Hierarchical Clustering Lab

June 29, 2018

1 Hierarchical Clustering Lab

In this notebook, we will be using sklearn to conduct hierarchical clustering on the Iris dataset which contains 4 dimensions/attributes and 150 samples. Each sample is labeled as one of the three type of Iris flowers.

In this exercise, we'll ignore the labeling and cluster based on the attributes, then we'll compare the results of different hierarchical clustering techniques with the original labels to see which one does a better job in this scenario. We'll then proceed to visualize the resulting cluster hierarchies.

1.1 1. Importing the Iris dataset

```
In [6]: from sklearn import datasets

        iris = datasets.load_iris()
```

A look at the first 10 samples in the dataset

```
In [7]: iris.data[:10]

Out[7]: array([[ 5.1,  3.5,  1.4,  0.2],
               [ 4.9,  3. ,  1.4,  0.2],
               [ 4.7,  3.2,  1.3,  0.2],
               [ 4.6,  3.1,  1.5,  0.2],
               [ 5. ,  3.6,  1.4,  0.2],
               [ 5.4,  3.9,  1.7,  0.4],
               [ 4.6,  3.4,  1.4,  0.3],
               [ 5. ,  3.4,  1.5,  0.2],
               [ 4.4,  2.9,  1.4,  0.2],
               [ 4.9,  3.1,  1.5,  0.1]])
```

`iris.target` contains the labels that indicate which type of Iris flower each sample is

```
In [8]: iris.target

Out[8]: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
               1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

1.2 2. Clustering

Let's now use sklearn's AgglomerativeClustering to conduct the heirarchical clustering

```
In [9]: from sklearn.cluster import AgglomerativeClustering

        # Hierarchical clustering
        # Ward is the default linkage algorithm, so we'll start with that
        ward = AgglomerativeClustering(n_clusters=3)
        ward_pred = ward.fit_predict(iris.data)
```

Let's also try complete and average linkages

Exercise: * Conduct hierarchical clustering with complete linkage, store the predicted labels in the variable `complete_pred` * Conduct hierarchical clustering with average linkage, store the predicted labels in the variable `avg_pred`

Note: look at the documentation of AgglomerativeClustering to find the appropriate value to pass as the `linkage` value

```
In [11]: # Hierarchical clustering using complete linkage
         # TODO: Create an instance of AgglomerativeClustering with the appropriate parameters
         complete = AgglomerativeClustering(n_clusters = 3, linkage = 'complete')
         # Fit & predict
         # TODO: Make AgglomerativeClustering fit the dataset and predict the cluster labels
         complete_pred = complete.fit_predict(iris.data)

         # Hierarchical clustering using average linkage
         # TODO: Create an instance of AgglomerativeClustering with the appropriate parameters
         avg = AgglomerativeClustering(n_clusters =3, linkage = 'average')
         # Fit & predict
         # TODO: Make AgglomerativeClustering fit the dataset and predict the cluster labels
         avg_pred = avg.fit_predict(iris.data)
```

To determine which clustering result better matches the original labels of the samples, we can use `adjusted_rand_score` which is an *external cluster validation index* which results in a score between -1 and 1, where 1 means two clusterings are identical of how they grouped the samples in a dataset (regardless of what label is assigned to each cluster).

Cluster validation indices are discussed later in the course.

```
In [12]: from sklearn.metrics import adjusted_rand_score

         ward_ar_score = adjusted_rand_score(iris.target, ward_pred)
```

Exercise: * Calculate the Adjusted Rand score of the clusters resulting from complete linkage and average linkage

```
In [14]: # TODO: Calculated the adjusted Rand score for the complete linkage clustering labels
         complete_ar_score = adjusted_rand_score(iris.target, complete_pred)

         # TODO: Calculated the adjusted Rand score for the average linkage clustering labels
         avg_ar_score = adjusted_rand_score(iris.target, avg_pred)
```

Which algorithm results in the higher Adjusted Rand Score?

```
In [15]: print( "Scores: \nWard:", ward_ar_score,"\nComplete: ", complete_ar_score, "\nAverage:

Scores:
Ward: 0.731198556771
Complete:  0.642251251836
Average:  0.759198707107
```

1.3 3. The Effect of Normalization on Clustering

Can we improve on this clustering result?

Let's take another look at the dataset

```
In [16]: iris.data[:15]

Out[16]: array([[ 5.1,  3.5,  1.4,  0.2],
                [ 4.9,  3. ,  1.4,  0.2],
                [ 4.7,  3.2,  1.3,  0.2],
                [ 4.6,  3.1,  1.5,  0.2],
                [ 5. ,  3.6,  1.4,  0.2],
                [ 5.4,  3.9,  1.7,  0.4],
                [ 4.6,  3.4,  1.4,  0.3],
                [ 5. ,  3.4,  1.5,  0.2],
                [ 4.4,  2.9,  1.4,  0.2],
                [ 4.9,  3.1,  1.5,  0.1],
                [ 5.4,  3.7,  1.5,  0.2],
                [ 4.8,  3.4,  1.6,  0.2],
                [ 4.8,  3. ,  1.4,  0.1],
                [ 4.3,  3. ,  1.1,  0.1],
                [ 5.8,  4. ,  1.2,  0.2]])
```

Looking at this, we can see that the forth column has smaller values than the rest of the columns, and so its variance counts for less in the clustering process (since clustering is based on distance). Let us normalize the dataset so that each dimension lies between 0 and 1, so they have equal weight in the clustering process.

This is done by subtracting the minimum from each column then dividing the difference by the range.

sklearn provides us with a useful utility called `preprocessing.normalize()` that can do that for us

```
In [17]: from sklearn import preprocessing

         normalized_X = preprocessing.normalize(iris.data)
         normalized_X[:10]

Out[17]: array([[ 0.80377277,  0.55160877,  0.22064351,  0.0315205 ],
                [ 0.82813287,  0.50702013,  0.23660939,  0.03380134],
                [ 0.80533308,  0.54831188,  0.2227517 ,  0.03426949],
                [ 0.80003025,  0.53915082,  0.26087943,  0.03478392],
                [ 0.790965  ,  0.5694948 ,  0.2214702 ,  0.0316386 ],
                [ 0.78417499,  0.5663486 ,  0.2468699 ,  0.05808704],
                [ 0.78010936,  0.57660257,  0.23742459,  0.0508767 ],
                [ 0.80218492,  0.54548574,  0.24065548,  0.0320874 ],
                [ 0.80642366,  0.5315065 ,  0.25658935,  0.03665562],
                [ 0.81803119,  0.51752994,  0.25041771,  0.01669451]])
```

Now all the columns are in the range between 0 and 1. Would clustering the dataset after this transformation lead to a better clustering? (one that better matches the original labels of the samples)

```
In [18]: ward = AgglomerativeClustering(n_clusters=3)
         ward_pred = ward.fit_predict(normalized_X)

         complete = AgglomerativeClustering(n_clusters=3, linkage="complete")
         complete_pred = complete.fit_predict(normalized_X)

         avg = AgglomerativeClustering(n_clusters=3, linkage="average")
         avg_pred = avg.fit_predict(normalized_X)


         ward_ar_score = adjusted_rand_score(iris.target, ward_pred)
         complete_ar_score = adjusted_rand_score(iris.target, complete_pred)
         avg_ar_score = adjusted_rand_score(iris.target, avg_pred)

         print( "Scores: \nWard:", ward_ar_score,"\nComplete: ", complete_ar_score, "\nAverage:

Scores:
Ward: 0.885697031028
Complete:  0.644447235392
Average:  0.558371443754
```

1.4 4. Dendrogram visualization with scipy

Let's visualize the highest scoring clustering result.

To do that, we'll need to use Scipy's `linkage` function to perform the clusteirng again so we can obtain the linkage matrix it will later use to visualize the hierarchy

```
In [19]: # Import scipy's linkage function to conduct the clustering
         from scipy.cluster.hierarchy import linkage

         # Specify the linkage type. Scipy accepts 'ward', 'complete', 'average', as well as oth
         # Pick the one that resulted in the highest Adjusted Rand Score
         linkage_type = 'ward'

         linkage_matrix = linkage(normalized_X, linkage_type)
```

Plot using scipy's dendrogram function

```
In [20]: from scipy.cluster.hierarchy import dendrogram
         import matplotlib.pyplot as plt
         plt.figure(figsize=(22,18))

         # plot using 'dendrogram()'
         dendrogram(linkage_matrix)

         plt.show()
```

```
<matplotlib.figure.Figure at 0x7faff10bf860>
```

1.5 5. Visualization with Seaborn's `clustermap`

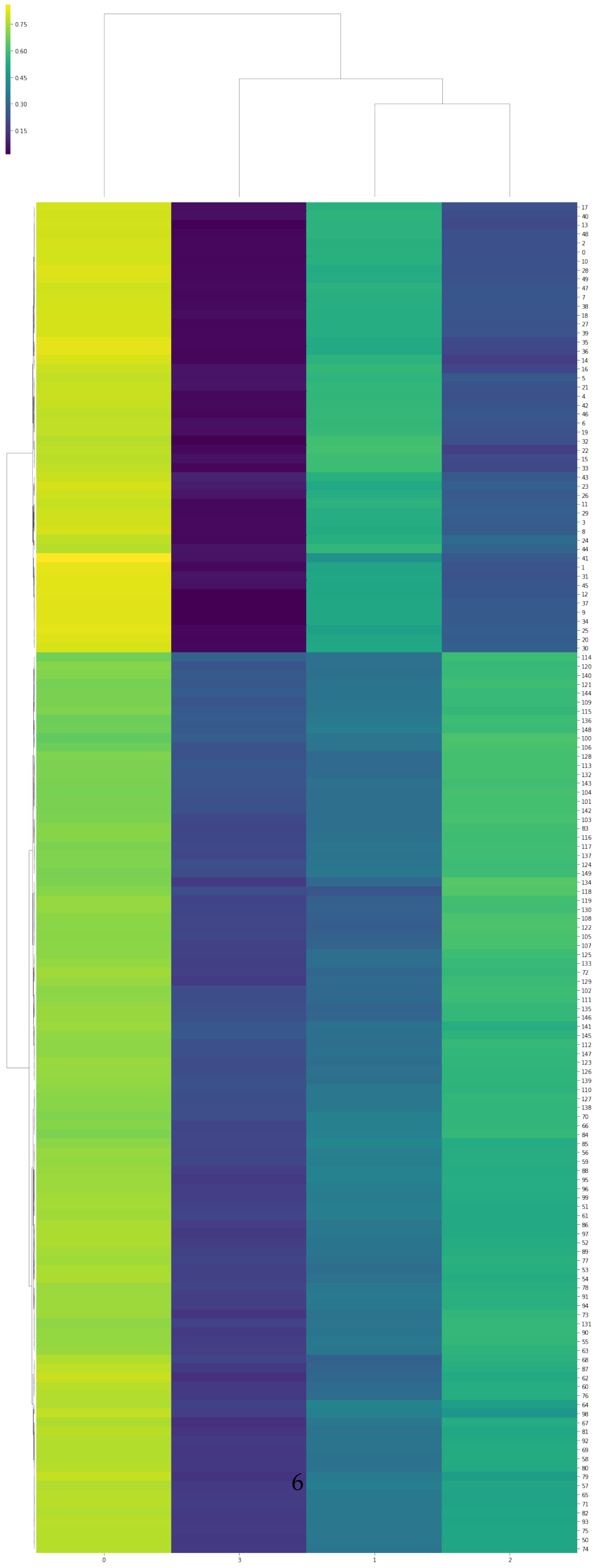
The seaborn plotting library for python can plot a clustermap, which is a detailed dendrogram which also visualizes the dataset in more detail. It conducts the clustering as well, so we only need to pass it the dataset and the linkage type we want, and it will use scipy internally to conduct the clustering

```
In [22]: import seaborn as sns

         sns.clustermap(normalized_X, figsize=(18,50), method=linkage_type, cmap='viridis')

         # Expand figsize to a value like (18, 50) if you want the sample labels to be readable
         # Draw back is that you'll need more scrolling to observe the dendrogram

         plt.show()
```


0.75
0.60
0.45
0.30
0.15
0
3
1
2
17
40
13
48
2
0
10
28
49
47
7
38
18
27
39
35
36
14
16
5
21
4
42
46
6
19
32
22
15
33
43
23
26
11
29
3
8
24
44
41
1
31
45
12
37
9
34
25
20
30
114
120
140
121
144
109
115
136
148
100
106
128
113
132
143
104
101
142
103
83
116
117
137
124
149
134
118
119
130
108
122
105
107
125
133
72
129
102
111
135
146
141
145
112
147
123
126
139
110
127
138
70
66
84
85
56
59
88
95
96
99
51
61
86
97
52
89
77
53
54
78
91
94
73
131
90
55
63
68
87
62
60
76
64
98
67
81
92
69
58
80
79
57
65
71
82
93
75
50
74

Looking at the colors of the dimensions can you observe how they differ between the three type of flowers? You should at least be able to notice how one is vastly different from the two others (in the top third of the image).