# Mean-Link

February 28, 2018

## 1 TARUN SUNKARANENI'S Hierarchical and Point-Clustering Notebook Pt. 3

```
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

        # Any results you write to the current directory are saved as output.
```

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

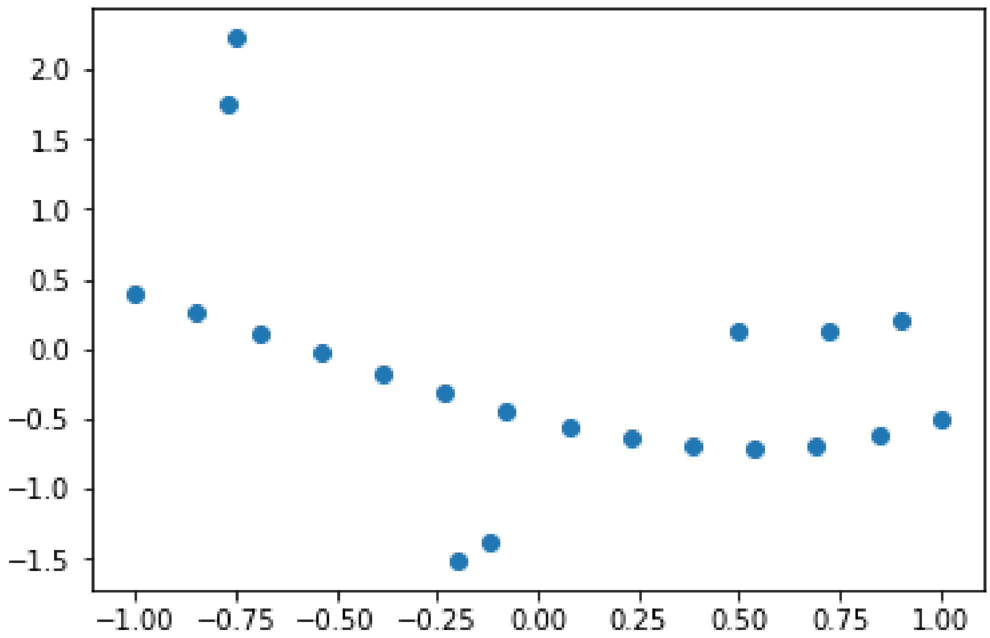
## 2 Mean-Link Hierarchical

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

First compute $a_1 = \frac{1}{|S_1|} \sum_{s \in S_1} s$ and
$a_2 = \frac{1}{|S_2|} \sum_{s \in S_2} s$ then
$\mathbf{d}(S_1, S_2) = \|a_1 - a_2\|_2$

```
In [7]: plt.scatter(c1['x0'],c1['x1'])
```

```
Out[7]: <matplotlib.collections.PathCollection at 0x11a379eb8>
```




```
In [8]: def avg(cluster):
            if len(cluster) < 0:
                return
            current_sum = cluster[0]
            for i in range(1,len(cluster)):
                current_sum = np.add(current_sum , cluster[i])
            # Divide by total samples
            for k in range(len(current_sum)):
                current_sum[k]  = current_sum[k]/len(cluster)
            return current_sum
```

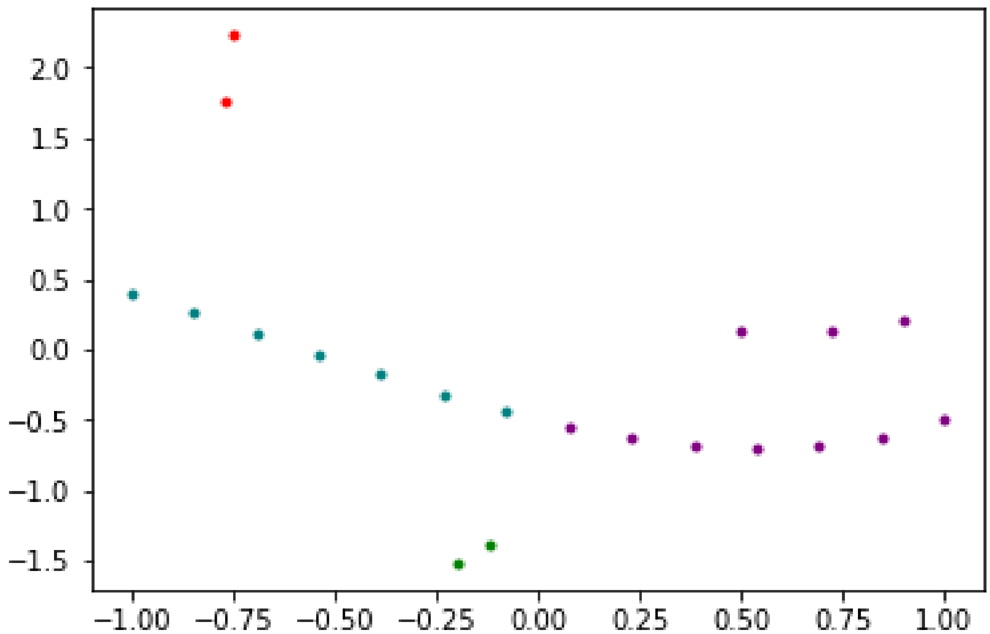
```
In [9]: def mean_distance(clusters ,cluster_num):
            print('first cluster | ','second cluster | ', 'distance')
            while len(clusters) is not cluster_num:
                # Clustering           (
                closest_distance=clust_1=clust_2 = math.inf
                # for every cluster (until second last element)
                for cluster_id, cluster in enumerate(clusters[:len(clusters)]):
                    cluster_avg = avg(cluster)
                    for cluster2_id, cluster2 in enumerate(clusters[(cluster_id+1):]):
                        cluster2_avg = avg (cluster2)
                        if distance.euclidean(cluster_avg,cluster2_avg) < closest_distance:
                            closest_distance = distance.euclidean(cluster_avg,cluster2_avg)
                            clust_1 = cluster_id
                            clust_2 = cluster2_id+cluster_id+1
                print(clust_1,' | ',clust_2, ' | ',closest_distance)
```

```
            clusters[clust_1].extend(clusters[clust_2])
            clusters.pop(clust_2)
        return(clusters)
```

In [10]:
```
### Hierarchical clustering
def hierarchical(data, cluster_num, metric = 'mean'):
    # 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 'mean':
        return mean_distance(init_clusters, cluster_num)
```

In [11]:
```
clusters = hierarchical(c1,4)
colors = ['red', 'green', '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[cluste
```

```
first cluster |  second cluster |  distance
2  |  3  |  0.15085042956518227
15  |  16  |  0.15501250939640307
16  |  17  |  0.1679299964569166
13  |  14  |  0.17501291697817623
4  |  5  |  0.19186599490269243
10  |  11  |  0.19888079280176854
5  |  6  |  0.20597708099371148
7  |  8  |  0.2126411284874588
11  |  12  |  0.27650721583963234
3  |  4  |  0.3141928388744722
4  |  5  |  0.31484413939764316
7  |  8  |  0.32516101556436616
5  |  6  |  0.41304544834321805
0  |  1  |  0.47931424457263944
5  |  6  |  0.5402407941042792
3  |  4  |  0.7368074545203777
2  |  4  |  0.7941786051376811
```




## 3 Validation

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

```
In [12]: X = c1.as_matrix()
         # generate the linkage matrix
         mean_link = linkage(X, 'average') # 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 [13]: from scipy.cluster.hierarchy import cophenet
         from scipy.spatial.distance import pdist

In [14]: c, coph_dists = cophenet(mean_link, pdist(X))
         c

Out[14]: 0.86708962296996617
```

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. `mean_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 [15]: mean_link[0]
```

```
Out[15]: 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 [16]: mean_link[1]
```

```
Out[16]: 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 [17]: mean_link[:20]
```

```
Out[17]: array([[  2.     ,   3.     ,   0.15085,   2.     ],
                [ 16.     ,  17.     ,   0.15501,   2.     ],
                [ 18.     ,  19.     ,   0.16793,   2.     ],
                [ 14.     ,  15.     ,   0.17501,   2.     ],
                [  5.     ,   6.     ,   0.19187,   2.     ],
                [ 12.     ,  13.     ,   0.19888,   2.     ],
                [  7.     ,   8.     ,   0.20598,   2.     ],
                [ 10.     ,  11.     ,   0.21264,   2.     ],
                [ 20.     ,  23.     ,   0.27691,   3.     ],
                [  9.     ,  27.     ,   0.31485,   3.     ],
                [  4.     ,  25.     ,   0.31538,   3.     ],
                [ 22.     ,  24.     ,   0.32588,   4.     ],
                [ 26.     ,  28.     ,   0.41308,   4.     ],
                [  0.     ,   1.     ,   0.47931,   2.     ],
                [ 29.     ,  32.     ,   0.54894,   7.     ],
                [ 30.     ,  33.     ,   0.73684,   7.     ],
                [ 31.     ,  35.     ,   0.8642 ,  10.     ],
                [ 36.     ,  37.     ,   1.26468,  17.     ],
                [ 21.     ,  38.     ,   1.39423,  19.     ],
                [ 34.     ,  39.     ,   2.58   ,  21.     ]])
```

We can observe the monotonic increase of the distance. This is also similar to the results we had with our run. Note that out algorithm with `4` clusters ends at the distance of `0.79` ish, whereas the above information pertains to finishing with `1` cluster.

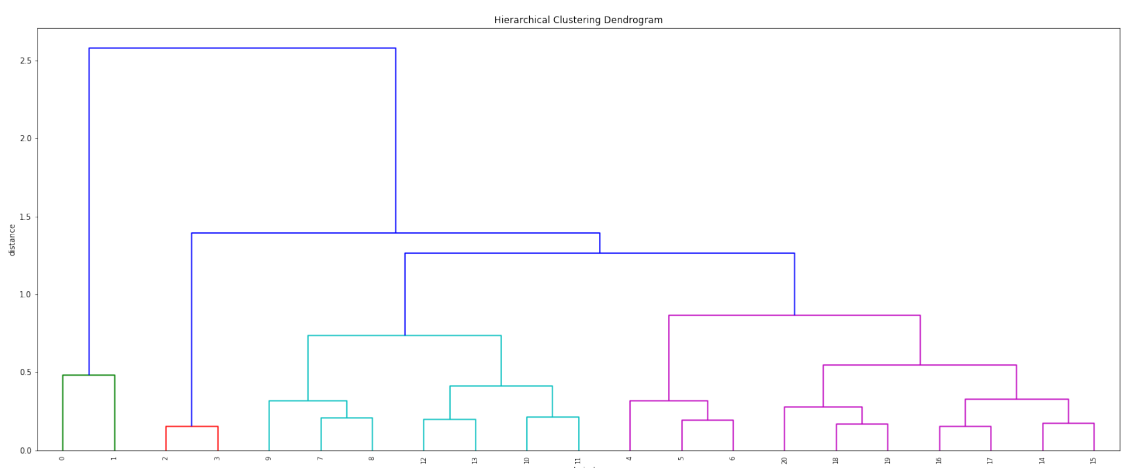
Our Output:

```
first cluster |  second cluster |  distance 2  |  3  |  0.15085042956518227
15  |  16  |  0.15501250939640307 16  |  17  |  0.1679299964569166
13  |  14  |  0.17501291697817623 4  |  5  |  0.19186599490269243
10  |  11  |  0.19888079280176854 5  |  6  |  0.20597708099371148
7  |  8  |  0.2126411284874588 11  |  12  |  0.27650721583963234 3
|  4  |  0.3141928388744722 4  |  5  |  0.31484413939764316 7  |  8
|  0.32516101556436616 5  |  6  |  0.41304544834321805 0  |  1  |
0.47931424457263944 5  |  6  |  0.5402407941042792 3  |  4  |  0.7368074545203777
2  |  4  |  0.7941786051376811
```

## 3.1 Dendogram

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

```
In [18]: # calculate full dendrogram
         plt.figure(figsize=(25, 10))
         plt.title('Hierarchical Clustering Dendrogram')
         plt.xlabel('sample index')
         plt.ylabel('distance')
         dendrogram(
             mean_link,
             leaf_rotation=90.,  # rotates the x axis labels
             leaf_font_size=8.,  # font size for the x axis labels
             color_threshold= 1
         )
         plt.show()
```




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

```
In [19]: clusters = hierarchical(c1,4)
         colors = ['red', 'green', '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[cluste
```

```
first cluster |  second cluster |  distance
2  |  3  |  0.15085042956518227
15  |  16  |  0.15501250939640307
16  |  17  |  0.1679299964569166
13  |  14  |  0.17501291697817623
4  |  5  |  0.19186599490269243
10  |  11  |  0.19888079280176854
5  |  6  |  0.20597708099371148
7  |  8  |  0.2126411284874588
11  |  12  |  0.27650721583963234
3  |  4  |  0.3141928388744722
4  |  5  |  0.31484413939764316
7  |  8  |  0.32516101556436616
5  |  6  |  0.41304544834321805
0  |  1  |  0.47931424457263944
5  |  6  |  0.5402407941042792
3  |  4  |  0.7368074545203777
2  |  4  |  0.7941786051376811
```