04_hierarchical_clustering

September 29, 2021

1 Hierarchical Clustering

Hierarchical clustering avoids the need to specify a target number of clusters because it assumes that data can successively be merged into increasingly dissimilar clusters. It does not pursue a global objective but decides incrementally how to produce a sequence of nested clusters that range from a single cluster to clusters consisting of the individual data points.

While hierarchical clustering does not have hyperparameters like k-Means, the measure of dissimilarity between clusters (as opposed to individual data points) has an important impact on the clustering result. The options differ as follows:

- Single-link: distance between nearest neighbors of two clusters
- Complete link: maximum distance between respective cluster members
- Group average
- Ward's method: minimize within-cluster variance

1.1 Imports & Settings

[1]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from scipy.cluster.hierarchy import dendrogram, linkage, cophenet
from sklearn.decomposition import PCA
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import adjusted_mutual_info_score
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris
from matplotlib.animation import FuncAnimation
from scipy.spatial.distance import pdist
from IPython.display import HTML
```

[2]:

```
# if you have difficulties with ffmpeg to run the simulation, see https://
 ↪stackoverflow.com/questions/13316397/
 ↪matplotlib-animation-no-moviewriters-available
# plt.rcParams['animation.ffmpeg_path'] = your_windows_path
plt.rcParams['animation.ffmpeg_args'] = '-report'
plt.rcParams['animation.bitrate'] = 2000
```

```
sns.set_style('whitegrid')
pd.options.display.float_format = '{:,.2f}'.format
```

1.2 Load Iris Data

[3]:

```
iris = load_iris()
iris.keys()
```

[3]:

```
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names',
'filename'])
```

[4]:

```
print(iris.DESCR)
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken
from Fisher's paper. Note that it's the same as in R, but not as in the UCI
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field and
is referenced frequently to this day.  (See Duda & Hart, for example.)  The
data set contains 3 classes of 50 instances each, where each class refers to a
type of iris plant.  One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.

.. topic:: References

   - Fisher, R.A. "The use of multiple measurements in taxonomic problems"
     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
     Mathematical Statistics" (John Wiley, NY, 1950).
   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.
     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
     Structure and Classification Rule for Recognition in Partially Exposed
     Environments".  IEEE Transactions on Pattern Analysis and Machine
     Intelligence, Vol. PAMI-2, No. 1, 67-71.
   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions
     on Information Theory, May 1972, 431-433.
   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
     conceptual clustering system finds 3 classes in the data.
   - Many, many more …
```

1.3 Create DataFrame

[5]:

```
features = iris.feature_names
data = pd.DataFrame(data=np.column_stack([iris.data, iris.target]),
                    columns=features + ['label'])
data.label = data.label.astype(int)
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
sepal length (cm)    150 non-null float64
sepal width (cm)     150 non-null float64
petal length (cm)    150 non-null float64
petal width (cm)     150 non-null float64
label                150 non-null int64
dtypes: float64(4), int64(1)
memory usage: 5.9 KB
```

1.3.1 Standardize Data

The use of a distance metric makes hierarchical clustering sensitive to scale:

[6]:

```
scaler = StandardScaler()
features_standardized = scaler.fit_transform(data[features])
n = len(data)
```

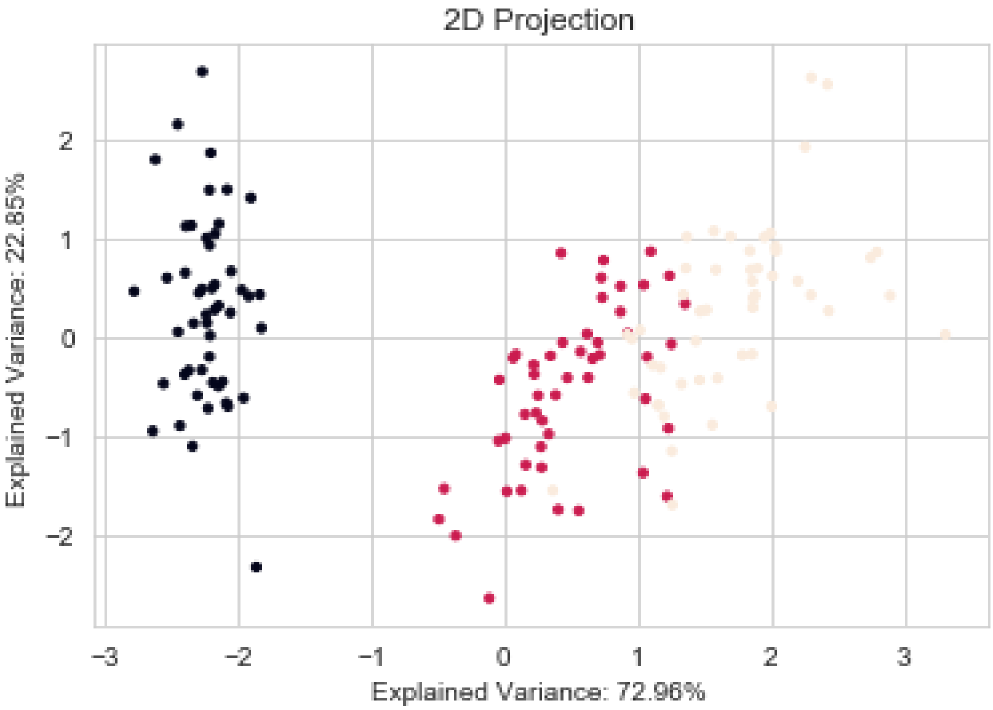
1.3.2 Reduce Dimensionality to visualize clusters

[7]:

```
pca = PCA(n_components=2)
features_2D = pca.fit_transform(features_standardized)
```

[8]:

```
ev1, ev2 = pca.explained_variance_ratio_
ax = plt.figure().gca(title='2D Projection',
                      xlabel='Explained Variance: {:.2%}'.format(ev1),
                      ylabel='Explained Variance: {:.2%}'.format(ev2))
ax.scatter(*features_2D.T, c=data.label, s=10);
```



1.3.3 Perform agglomerative clustering

```
[9]: Z = linkage(features_standardized, 'ward')
     Z[:5]
```

```
[9]: array([[1.01000000e+02, 1.42000000e+02, 0.00000000e+00, 2.00000000e+00],
            [7.00000000e+00, 3.90000000e+01, 1.21167870e-01, 2.00000000e+00],
            [1.00000000e+01, 4.80000000e+01, 1.21167870e-01, 2.00000000e+00],
            [9.00000000e+00, 3.40000000e+01, 1.31632184e-01, 2.00000000e+00],
            [0.00000000e+00, 1.70000000e+01, 1.31632184e-01, 2.00000000e+00]])
```

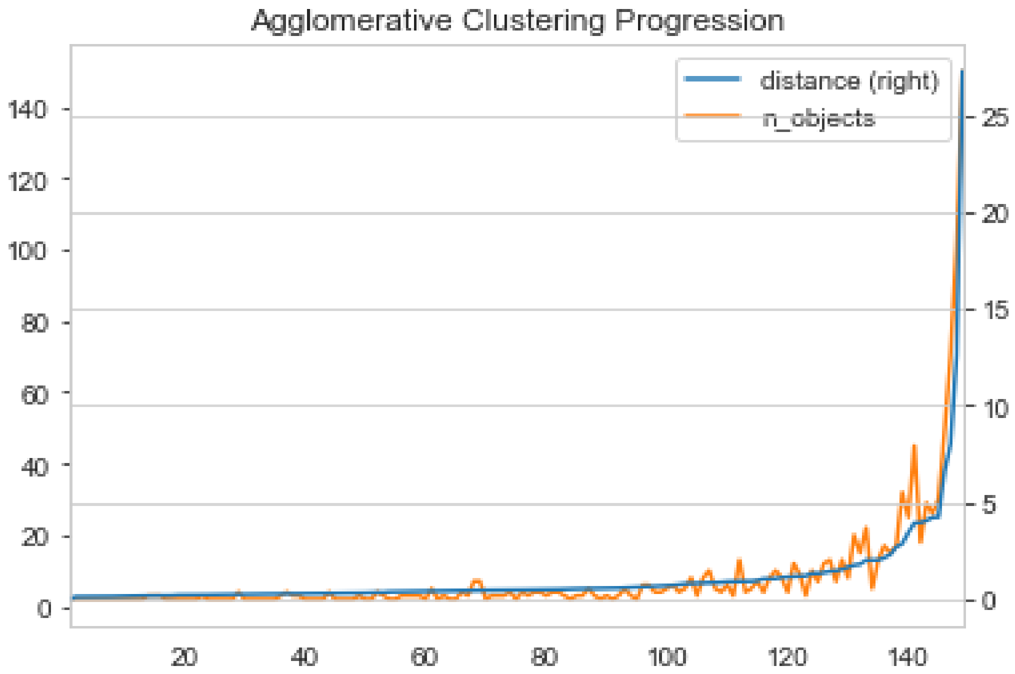
```
[10]: linkage_matrix = pd.DataFrame(data=Z,
                                    columns=['cluster_1', 'cluster_2',
                                             'distance', 'n_objects'],
                                    index=range(1, n))
      for col in ['cluster_1', 'cluster_2', 'n_objects']:
          linkage_matrix[col] = linkage_matrix[col].astype(int)
      linkage_matrix.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 149 entries, 1 to 149
Data columns (total 4 columns):
cluster_1    149 non-null int64
cluster_2    149 non-null int64
distance     149 non-null float64
n_objects    149 non-null int64
dtypes: float64(1), int64(3)
memory usage: 4.7 KB
```

```
[11]: linkage_matrix.head()
```

```
[11]:    cluster_1  cluster_2  distance  n_objects
      1        101        142      0.00          2
      2          7         39      0.12          2
      3         10         48      0.12          2
      4          9         34      0.13          2
      5          0         17      0.13          2
```

```
[12]: linkage_matrix[['distance', 'n_objects']].plot(secondary_y=['distance'],
                                 title='Agglomerative Clustering Progression');
```



1.3.4 Compare linkage types

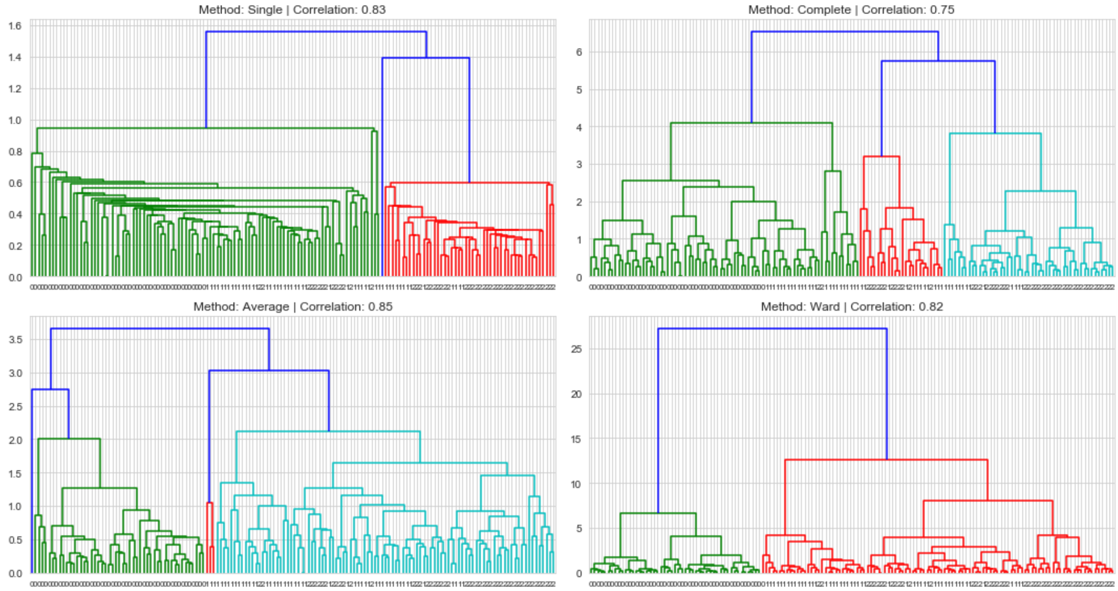
Hierarchical clustering provides insight into degrees of similarity among observations as it continues to merge data. A significant change in the similarity metric from one merge to the next suggests a natural clustering existed prior to this point. The dendrogram visualizes the successive merges as a binary tree, displaying the individual data points as leaves and the final merge as the root of the tree. It also shows how the similarity monotonically decreases from bottom to top. Hence, it is natural to select a clustering by cutting the dendrogram.

The following figure illustrates the dendrogram for the classic Iris dataset with four classes and three features using the four different distance metrics introduced above. It evaluates the fit of the hierarchical clustering using the cophenetic correlation coefficient that compares the pairwise distances among points and the cluster similarity metric at which a pairwise merge occurred. A coefficient of 1 implies that closer points always merge earlier.

[13]:
```
methods = ['single', 'complete', 'average', 'ward']
pairwise_distance = pdist(features_standardized)
```

[14]:
```
fig, axes = plt.subplots(figsize=(15, 8), nrows=2, ncols=2, sharex=True)
axes = axes.flatten()
for i, method in enumerate(methods):
    Z = linkage(features_standardized, method)
    c, coph_dists = cophenet(Z, pairwise_distance)
    dendrogram(Z, labels=data.label.values,
        orientation='top', leaf_rotation=0.,
        leaf_font_size=8., ax = axes[i])
    axes[i].set_title('Method: {} | Correlation: {:.2f}'.format(
                                                method.capitalize(), c))

fig.tight_layout()
fig.savefig('dendrogram', dpi=300)
```



Different linkage methods produce different dendrogram 'looks' so that we can not use this visualization to compare results across methods. In addition, the Ward method that minimizes the within-cluster variance may not properly reflect the change in variance but the total variance that may be misleading. Instead, other quality metrics like the cophenetic correlation or measures like inertia if aligned with the overall goal are more appropriate.

1.3.5 Get Cluster Members

[15]:
```
n = len(Z)
from collections import OrderedDict
clusters = OrderedDict()

for i, row in enumerate(Z, 1):
    cluster = []
    for c in row[:2]:
        if c <= n:
            cluster.append(int(c))
        else:
            cluster += clusters[int(c)]
    clusters[n+i] = cluster
```

[16]:
```
clusters[230]
```

[16]:
```
[144, 124, 120, 143]
```

1.3.6 Animate Agglomerative Clustering

[17]:
```
def get_2D_coordinates():
    points = pd.DataFrame(features_2D).assign(n=1)
    return dict(enumerate(points.values.tolist()))
```

[18]:
```
n_clusters = Z.shape[0]
points = get_2D_coordinates()
cluster_states = {0: get_2D_coordinates()}

for i, cluster in enumerate(Z[:, :2], 1):
    cluster_state = dict(cluster_states[i-1])
    merged_points = np.array([cluster_state.pop(c) for c in cluster])
    cluster_size = merged_points[:, 2]
    new_point = np.average(merged_points[:, :2],
                           axis=0, weights=cluster_size).tolist()
    new_point.append(cluster_size.sum())
    cluster_state[n_clusters+i] = new_point
    cluster_states[i] = cluster_state
```

[19]:
```
cluster_states[100]
```

[19]:
```
{15: [-2.2622145316010216, 2.686284485110592, 1.0],
 41: [-1.8581224563735717, -2.337415157553348, 1.0],
 60: [-0.11019628000062932, -2.6540728185365645, 1.0],
 62: [0.5621083064431792, -1.7647243806169446, 1.0],
 106: [0.36701768786072436, -1.5615028914765063, 1.0],
 108: [2.006686467676607, -0.7114386535471587, 1.0],
 109: [2.2597773490125017, 1.9210103764598867, 1.0],
 114: [1.4676452010173247, -0.44227158737708266, 1.0],
 176: [0.28711522075373114, -0.24204262329732348, 2.0],
 179: [-2.2058211672986987, -0.09844552008358048, 2.0],
 185: [0.9688104990917799, 0.01907915384081543, 3.0],
 194: [0.7389335833626245, 0.6838170188840107, 2.0],
 197: [2.3691965862739517, 0.33967500639352843, 2.0],
 198: [-2.4657912495858, -0.9936714816693262, 3.0],
 205: [1.9290326043911596, 0.5634757376107679, 3.0],
 206: [-1.9303546578375999, 0.4980362956076001, 3.0],
 208: [-2.3324410024278306, 0.10599232941089061, 3.0],
 209: [2.365628354573734, 2.5914923595139356, 2.0],
 210: [-2.3753778947842483, -0.428476399606261, 5.0],
 211: [2.775321891622194, 0.8286062652623241, 2.0],
 212: [1.3703381593353579, -0.4066353802439459, 3.0],
 214: [1.4771264049002877, 0.6845006301068535, 2.0],
 215: [0.6124063740956357, -0.11263115492758463, 4.0],
 217: [-2.107918266740138, -0.5961767325807753, 7.0],
 218: [-2.2074608511148277, 0.39185352432566406, 7.0],
 219: [-2.530562663777726, 1.9721518667486764, 2.0],
 222: [-1.9447561950630023, 0.26668291555609785, 3.0],
 224: [1.1317839552605031, -0.7225792577873079, 2.0],
 225: [0.17415501993333077, -0.9041186723104582, 4.0],
 226: [1.9804041815154314, 0.5524627673500003, 3.0],
 228: [0.5970487156101139, -0.2629983547610023, 4.0],
 229: [1.5978708558106194, 0.9834189460574745, 3.0],
 230: [1.944288880741274, 0.9602696562701191, 4.0],
 231: [1.066723936158902, 0.6275964565840024, 4.0],
 232: [-0.43239104859121186, -1.8051529981646477, 3.0],
 233: [1.3993792903437394, -0.9150062688479023, 2.0],
 236: [0.20202521219193983, -1.5042170574230862, 5.0],
 237: [1.1777972305328468, -1.5706593008861836, 3.0],
 238: [1.9089871295906673, 0.7850341610197987, 2.0],
 239: [-2.5638395941372223, 0.5658239416152503, 3.0],
 240: [0.5834108821437097, 0.6210769014326585, 2.0],
 241: [1.8396217867683873, -0.02362196158643326, 3.0],
 242: [1.4274899699312786, 0.24628913465740437, 5.0],
 243: [0.19303006787186602, -1.003259039619637, 3.0],
 244: [3.1041735373340296, 0.2157110044581378, 2.0],
 245: [0.16678817742501761, -0.402792363931369, 6.0],
 246: [-2.2446551194172795, 1.0593800316735267, 6.0],
 247: [1.1384647649646227, -0.7829411177202094, 4.0],
 248: [-2.0951328107676597, 1.5594656985593047, 4.0],
 249: [1.049661370014735, -0.05911604365815985, 5.0]}
```

1.3.7 Set up Animation

[20]:

```
%%capture
fig, ax = plt.subplots()
xmin, ymin = np.min(features_2D, axis=0) * 1.1
xmax, ymax = np.max(features_2D, axis=0) * 1.1
ax.set(xlim=(xmin, xmax), ylim=(ymin, ymax))
```

[21]:

```
scat = ax.scatter([], [])
def animate(i):
    df = pd.DataFrame(cluster_states[i]).values.T
    scat.set_offsets(df[:, :2])
    scat.set_sizes((df[:, 2] * 2) ** 2)
    return scat,

anim = FuncAnimation(
    fig, animate, frames=cluster_states.keys(), interval=250, blit=False)
```

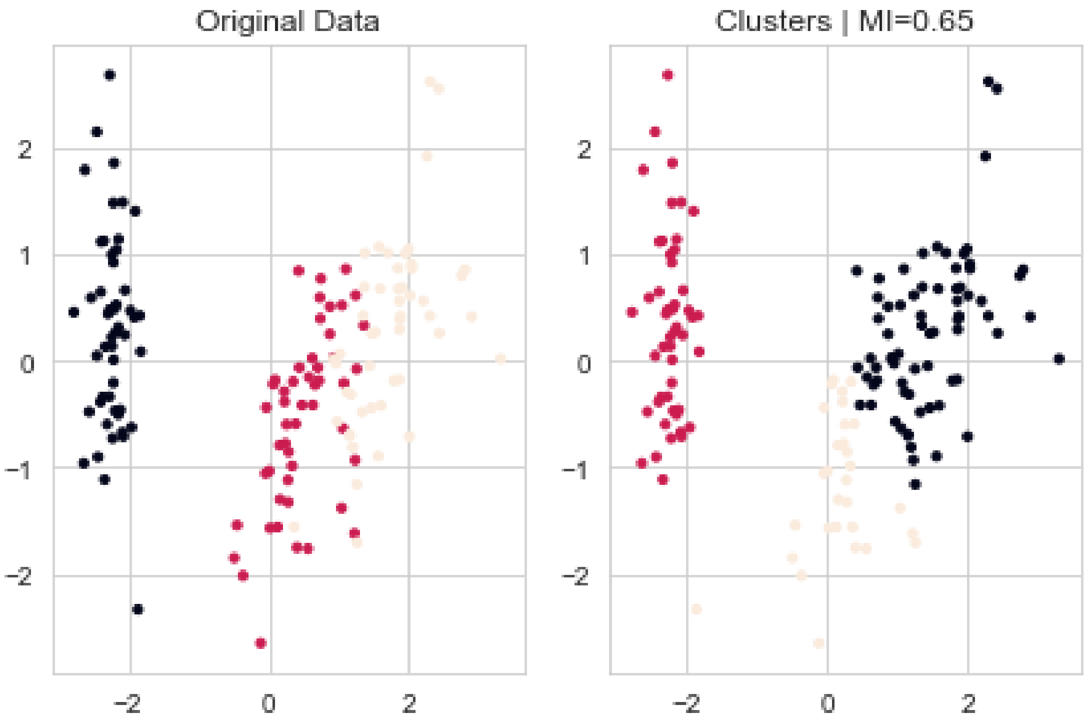
```
HTML(anim.to_html5_video())
```

[21]: <IPython.core.display.HTML object>

1.3.8 Scikit-Learn implementation

[22]:
```
clusterer = AgglomerativeClustering(n_clusters=3)
data['clusters'] = clusterer.fit_predict(features_standardized)
fig, axes = plt.subplots(ncols=2)
labels, clusters = data.label, data.clusters
mi = adjusted_mutual_info_score(labels, clusters)
axes[0].scatter(*features_2D.T, c=data.label, s=10)
axes[0].set_title('Original Data')
axes[1].scatter(*features_2D.T, c=data.clusters, s=10)
axes[1].set_title('Clusters | MI={:.2f}'.format(mi))
plt.tight_layout()
```

```
/home/stefan/.pyenv/versions/miniconda3-latest/envs/ml4t/lib/python3.7/site-
packages/sklearn/metrics/cluster/supervised.py:732: FutureWarning: The behavior
of AMI will change in version 0.22. To match the behavior of 'v_measure_score',
AMI will use average_method='arithmetic' by default.
  FutureWarning)
```

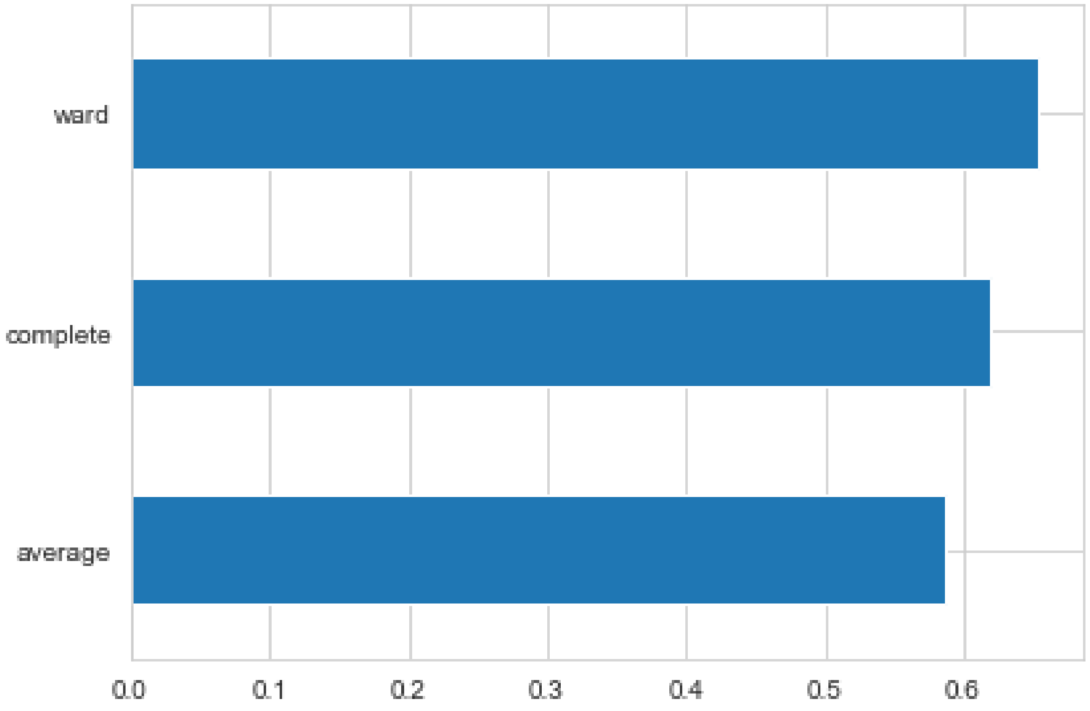


1.3.9 Comparing Mutual Information for different Linkage Options

[23]:

```
mutual_info = {}
for linkage_method in ['ward', 'complete', 'average']:
    clusterer = AgglomerativeClustering(n_clusters=3, linkage=linkage_method)
    clusters = clusterer.fit_predict(features_standardized)
    mutual_info[linkage_method] = adjusted_mutual_info_score(clusters, labels)
fig, ax = plt.subplots()
pd.Series(mutual_info).sort_values().plot.barh()
plt.tight_layout()
```

```
/home/stefan/.pyenv/versions/miniconda3-latest/envs/ml4t/lib/python3.7/site-
packages/sklearn/metrics/cluster/supervised.py:732: FutureWarning: The behavior
of AMI will change in version 0.22. To match the behavior of 'v_measure_score',
AMI will use average_method='arithmetic' by default.
  FutureWarning)
/home/stefan/.pyenv/versions/miniconda3-latest/envs/ml4t/lib/python3.7/site-
packages/sklearn/metrics/cluster/supervised.py:732: FutureWarning: The behavior
of AMI will change in version 0.22. To match the behavior of 'v_measure_score',
AMI will use average_method='arithmetic' by default.
  FutureWarning)
/home/stefan/.pyenv/versions/miniconda3-latest/envs/ml4t/lib/python3.7/site-
packages/sklearn/metrics/cluster/supervised.py:732: FutureWarning: The behavior
of AMI will change in version 0.22. To match the behavior of 'v_measure_score',
AMI will use average_method='arithmetic' by default.
  FutureWarning)
```



1.4 Strengths and Weaknesses

The strengths of hierarchical clustering include that - you do not need to specify the number of clusters but instead offers insight about potential clustering by means of an intuitive visualization. - It produces a hierarchy of clusters that can serve as a taxonomy. - It can be combined with k-means to reduce the number of items at the start of the agglomerative process.

The weaknesses include - the high cost in terms of computation and memory because of the numerous similarity matrix updates. - Another downside is that all merges are final so that it does not achieve the global optimum. - - Furthermore, the curse of dimensionality leads to difficulties with noisy, high-dimensional data.