, 278, 279, 27, 280, 28, 2, 301,
302, 306, 307, 308, 309, 311, 312, 313, 314, 315, 316, 317, 318, 319, 31, 320,
328, 334, 335, 337, 33, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351,
352, 353, 354, 355, 356, 357, 358, 359, 360, 368, 36, 381, 382, 383, 384, 385,
386, 387, 388, 389, 38, 390, 391, 393, 396, 399, 3, 401, 402, 403, 404, 405,
```

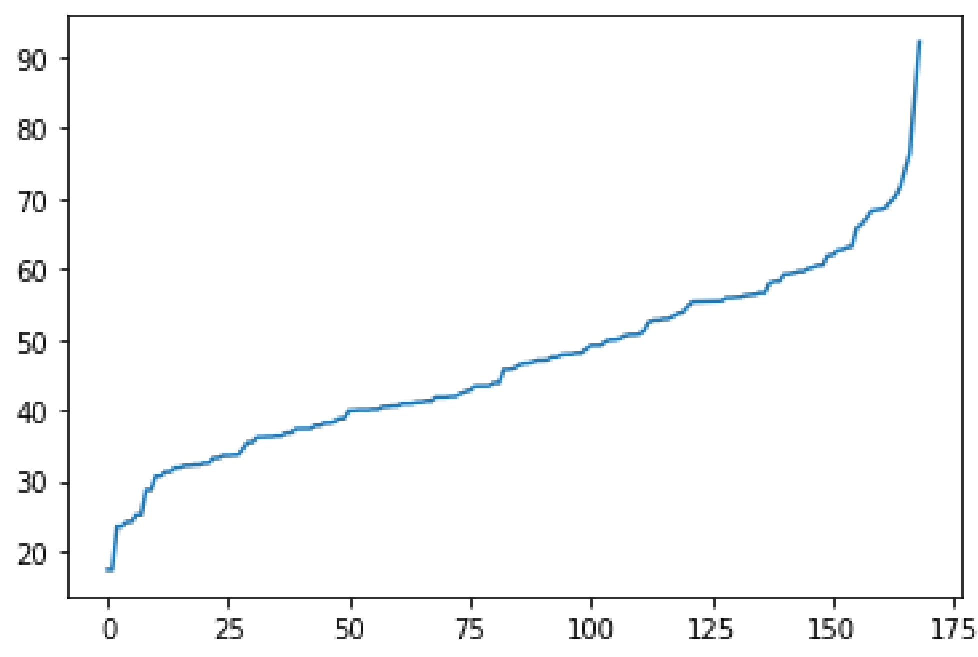
```
406, 407, 408, 409, 40, 410, 4, 5, 6, 7, 86, 87, 8, 91, 92, 93, 94, 95, 96, 97,
98, 99, 9]
0
[101, 103, 106, 110]
1
[112, 115, 117, 120]
2
[11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
3
[121, 122, 125, 126, 127, 128, 129, 130]
4
[141, 143, 144, 146]
5
[142, 145, 147, 149, 150]
6
[166, 167, 168, 169, 170]
7
[171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 392, 395, 397, 50]
8
[181, 182, 183, 186, 190]
9
[184, 185, 187, 188]
10
[191, 192, 193, 200]
11
[201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 291, 292, 293, 294, 295, 296,
297, 298, 299, 300]
12
[211, 212, 213, 215, 216, 217, 218, 219, 220]
13
[21, 22, 23, 29, 30, 32, 34, 35, 374, 37, 39]
14
[221, 224, 226, 227, 229, 230, 371, 372, 373, 375, 376, 377, 378, 379, 380, 81,
82, 83, 84, 85, 88, 89, 90]
15
[231, 232, 236, 237, 238]
16
[233, 234, 239, 240]
17
[241, 242, 243, 246, 247, 249, 250]
18
[251, 253, 254, 255, 256, 258, 259, 260]
19
[261, 262, 263, 264, 265, 270]
20
[281, 282, 283, 284, 285, 286, 287, 288, 289, 290]
21
[303, 304, 305, 310]
```

```
22
[321, 322, 323, 324, 325, 326, 327, 329, 330]
23
[331, 332, 333, 336, 338, 339, 340]
24
[361, 363, 367, 369, 370]
25
[362, 364, 365, 366]
26
[394, 398, 400, 41, 42, 43, 44, 45, 46, 47, 48, 49]
27
[51, 52, 53, 54, 55, 56, 57, 58, 59, 60]
28
[61, 62, 63, 64, 65, 66, 67, 68, 69, 70]
29
[71, 72, 74, 77, 79]
30
[73, 75, 76, 78, 80]
continue?
```

1

```
n_neighbors for plotting…
```

3




```
eps…
```

```
 60

min_points…

 3

Clustering for noiese:
-1
[100, 102, 104, 105, 123, 124, 136, 137, 138, 139, 140, 148, 151, 154, 155, 156,
158, 189, 194, 196, 235, 257, 274, 278, 279, 2, 302, 317, 328, 341, 351, 355,
360, 368, 391, 393, 396, 401, 403, 407, 40, 4, 6, 99]
0
[107, 108, 109]
1
[10, 5, 8, 9]
2
[111, 113, 114, 116, 118, 119, 163, 164, 244, 245, 248, 24, 252, 25, 26, 27, 28,
31, 33, 342, 344, 346, 347, 348, 36, 38, 86, 87]
3
[131, 132, 133, 134, 135]
4
[152, 153, 157, 159, 160, 1, 3, 7]
5
[161, 162, 165]
6
[195, 197, 198, 199]
17
[214, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390]
7
[222, 223, 225, 228]
8
[266, 267, 268, 269]
9
[271, 273, 276, 280]
10
[272, 275, 277]
11
[301, 306, 307, 308, 309]
12
[311, 312, 313, 314, 315, 316, 318, 319, 320]
13
[334, 335, 337]
14
[343, 345, 349, 350, 399]
16
[352, 356, 357]
15
[353, 354, 358, 359]
19
```

```
[402, 405, 406]
18
[404, 408, 409, 410]
20
[91, 92, 93, 94, 95, 96, 97, 98]
Clustering for  whole data:
-1
[100, 102, 104, 105, 123, 124, 136, 137, 138, 139, 140, 148, 151, 154, 155, 156,
158, 189, 194, 196, 235, 257, 274, 278, 279, 2, 302, 317, 328, 341, 351, 355,
360, 368, 391, 393, 396, 401, 403, 407, 40, 4, 6, 99]
0
[101, 103, 106, 110]
31
[107, 108, 109]
32
[10, 5, 8, 9]
33
[111, 113, 114, 116, 118, 119, 163, 164, 244, 245, 248, 24, 252, 25, 26, 27, 28,
31, 33, 342, 344, 346, 347, 348, 36, 38, 86, 87]
1
[112, 115, 117, 120]
2
[11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
3
[121, 122, 125, 126, 127, 128, 129, 130]
34
[131, 132, 133, 134, 135]
4
[141, 143, 144, 146]
5
[142, 145, 147, 149, 150]
35
[152, 153, 157, 159, 160, 1, 3, 7]
36
[161, 162, 165]
6
[166, 167, 168, 169, 170]
7
[171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 392, 395, 397, 50]
8
[181, 182, 183, 186, 190]
9
[184, 185, 187, 188]
10
[191, 192, 193, 200]
37
[195, 197, 198, 199]
11
```

```
[201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 291, 292, 293, 294, 295, 296,
297, 298, 299, 300]
12
[211, 212, 213, 215, 216, 217, 218, 219, 220]
48
[214, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390]
13
[21, 22, 23, 29, 30, 32, 34, 35, 374, 37, 39]
14
[221, 224, 226, 227, 229, 230, 371, 372, 373, 375, 376, 377, 378, 379, 380, 81,
82, 83, 84, 85, 88, 89, 90]
38
[222, 223, 225, 228]
15
[231, 232, 236, 237, 238]
16
[233, 234, 239, 240]
17
[241, 242, 243, 246, 247, 249, 250]
18
[251, 253, 254, 255, 256, 258, 259, 260]
19
[261, 262, 263, 264, 265, 270]
39
[266, 267, 268, 269]
40
[271, 273, 276, 280]
41
[272, 275, 277]
20
[281, 282, 283, 284, 285, 286, 287, 288, 289, 290]
42
[301, 306, 307, 308, 309]
21
[303, 304, 305, 310]
43
[311, 312, 313, 314, 315, 316, 318, 319, 320]
22
[321, 322, 323, 324, 325, 326, 327, 329, 330]
23
[331, 332, 333, 336, 338, 339, 340]
44
[334, 335, 337]
45
[343, 345, 349, 350, 399]
47
[352, 356, 357]
46
```

```
[353, 354, 358, 359]
24
[361, 363, 367, 369, 370]
25
[362, 364, 365, 366]
26
[394, 398, 400, 41, 42, 43, 44, 45, 46, 47, 48, 49]
50
[402, 405, 406]
49
[404, 408, 409, 410]
27
[51, 52, 53, 54, 55, 56, 57, 58, 59, 60]
28
[61, 62, 63, 64, 65, 66, 67, 68, 69, 70]
29
[71, 72, 74, 77, 79]
30
[73, 75, 76, 78, 80]
51
[91, 92, 93, 94, 95, 96, 97, 98]
ri:  0.9709463891704931
continue?

 0
```

```
[ ]:
```