id5055-tutorial-3

September 6, 2023

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import seaborn as sns
import scipy.cluster.hierarchy as shc
from sklearn import metrics
from sklearn.datasets import make_blobs, make_circles, make_moons
from sklearn.cluster import AgglomerativeClustering, DBSCAN, SpectralClustering
from sklearn.preprocessing import StandardScaler
```

[2]:
```
# Helper function to plot
def plot_clusters(data, true_labels=None, cluster_labels=None, title_true="True␣
 ↪clusters", title_cluster="Agglomerative Clustering"):
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

    ax1.scatter(data[:, 0], data[:, 1], c=true_labels)
    ax1.set_title(title_true)

    if cluster_labels is not None:
        ax2.scatter(data[:, 0], data[:, 1], c=cluster_labels)
        ax2.set_title(title_cluster)

    plt.show()

# Example usage:
# plot_clusters(blob_X, blob_true_labels, agglo_cluster.labels_,␣
 ↪title_cluster="Agglomerative Clustering")
```

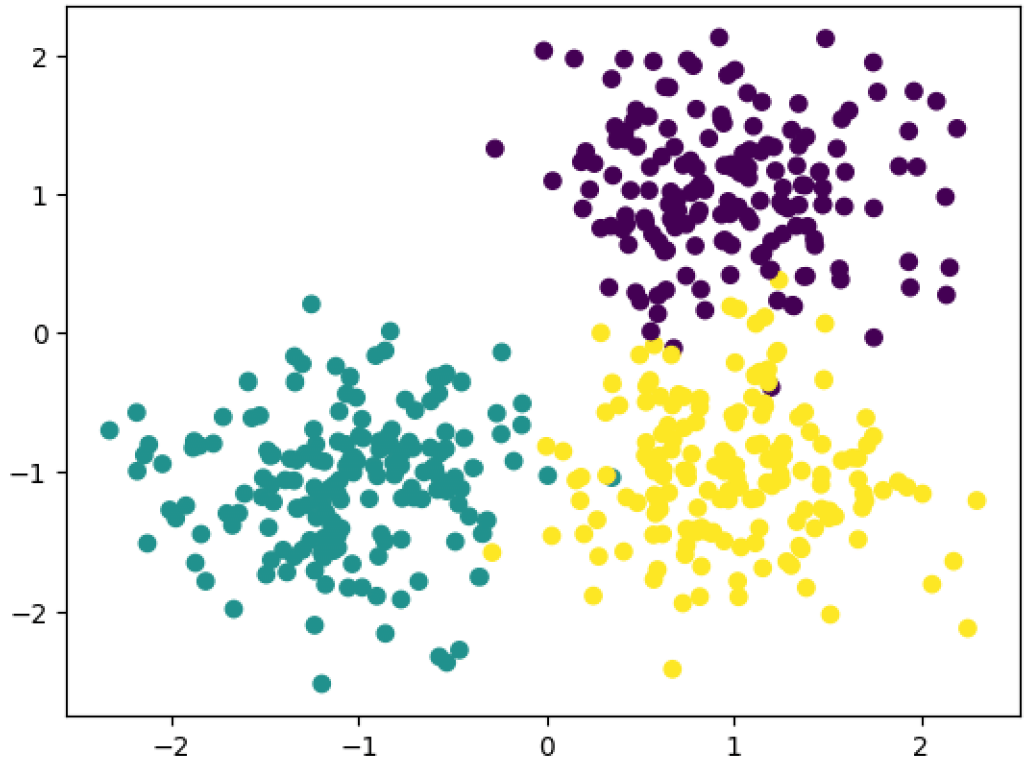
0.0.1 common

[3]:
```
seed = 0
```

[4]:
```
# blob data

centers = [[1, 1], [-1, -1], [1, -1]]
```

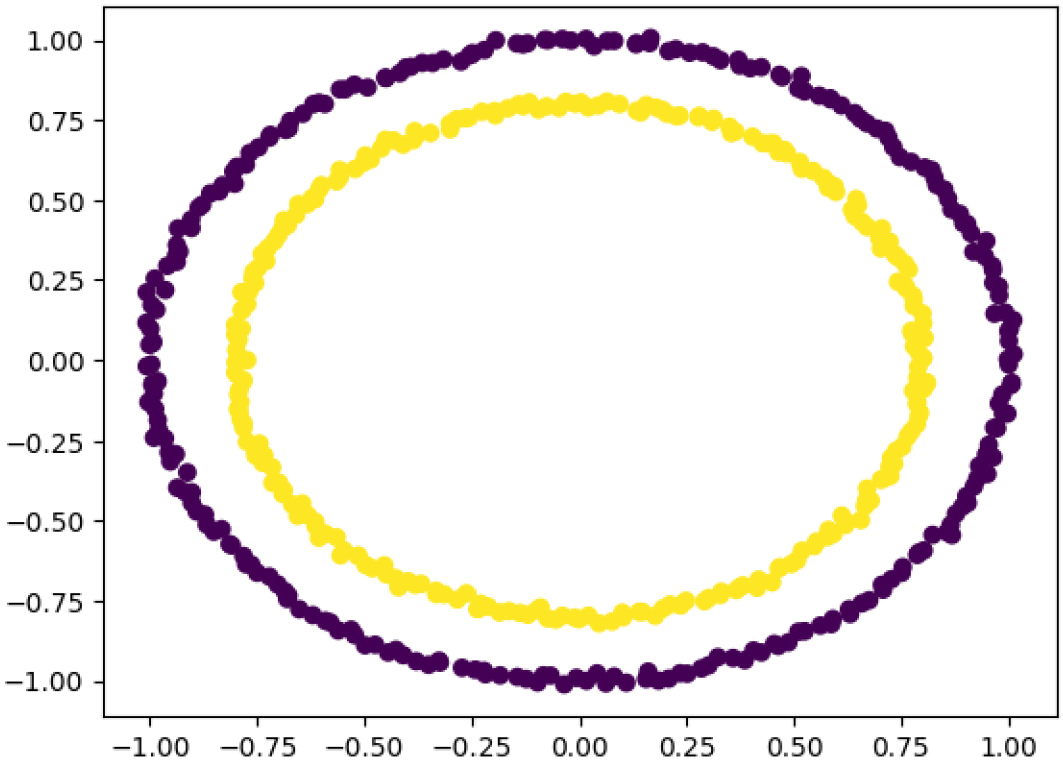
```
blob_X, blob_true_labels = make_blobs(n_samples=500, centers=centers,␣
 ↪cluster_std=0.5, random_state=seed)

scaled_blob_X = StandardScaler().fit_transform(blob_X)

plt.scatter(blob_X[:, 0], blob_X[:, 1], c=blob_true_labels)
```

[4]: <matplotlib.collections.PathCollection at 0x7f58654156f0>



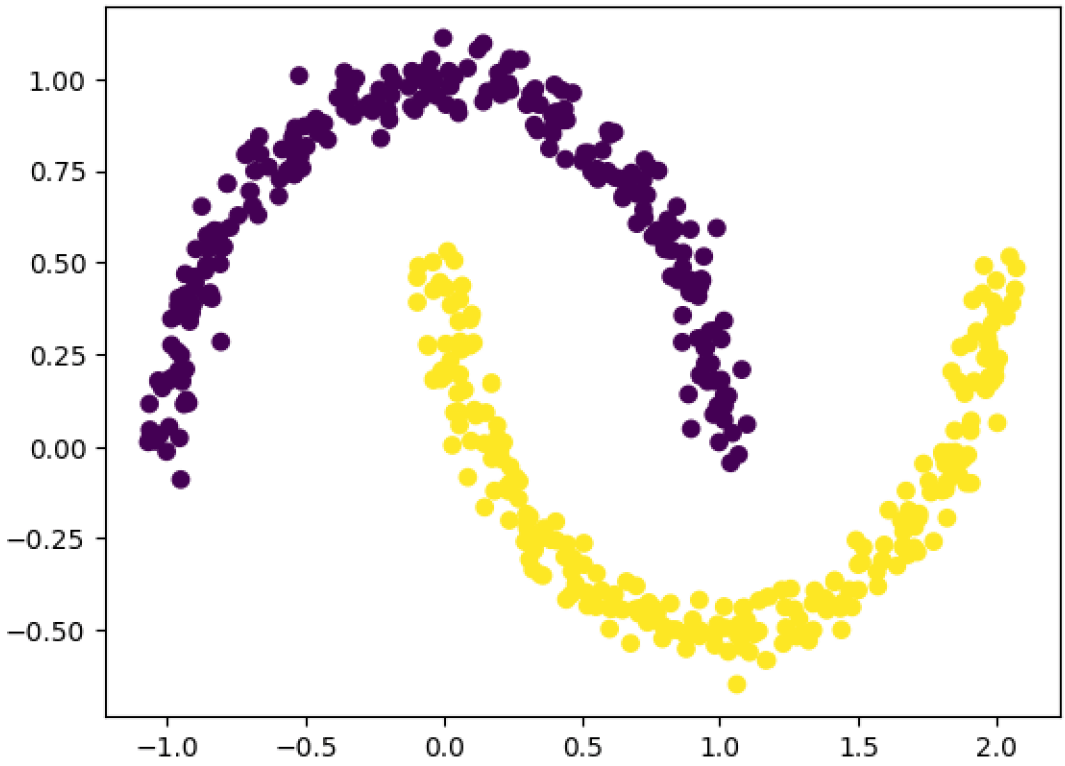
[5]:
```
# concentric circles data

circle_X, circle_true_labels = make_circles(n_samples=500, noise=.01,␣
 ↪random_state=seed)

scaled_circle_X = StandardScaler().fit_transform(circle_X)

plt.scatter(circle_X[:, 0], circle_X[:, 1], c=circle_true_labels)
```

[5]: <matplotlib.collections.PathCollection at 0x7f586328a290>



[6]:
```
# moons data

moons_X, moons_true_labels = make_moons(n_samples=500, noise=0.05,␣
↪random_state=seed)

scaled_moons_X = StandardScaler().fit_transform(moons_X)

plt.scatter(moons_X[:, 0], moons_X[:, 1], c=moons_true_labels)
```

[6]:
```
<matplotlib.collections.PathCollection at 0x7f5861922da0>
```



[ ]:

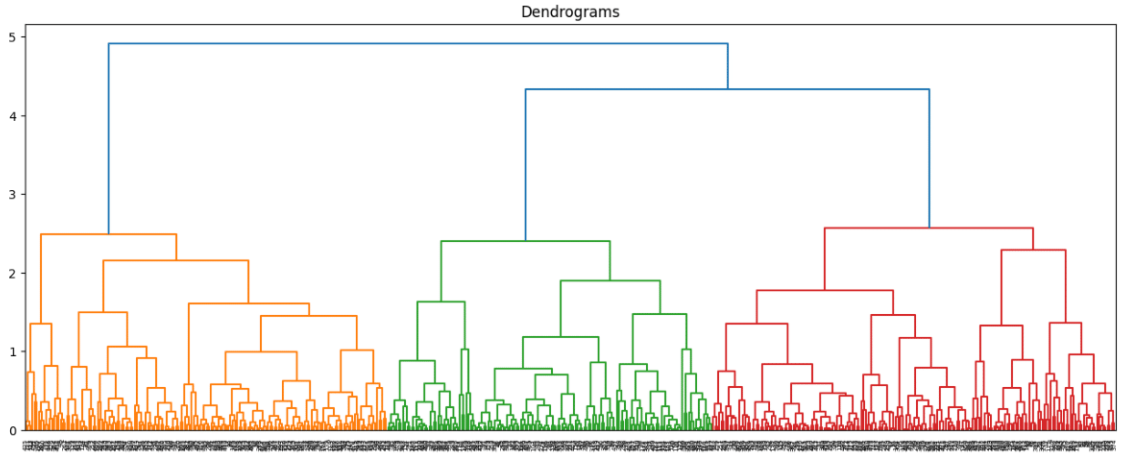
0.0.2 Hierarchical Clustering

Unlike **K-means clustering**, tree-like morphologies are used to bunch the dataset, and dendrograms are used to create the hierarchy of the clusters.

Agglomerarive Clustering is based on the distance between groups, similar collections are merged based on the loss of the algorithm after one iteration. Again the loss value is calculated in the next iteration, where similar clusters are combined again. The process continues until we reach the minimum value of the loss.

Dendogram Clustering Plot A **Hierarchical clustering** is typically visualized as a **dendrogram** as shown in the following cell. Each merge is represented by a *horizontal line*. The y-coordinate of the horizontal line is the similarity of the two clusters that were merged, where cities are viewed as singleton clusters. By moving up from the bottom layer to the top node, a dendrogram allows us to reconstruct the history of merges that resulted in the depicted clustering.

[7]:
```
plt.figure(figsize=(16, 6))
plt.title("Dendrograms")
Z = shc.linkage(scaled_blob_X, method="complete")
dend = shc.dendrogram(Z)
```



Agglomerative Clustering The **Agglomerative Clustering** class will require two inputs:

- **n_clusters**: The number of clusters to form as well as the number of centroids to generate.
  - Value will be: 3
- **linkage**: Which linkage criterion to use. The linkage criterion determines which distance to use between sets of observation. The algorithm will merge the pairs of cluster that minimize this criterion.
  - Value will be: 'complete'

Hierarchical Agglomerative clustering starts with treating each observation as an individual cluster, and then iteratively merges clusters until all the data points are merged into a single cluster.

Clusters are merged based on the distance between them and to calculate the distance between the clusters we have different types of linkages.

Linkage Criteria: It determines the distance between sets of observations as a function of the pairwise distance between observations.

- **Single Linkage**: the distance between two clusters is the minimum distance between members of the two clusters
- **Complete Linkage**: the distance between two clusters is the maximum distance between members of the two clusters
- **Average Linkage**: the distance between two clusters is the average of all distances between members of the two clusters
- **Centroid Linkage**: the distance between two clusters is is the distance between their centroids

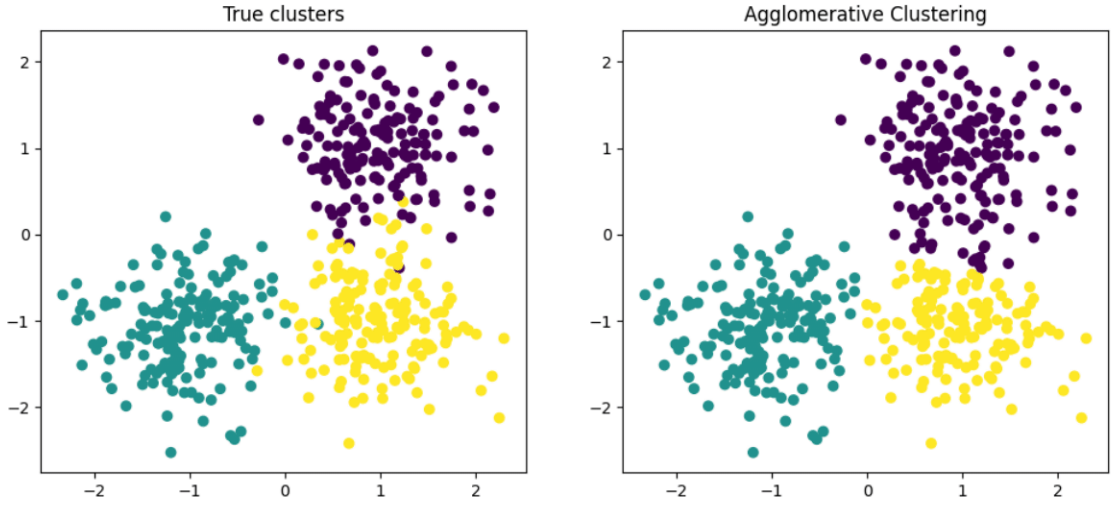
At least $O(n^2 \log n)$, where 'n' is the number of data points.

For Blob Data

[8]:
```
agglo_cluster = AgglomerativeClustering(n_clusters=3, metric='euclidean',␣
↪linkage='complete')
agglo_cluster.fit(scaled_blob_X)
```

[8]:
```
AgglomerativeClustering(linkage='complete', metric='euclidean', n_clusters=3)
```

[9]:
```
plot_clusters(blob_X, blob_true_labels, agglo_cluster.labels_,␣
↪title_cluster="Agglomerative Clustering")
```



[10]:
```
X, pred_labels, true_labels = scaled_blob_X, agglo_cluster.labels_,␣
↪blob_true_labels

print("Silhouette Coefficient: %0.3f" % metrics.silhouette_score(X,␣
↪pred_labels))
print("Adjusted Rand Index: %0.3f" % metrics.adjusted_rand_score(true_labels,␣
↪pred_labels))
print("Adjusted Mutual Information: %0.3f"
      % metrics.adjusted_mutual_info_score(true_labels, pred_labels,␣
↪average_method='arithmetic'))
```

```
Silhouette Coefficient: 0.555
Adjusted Rand Index: 0.881
Adjusted Mutual Information: 0.863
```

[11]:
```

```

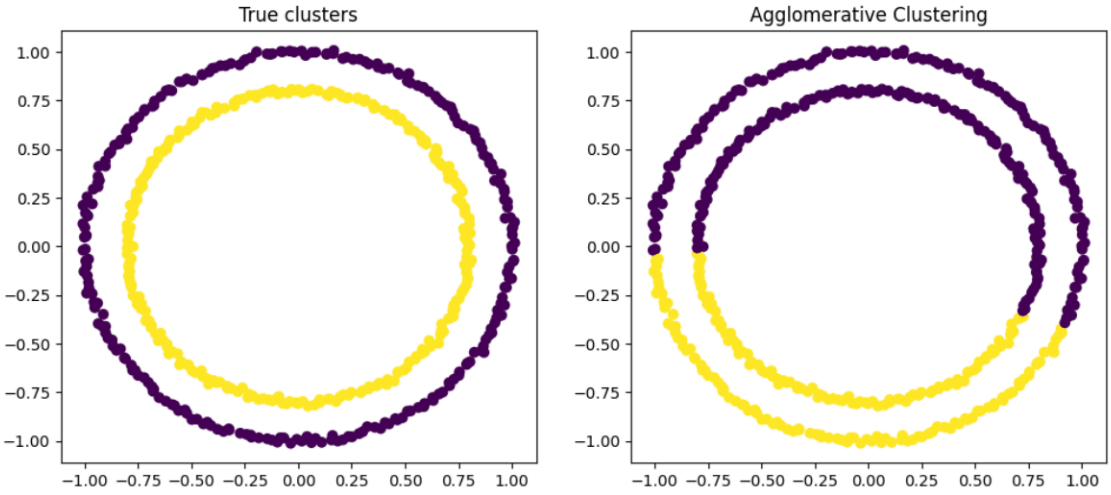
For Circular Data

[11]:
```
agglo_cluster = AgglomerativeClustering(n_clusters=2, metric='euclidean',␣
↪linkage='complete')
```

```
agglo_cluster.fit(scaled_circle_X)
```

[11]:
```
AgglomerativeClustering(linkage='complete', metric='euclidean')
```

[12]:
```
plot_clusters(circle_X, circle_true_labels, agglo_cluster.labels_,␣
 ↪title_cluster="Agglomerative Clustering")
```



[13]:
```
X, pred_labels, true_labels = scaled_circle_X, agglo_cluster.labels_,␣
 ↪circle_true_labels

print("Silhouette Coefficient: %0.3f" % metrics.silhouette_score(X,␣
 ↪pred_labels))
print("Adjusted Rand Index: %0.3f" % metrics.adjusted_rand_score(true_labels,␣
 ↪pred_labels))
print("Adjusted Mutual Information: %0.3f"
      % metrics.adjusted_mutual_info_score(true_labels, pred_labels,␣
 ↪average_method='arithmetic'))
```

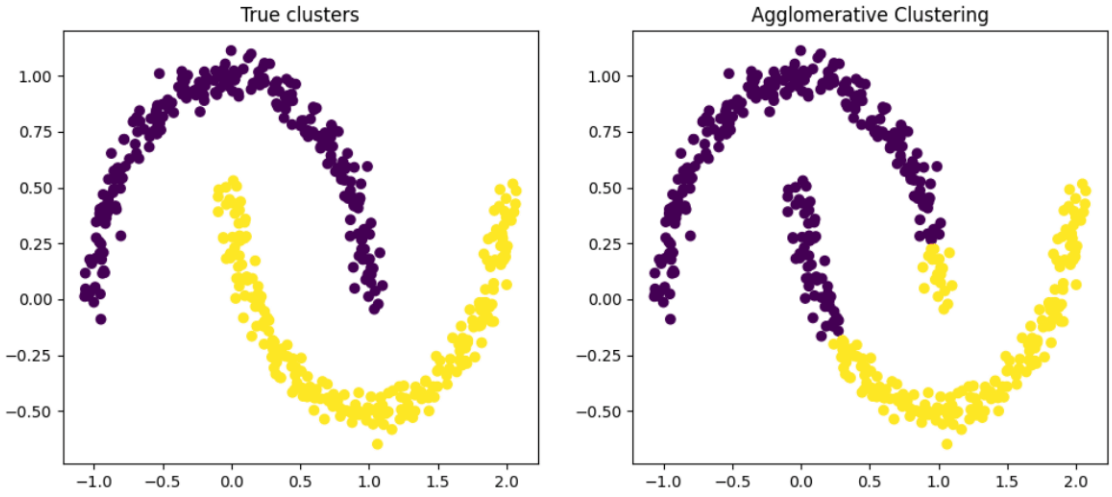
```
Silhouette Coefficient: 0.389
Adjusted Rand Index: -0.002
Adjusted Mutual Information: -0.001
```

For Moons Data

[14]:
```
agglo_cluster = AgglomerativeClustering(n_clusters=2, metric='euclidean',␣
 ↪linkage='ward')
agglo_cluster.fit(scaled_moons_X)
```

[14]:
```
AgglomerativeClustering(metric='euclidean')
```

[15]:
```
plot_clusters(moons_X, moons_true_labels, agglo_cluster.labels_,␣
↪title_cluster="Agglomerative Clustering")
```



[16]:
```
X, pred_labels, true_labels = scaled_moons_X, agglo_cluster.labels_,␣
↪moons_true_labels

print("Silhouette Coefficient: %0.3f" % metrics.silhouette_score(X,␣
↪pred_labels))
print("Adjusted Rand Index: %0.3f" % metrics.adjusted_rand_score(true_labels,␣
↪pred_labels))
print("Adjusted Mutual Information: %0.3f"
      % metrics.adjusted_mutual_info_score(true_labels, pred_labels,␣
↪average_method='arithmetic'))
```

```
Silhouette Coefficient: 0.484
Adjusted Rand Index: 0.467
Adjusted Mutual Information: 0.390
```

[ ]:
```
```

0.0.3 DBSCAN

DBSCAN stands for Density-Based Spatial Clustering of Applications with Noise.

- It groups 'densely grouped' data points into a single cluster. The most exciting feature of DBSCAN clustering is that it is **robust to outliers**.
- It also does not require the number of clusters to be told beforehand, unlike K-Means, where we have to specify the number of centroids.

Two hyperparameters: - epsilon: Epsilon is the radius of the circle to be created around each data point to check the density. - minPoints: minPoints is the minimum number of data points required

inside that circle for that data point to be classified as a **Core** point.

In higher dimensions the circle becomes **hypersphere**, epsilon becomes the **radius** of that hypersphere, and minPoints is the minimum number of data points required inside that hypersphere.

If the number of points is less than minPoints, then it is classified as **Border Point**, and if there are no other data points around any data point within epsilon radius, then it treated as **Noise**.

For locating data points in space, DBSCAN uses **Euclidean distance**. It also needs to scan through the entire dataset once, whereas in other algorithms we have to do it multiple times.

The value of minPoints should be at least one greater than the number of dimensions of the dataset, i.e.,

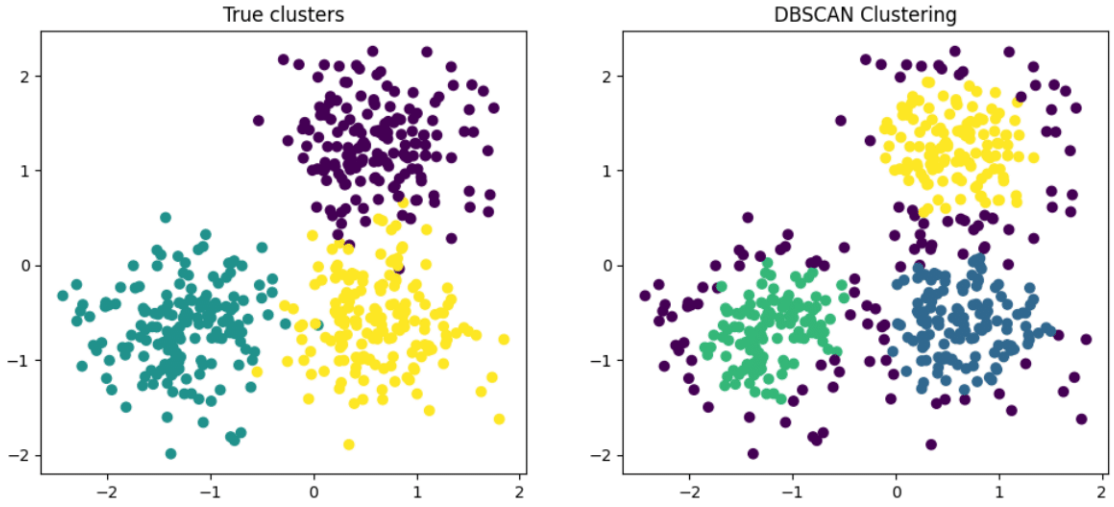
minPoints>=Dimensions+1. Generally, it is twice the dimensions. (Domain Knowledge, in practice.)

For Blob Data

[17]:
```
dbscan_cluster = DBSCAN(eps=0.3, min_samples=20)
dbscan_cluster.fit(scaled_blob_X)
```

[17]:
```
DBSCAN(eps=0.3, min_samples=20)
```

[18]:
```
plot_clusters(scaled_blob_X, blob_true_labels, dbscan_cluster.labels_,␣
↪title_cluster="DBSCAN Clustering")
```



[19]:
```
X, pred_labels, true_labels = scaled_blob_X, dbscan_cluster.labels_,␣
↪blob_true_labels
```

```
print("Silhouette Coefficient: %0.3f" % metrics.silhouette_score(X,␣
 ↪pred_labels))
print("Adjusted Rand Index: %0.3f" % metrics.adjusted_rand_score(true_labels,␣
 ↪pred_labels))
print("Adjusted Mutual Information: %0.3f"
      % metrics.adjusted_mutual_info_score(true_labels, pred_labels,␣
 ↪average_method='arithmetic'))
```

```
Silhouette Coefficient: 0.369
Adjusted Rand Index: 0.589
Adjusted Mutual Information: 0.637
```

Terms: - Directly Density-Reachable - Density-Reachable - Density-Connected

- Directly Density-Reachable

- Density-Reachable

- Density-Connected

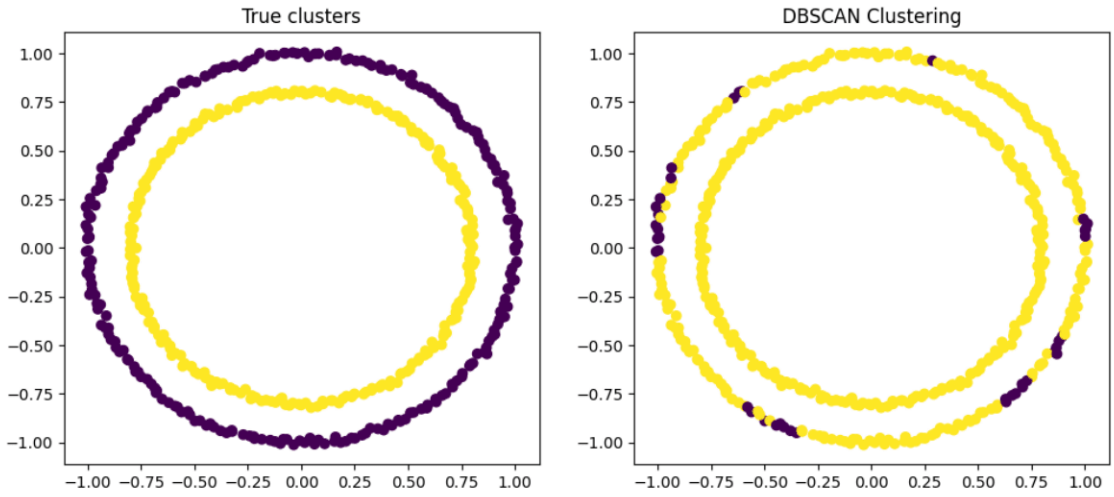
[ ]:

For Circular Data

[20]:
```
dbscan_cluster = DBSCAN(eps=0.3, min_samples=20)
dbscan_cluster.fit(scaled_circle_X)
```

[20]:
```
DBSCAN(eps=0.3, min_samples=20)
```

[21]:
```
plot_clusters(circle_X, circle_true_labels, dbscan_cluster.labels_,␣
 ↪title_cluster="DBSCAN Clustering")
```



[22]:
```
X, pred_labels, true_labels = scaled_circle_X, dbscan_cluster.labels_,␣
 ↪circle_true_labels

print("Silhouette Coefficient: %0.3f" % metrics.silhouette_score(X,␣
 ↪pred_labels))
print("Adjusted Rand Index: %0.3f" % metrics.adjusted_rand_score(true_labels,␣
 ↪pred_labels))
print("Adjusted Mutual Information: %0.3f"
      % metrics.adjusted_mutual_info_score(true_labels, pred_labels,␣
 ↪average_method='arithmetic'))
```

```
Silhouette Coefficient: 0.052
Adjusted Rand Index: 0.026
Adjusted Mutual Information: 0.122
```

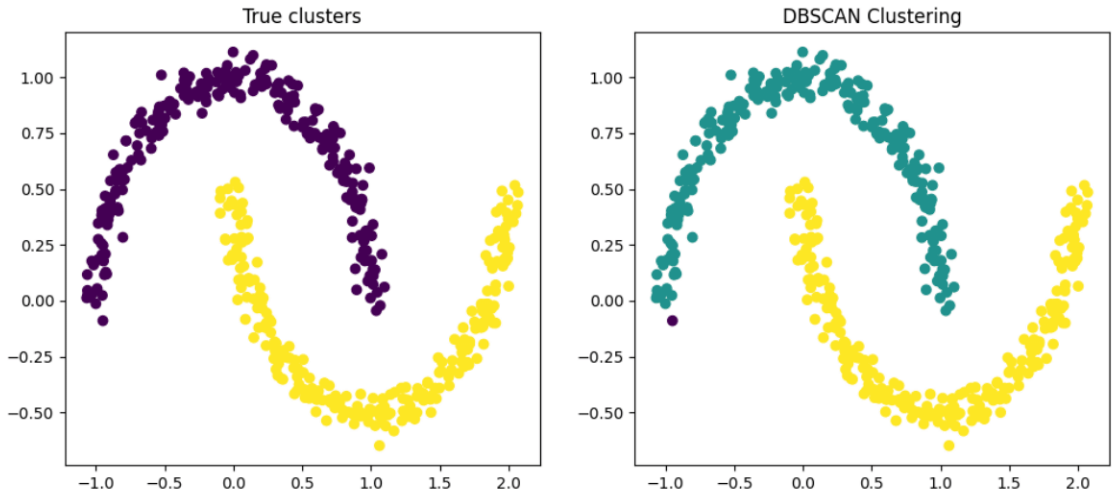
[22]:
```

```

For Moons Data

[23]:
```
dbscan_cluster = DBSCAN(eps=0.3, min_samples=20)
dbscan_cluster.fit(scaled_moons_X)
```

[23]:
```
DBSCAN(eps=0.3, min_samples=20)
```

[24]:
```
plot_clusters(moons_X, moons_true_labels, dbscan_cluster.labels_,␣
 ↪title_cluster="DBSCAN Clustering")
```



[25]:
```
X, pred_labels, true_labels = scaled_moons_X, dbscan_cluster.labels_,␣
 ↪moons_true_labels

print("Silhouette Coefficient: %0.3f" % metrics.silhouette_score(X,␣
 ↪pred_labels))
print("Adjusted Rand Index: %0.3f" % metrics.adjusted_rand_score(true_labels,␣
 ↪pred_labels))
print("Adjusted Mutual Information: %0.3f"
      % metrics.adjusted_mutual_info_score(true_labels, pred_labels,␣
 ↪average_method='arithmetic'))
```

```
Silhouette Coefficient: 0.250
Adjusted Rand Index: 0.996
Adjusted Mutual Information: 0.991
```

[ ]:
```

```

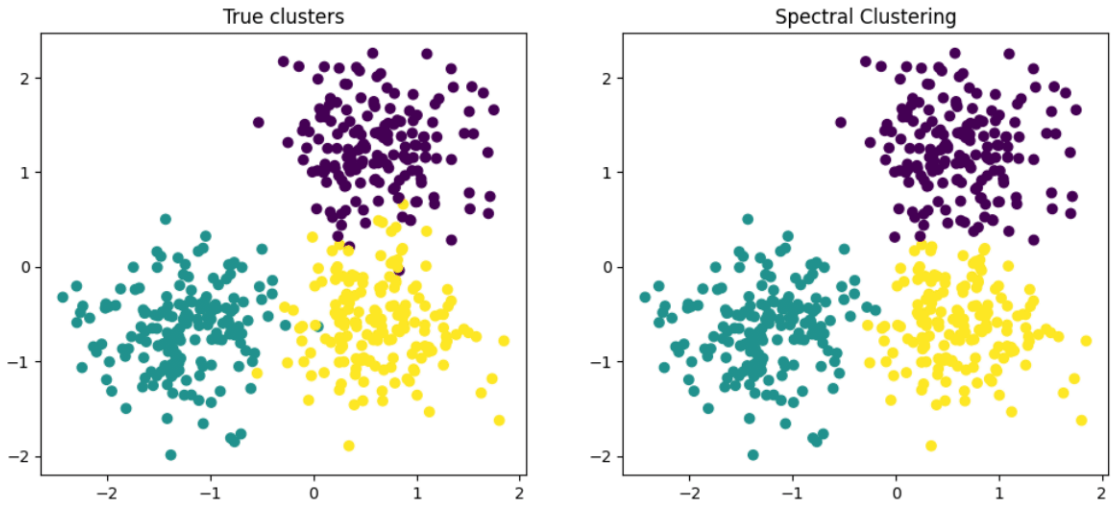
0.0.4 Spectral Clustering

For Blob Data

[26]:
```
spectral_cluster = SpectralClustering(n_clusters=3,␣
 ↪affinity='nearest_neighbors')
spectral_cluster.fit(scaled_blob_X)
```

[26]:
```
SpectralClustering(affinity='nearest_neighbors', n_clusters=3)
```

[27]:
```
plot_clusters(scaled_blob_X, blob_true_labels, spectral_cluster.labels_,␣
 ↪title_cluster="Spectral Clustering")
```



[28]:
```
X, pred_labels, true_labels = scaled_blob_X, spectral_cluster.labels_,␣
 ↪blob_true_labels

print("Silhouette Coefficient: %0.3f" % metrics.silhouette_score(X,␣
 ↪pred_labels))
print("Adjusted Rand Index: %0.3f" % metrics.adjusted_rand_score(true_labels,␣
 ↪pred_labels))
print("Adjusted Mutual Information: %0.3f"
      % metrics.adjusted_mutual_info_score(true_labels, pred_labels,␣
 ↪average_method='arithmetic'))
```

```
Silhouette Coefficient: 0.566
Adjusted Rand Index: 0.918
Adjusted Mutual Information: 0.880
```

[ ]:
```

```

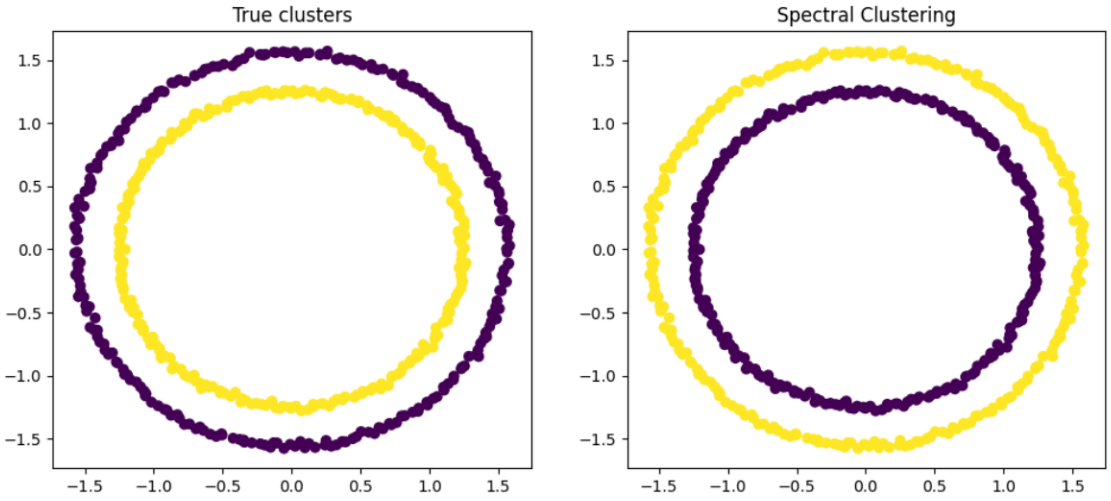
For Circular Data

[29]:
```
spectral_cluster = SpectralClustering(n_clusters=2,␣
 ↪affinity='nearest_neighbors')
spectral_cluster.fit(scaled_circle_X)
```

```
/usr/local/lib/python3.10/dist-
packages/sklearn/manifold/_spectral_embedding.py:274: UserWarning: Graph is not
fully connected, spectral embedding may not work as expected.
  warnings.warn(
```

[29]:
```
SpectralClustering(affinity='nearest_neighbors', n_clusters=2)
```

[30]:
```
plot_clusters(scaled_circle_X, circle_true_labels, spectral_cluster.labels_,␣
↪title_cluster="Spectral Clustering")
```



[31]:
```
X, pred_labels, true_labels = scaled_circle_X, spectral_cluster.labels_,␣
↪circle_true_labels

# Explain students why silhoutte score is low.

print("Silhouette Coefficient: %0.3f" % metrics.silhouette_score(X,␣
↪pred_labels))
print("Adjusted Rand Index: %0.3f" % metrics.adjusted_rand_score(true_labels,␣
↪pred_labels))
print("Adjusted Mutual Information: %0.3f"
      % metrics.adjusted_mutual_info_score(true_labels, pred_labels,␣
↪average_method='arithmetic'))
```

```
Silhouette Coefficient: 0.019
Adjusted Rand Index: 1.000
Adjusted Mutual Information: 1.000
```

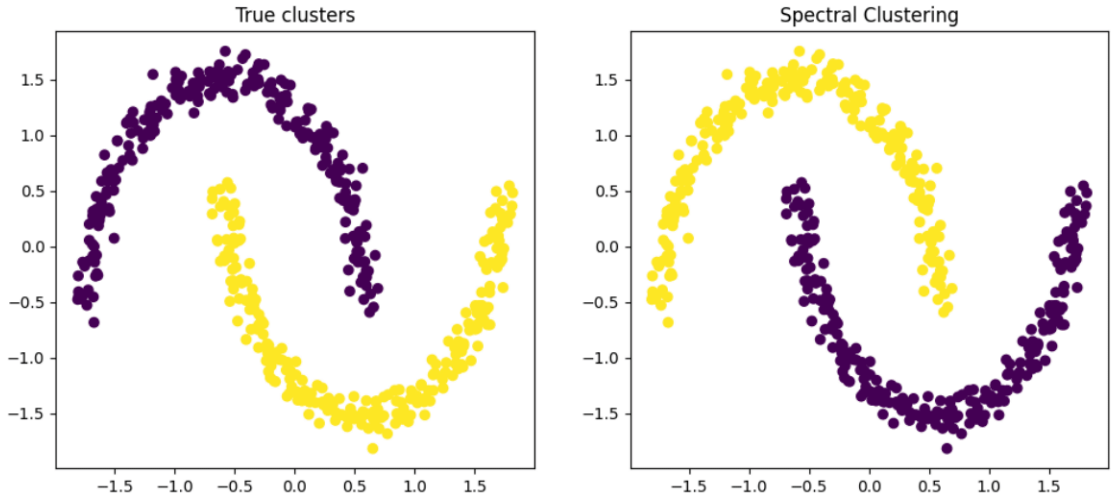
For Moons Data

[34]:
```
spectral_cluster = SpectralClustering(n_clusters=2,␣
↪affinity='nearest_neighbors')
spectral_cluster.fit(scaled_moons_X)
```

```
/usr/local/lib/python3.10/dist-
packages/sklearn/manifold/_spectral_embedding.py:274: UserWarning: Graph is not
fully connected, spectral embedding may not work as expected.
  warnings.warn(
```

[34]:
```
SpectralClustering(affinity='nearest_neighbors', n_clusters=2)
```

[35]:
```
plot_clusters(scaled_moons_X, moons_true_labels, spectral_cluster.labels_,␣
↪title_cluster="Spectral Clustering")
```



[36]:
```
X, pred_labels, true_labels = scaled_moons_X, spectral_cluster.labels_,␣
↪moons_true_labels

# Explain students why silhoutte score is low.

print("Silhouette Coefficient: %0.3f" % metrics.silhouette_score(X,␣
↪pred_labels))
print("Adjusted Rand Index: %0.3f" % metrics.adjusted_rand_score(true_labels,␣
↪pred_labels))
print("Adjusted Mutual Information: %0.3f"
      % metrics.adjusted_mutual_info_score(true_labels, pred_labels,␣
↪average_method='arithmetic'))
```

```
Silhouette Coefficient: 0.387
Adjusted Rand Index: 1.000
Adjusted Mutual Information: 1.000
```

[ ]:

1 Question

[ ]:
```
# Explore and Compare Linkage Techniques (Optional).
# Using the given test data report best rand index and mutual information score␣
 ↪\
#   for all the above discussed algorithms, along with the silhouette score.
# Explain the ambiguity in Silhoutte Scores.
```