

# Single-Link

February 28, 2018

## 1 TARUN SUNKARANENI'S Hierarchical and Point-Clustering Notebook

```
In [1]: import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from scipy.cluster.hierarchy import dendrogram, linkage
from scipy.spatial.distance import cdist
from matplotlib import pyplot as plt
from scipy.spatial import distance
import math
%matplotlib inline
np.set_printoptions(precision=5, suppress=True) # suppress scientific float notation

In [2]: c1 = pd.read_csv("../input/C1.csv", names=['x0', 'x1'])
```

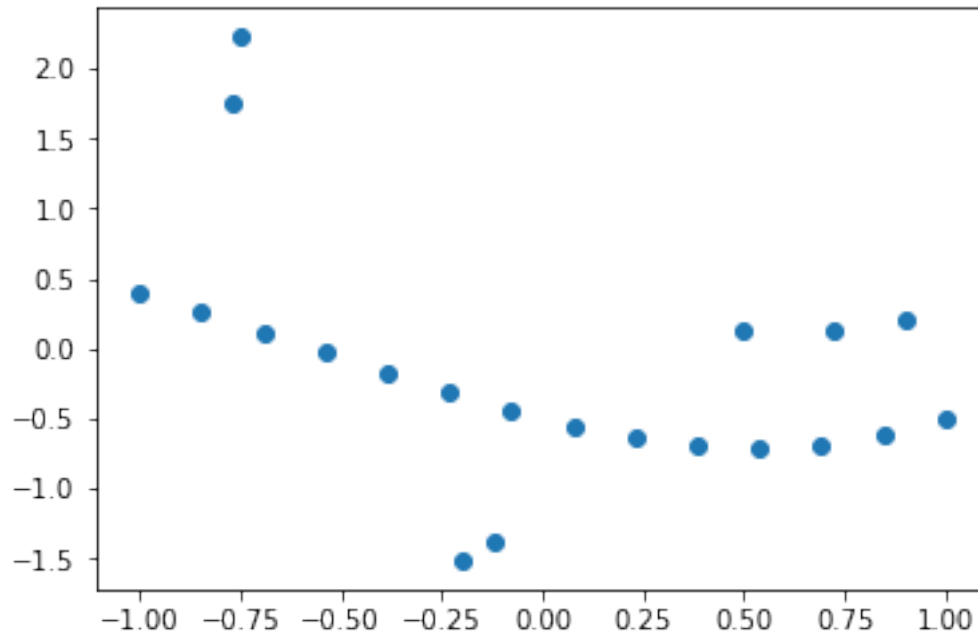
## 2 Single-Link Hierarchical

### 2.0.1 Single-Link: measures the shortest link

$$d(S_1, S_2) = \min_{(s_1, s_2) \in S_1 \times S_2} \|s_1 - s_2\|_2$$

```
In [4]: plt.scatter(c1['x0'], c1['x1'])
```

```
Out[4]: <matplotlib.collections.PathCollection at 0x10ab5ea90>
```



```
In [5]: def single_distance(clusters ,cluster_num):
    print('first cluster | ', 'second cluster | ', 'distance')
    while len(clusters) is not cluster_num:
        # Clustering
        closest_distance=clust_1=clust_2 = math.inf
        # for every cluster (until second last element)
        for cluster_id, cluster in enumerate(clusters[:len(clusters)]):
            # for each point in each cluster
            for point_id,point in enumerate(cluster):
                # we only need to compare with clusters after the current one
                for cluster2_id, cluster2 in enumerate(clusters[(cluster_id+1):]):
                    # go through every point in this prospective cluster as well
                    for point2_id, point2 in enumerate(cluster2):
                        # if this distance is better than our previous best distance then we are going to over
                        if distance.euclidean(point,point2) < closest_distance:
                            # Only used for comparing
                            closest_distance = distance.euclidean(point,point2)
                            # this will be used at the end to figure out which cluster to merge wi
                            clust_1 = cluster_id
                            # this cluster will be destroyed by the end
                            clust_2 = cluster2_id+cluster_id+1
                            # extend just appends the contents to the list without flattening it ou
    print(clust_1,' | ',clust_2, ' | ',closest_distance)
    clusters[clust_1].extend(clusters[clust_2])
    # don't need this index anymore, and we have just clustered once more
```

```

        clusters.pop(clust_2)
    return(clusters)

```

In [6]: *### Hierarchical clustering*

```

def hierarchical(data, cluster_num, metric = 'single'):
    # initialization of clusters at first (every point is a cluster)
    init_clusters=[]
    for index, row in data.iterrows():
        init_clusters.append([[row['x0'], row['x1']]])
    if metric is 'single':
        return single_distance(init_clusters, cluster_num)

```

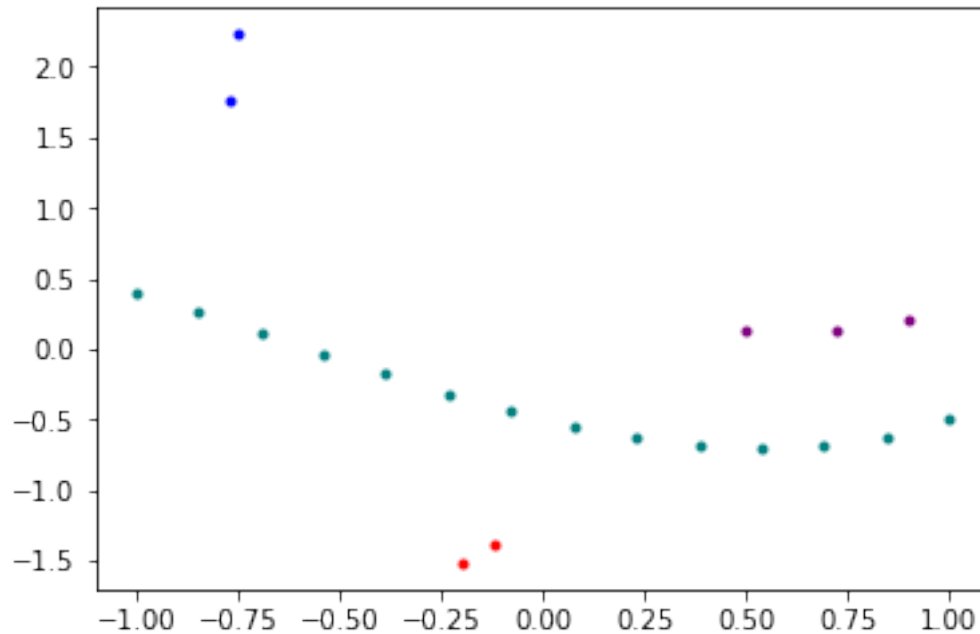
In [7]: clusters = hierarchical(c1,4)

```

colors = ['blue', 'red', 'purple', 'teal']
for cluster_index, cluster in enumerate(clusters):
    for point_index, point in enumerate(cluster):
        plt.plot([point[0]], [point[1]], marker='o', markersize=3, color=colors[cluster_index])

```

first cluster	second cluster	distance
2	3	0.15085042956518227
15	16	0.15501250939640307
15	16	0.155332720189051
14	15	0.16302722103464817
14	15	0.1679299964569166
13	14	0.17501291697817623
12	13	0.18766276935111553
4	5	0.19186599490269243
11	12	0.19429172919524904
10	11	0.19888079280176854
5	6	0.20597708099371148
8	9	0.20743622990434923
5	6	0.21188849347451605
6	7	0.2126411284874588
5	6	0.21415518261316957
3	4	0.22340895237210173
0	1	0.47931424457263944



### 3 Validation

Credit to <https://joernhees.de/blog/2015/08/26/scipy-hierarchical-clustering-and-dendrogram-tutorial/> for this Validation portion

```
In [8]: X = c1.as_matrix()
        # generate the linkage matrix
        single_link = linkage(X, 'single') # using single link metric to evaluate 'distance' b
```

As you can see there's a lot of choice here and while python and scipy make it very easy to do the clustering, it's you who has to understand and make these choices.. This compares the actual pairwise distances of all your samples to those implied by the hierarchical clustering. > The closer the value is to 1, the better the clustering preserves the original distances, which in our case is reasonably close:

```
In [9]: from scipy.cluster.hierarchy import cophenet
        from scipy.spatial.distance import pdist
```

```
In [10]: c, coph_dists = cophenet(single_link, pdist(X))
        c
```

```
Out[10]: 0.74321649752980556
```

No matter what method and metric you pick, the linkage() function will use that method and metric to calculate the distances of the clusters (starting with your n individual samples (aka data

points) as singleton clusters)) and in each iteration will merge the two clusters which have the smallest distance according the selected method and metric. It will return an array of length  $n - 1$  giving you information about the  $n - 1$  cluster merges which it needs to pairwise merge  $n$  clusters. `single_link[i]` will tell us which clusters were merged in the  $i$ -th iteration, let's take a look at the first two points that were merged:

```
In [11]: single_link[0]
```

```
Out[11]: array([ 2.      ,  3.      ,  0.15085,  2.      ])
```

In its first iteration the linkage algorithm decided to merge the two clusters with indices 2 and 3, as they only had a distance of 0.15085. This created a cluster with a total of 2 samples. > We can see that each row of the resulting array has the format `[idx1, idx2, dist, sample_count]`.

```
In [12]: single_link[1]
```

```
Out[12]: array([ 16.      ,  17.      ,  0.15501,  2.      ])
```

In the second iteration the algorithm decided to merge the clusters (original samples here as well) with indices 16 and 17, which had a distance of 0.15501. This again formed another cluster with a total of 2 samples.

The indices of the clusters until now correspond to our samples. Remember that we had a total of 21 samples, so indices 0 to 20. Let's have a look at the first 20 iterations:

```
In [13]: single_link[:20]
```

```
Out[13]: array([[ 2.      ,  3.      ,  0.15085,  2.      ],
 [ 16.     ,  17.     ,  0.15501,  2.      ],
 [ 18.     ,  22.     ,  0.15533,  3.      ],
 [ 15.     ,  23.     ,  0.16303,  4.      ],
 [ 19.     ,  24.     ,  0.16793,  5.      ],
 [ 14.     ,  25.     ,  0.17501,  6.      ],
 [ 13.     ,  26.     ,  0.18766,  7.      ],
 [  5.     ,  6.      ,  0.19187,  2.      ],
 [ 20.     ,  27.     ,  0.19429,  8.      ],
 [ 12.     ,  29.     ,  0.19888,  9.      ],
 [  7.     ,  8.      ,  0.20598,  2.      ],
 [ 11.     ,  30.     ,  0.20744, 10.      ],
 [  9.     ,  31.     ,  0.21189,  3.      ],
 [ 10.     ,  32.     ,  0.21264, 11.      ],
 [ 33.     ,  34.     ,  0.21416, 14.      ],
 [  4.     ,  28.     ,  0.22341,  3.      ],
 [  0.     ,  1.      ,  0.47931,  2.      ],
 [ 35.     ,  36.     ,  0.68347, 17.      ],
 [ 21.     ,  38.     ,  0.83226, 19.      ],
 [ 37.     ,  39.     ,  1.37181, 21.      ]])
```

We can observe the monotonic increase of the distance. This is also similar to the results we had with our run. Note that our algorithm with 4 clusters ends at the distance of 0.47 ish, whereas the above information pertains to finishing with 1 cluster.

Our Output:

```

first cluster | second cluster | distance 2 | 3 | 0.15085042956518227
15 | 16 | 0.15501250939640307 15 | 16 | 0.155332720189051
14 | 15 | 0.16302722103464817 14 | 15 | 0.1679299964569166
13 | 14 | 0.17501291697817623 12 | 13 | 0.18766276935111553
4 | 5 | 0.19186599490269243 11 | 12 | 0.19429172919524904
10 | 11 | 0.19888079280176854 5 | 6 | 0.20597708099371148
8 | 9 | 0.20743622990434923 5 | 6 | 0.21188849347451605 6 |
7 | 0.2126411284874588 5 | 6 | 0.21415518261316957 3 | 4 |
0.22340895237210173 0 | 1 | 0.47931424457263944

```

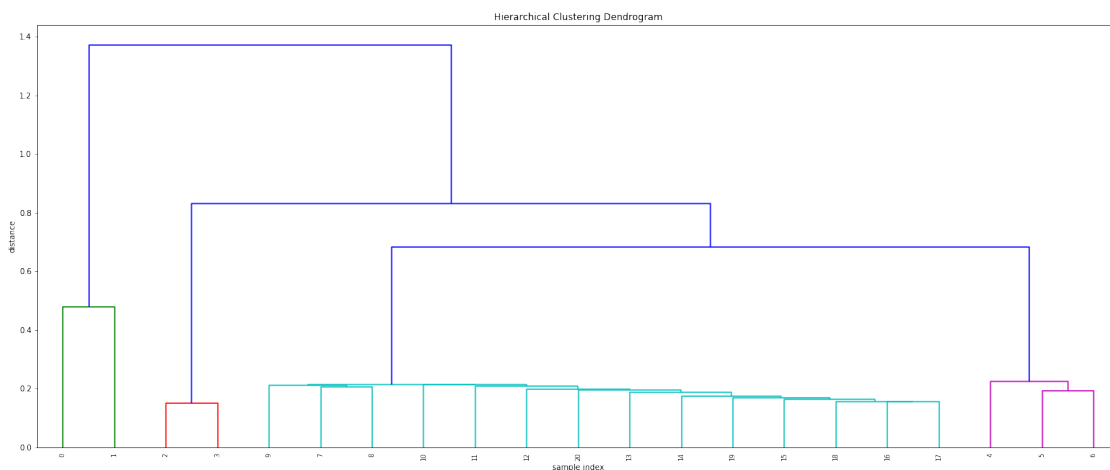
### 3.1 Dendrogram

A dendrogram is a visualization in form of a tree showing the order and distances of merges during the hierarchical clustering.

```

In [14]: # calculate full dendrogram
plt.figure(figsize=(25, 10))
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('sample index')
plt.ylabel('distance')
dendrogram(
    single_link,
    leaf_rotation=90., # rotates the x axis labels
    leaf_font_size=8., # font size for the x axis labels
    color_threshold= .6
)
plt.show()

```



Which actually corresponds to our results as well (pasted from top again)

```

In [15]: clusters = hierarchical(c1,4)
         colors = ['blue', 'red', 'purple', 'teal']

```

```

for cluster_index, cluster in enumerate(clusters):
    for point_index, point in enumerate(cluster):
        plt.plot([point[0]], [point[1]], marker='o', markersize=3, color=colors[cluster_index])

```

first cluster	second cluster	distance
2	3	0.15085042956518227
15	16	0.15501250939640307
15	16	0.155332720189051
14	15	0.16302722103464817
14	15	0.1679299964569166
13	14	0.17501291697817623
12	13	0.18766276935111553
4	5	0.19186599490269243
11	12	0.19429172919524904
10	11	0.19888079280176854
5	6	0.20597708099371148
8	9	0.20743622990434923
5	6	0.21188849347451605
6	7	0.2126411284874588
5	6	0.21415518261316957
3	4	0.22340895237210173
0	1	0.47931424457263944

