clustering-analysis

May 4, 2024

COMPARING THE PERFOMANCE OF VARIOUS CLUSTERING ALGORITHMS ON SYNTHETIC DATASETS WITH DIFFERENT STRUCTURES.

The purpose of this code is to compare the performance of various clustering algorithms on 2 different synthetic datasets with different structures. It allows you to visualize how each algorithm clusters the data and how well it performs compared to others.

Three different synthetic datasets are generated: Circular structure (X1) with num_samples_circle samples. Blob structure (X2) with num_samples_square samples. Moons structure (X3) with num_samples_moons samples.

Combining the datasets: The three datasets are combined into a single dataset X. y label is assigned to the combined dataset.

A plot of the combined dataset X with different structures labeled with different colors is dispalyed.

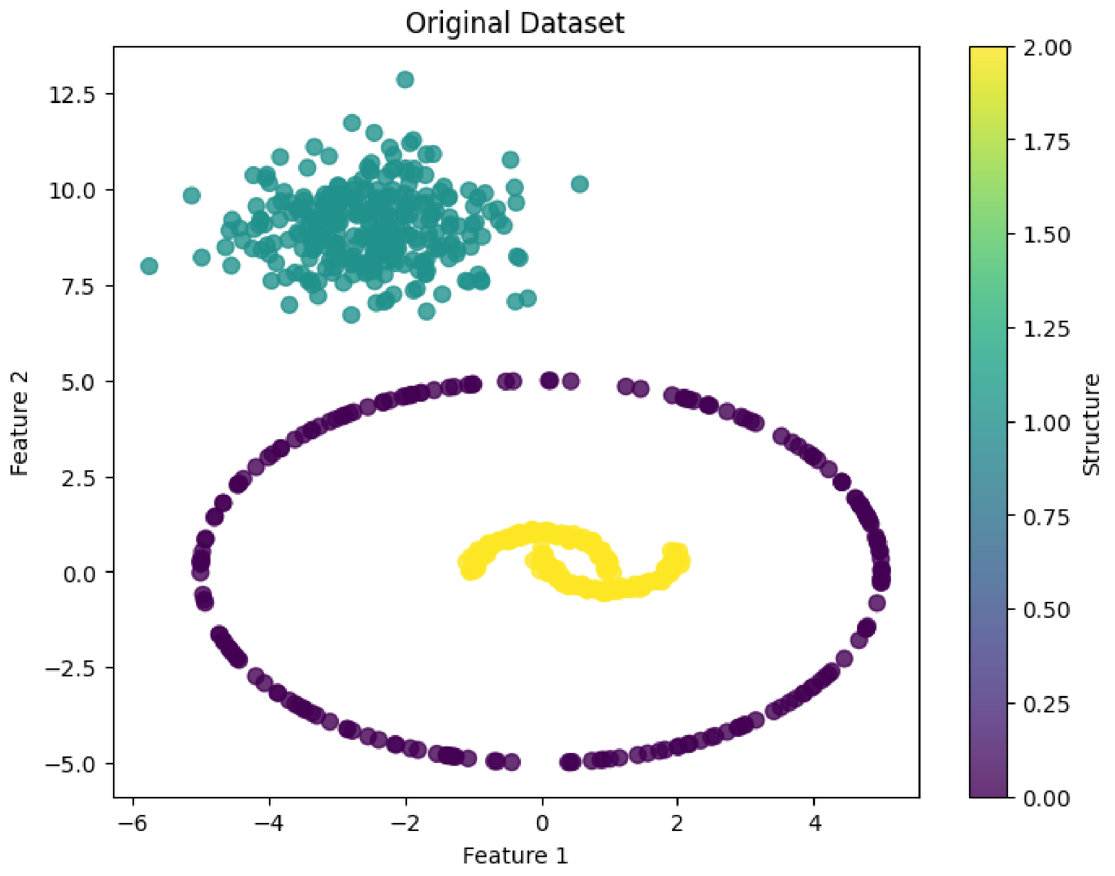
A list of clustering algorithms along with their corresponding parameter configurations is created. Algorithms used are: KMeans OPTICS HDBSCAN Spectral Clustering Gaussian Mixture Models Agglomerative Clustering MeanShift Affinity Propagation Birch

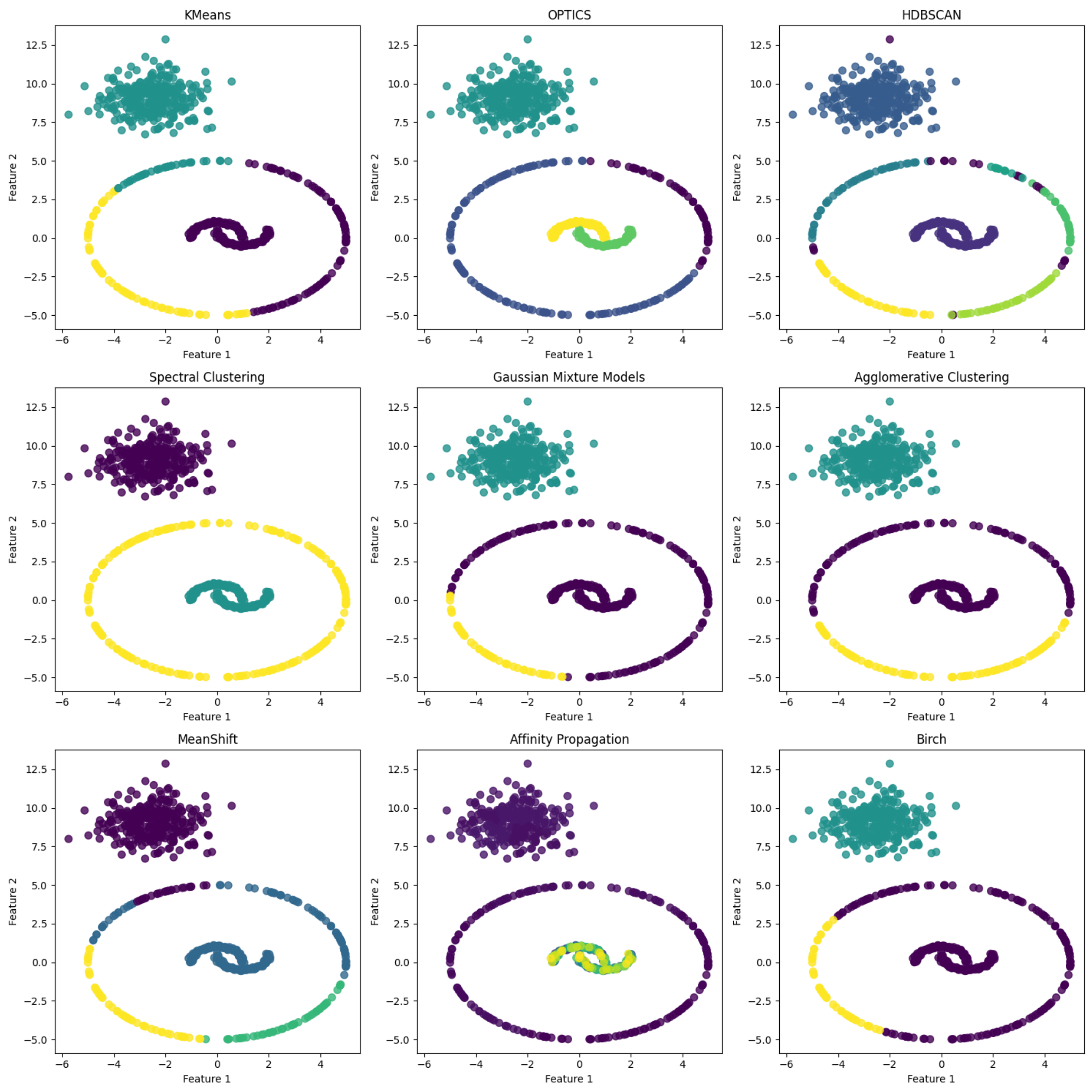
For each algorithm, the algorithm is fitted to the data and the resulting clusters are plotted. Subplots are used to display the clustering results for all algorithms.

The follwing is synthetic dataset 1:

[3]:
```
import warnings
from sklearn.exceptions import ConvergenceWarning

warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=ConvergenceWarning)

import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_circles, make_blobs, make_moons
from sklearn.cluster import KMeans, OPTICS, SpectralClustering,␣
 ↪AgglomerativeClustering, MeanShift, AffinityPropagation, Birch
from sklearn.mixture import GaussianMixture
import hdbscan

# Generating different structures
num_samples_circle = 200
```

```
radius_circle = 5
theta_circle = np.random.uniform(0, 2*np.pi, num_samples_circle)
X1 = np.array([radius_circle * np.cos(theta_circle), radius_circle * np.
 ↪sin(theta_circle)]).T
y1 = np.zeros(num_samples_circle)  # Labels for structure 1

num_samples_square = 300
X2, _ = make_blobs(n_samples=num_samples_square, centers=1, cluster_std=1,␣
 ↪random_state=42)
y2 = np.ones(num_samples_square)  # Labels for structure 2

num_samples_moons = 200
X3, _ = make_moons(n_samples=num_samples_moons, noise=0.05, random_state=42)
y3 = np.full(num_samples_moons, 2)  # Labels for structure 3

# Combine all structures into a single dataset
X = np.vstack((X1, X2, X3))
y = np.concatenate((y1, y2, y3))

# Plotting the original dataset
plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis', marker='o', s=50, alpha=0.8)
plt.title('Original Dataset')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.colorbar(label='Structure')
plt.show()

# Apply all the clustering algorithms
clustering_algorithms = [
    ('KMeans', KMeans(n_clusters=3, random_state=42)),
    ('OPTICS', OPTICS(min_samples=10, xi=0.05, min_cluster_size=0.1)),
    ('HDBSCAN', hdbscan.HDBSCAN(min_cluster_size=10)),
    ('Spectral Clustering', SpectralClustering(n_clusters=3,␣
 ↪assign_labels='discretize', random_state=42)),
    ('Gaussian Mixture Models', GaussianMixture(n_components=3,␣
 ↪random_state=42)),
    ('Agglomerative Clustering', AgglomerativeClustering(n_clusters=3)),
    ('MeanShift', MeanShift()),
    ('Affinity Propagation', AffinityPropagation()),
    ('Birch', Birch(n_clusters=3))
]

# Plotting the clustering results
plt.figure(figsize=(15, 15))
for i, (name, algorithm) in enumerate(clustering_algorithms, start=1):
    plt.subplot(3, 3, i)
    clustering_result = algorithm.fit_predict(X)
    plt.scatter(X[:, 0], X[:, 1], c=clustering_result, cmap='viridis',␣
↪marker='o', s=50, alpha=0.8)
    plt.title(name)
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')

plt.tight_layout()
plt.show()
```





The following is synthetic dataset 2:

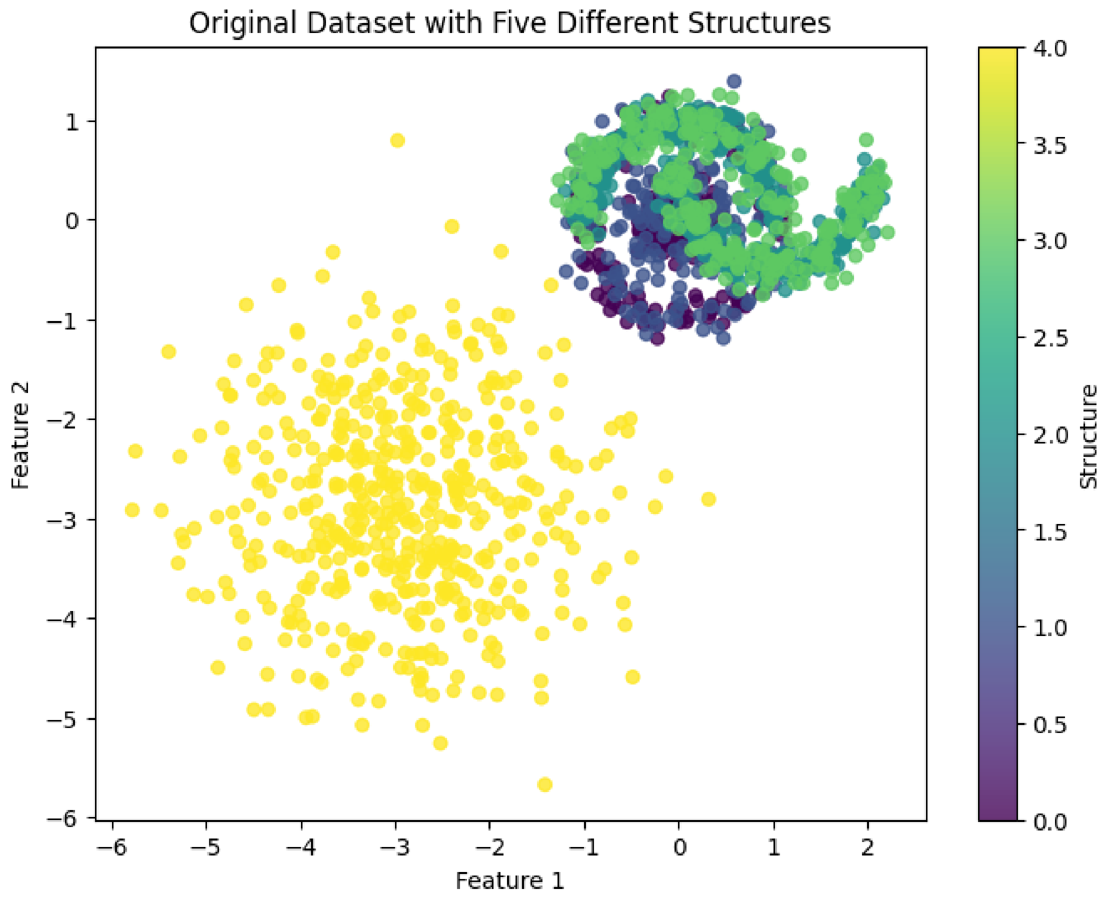
With the following slight deviations:

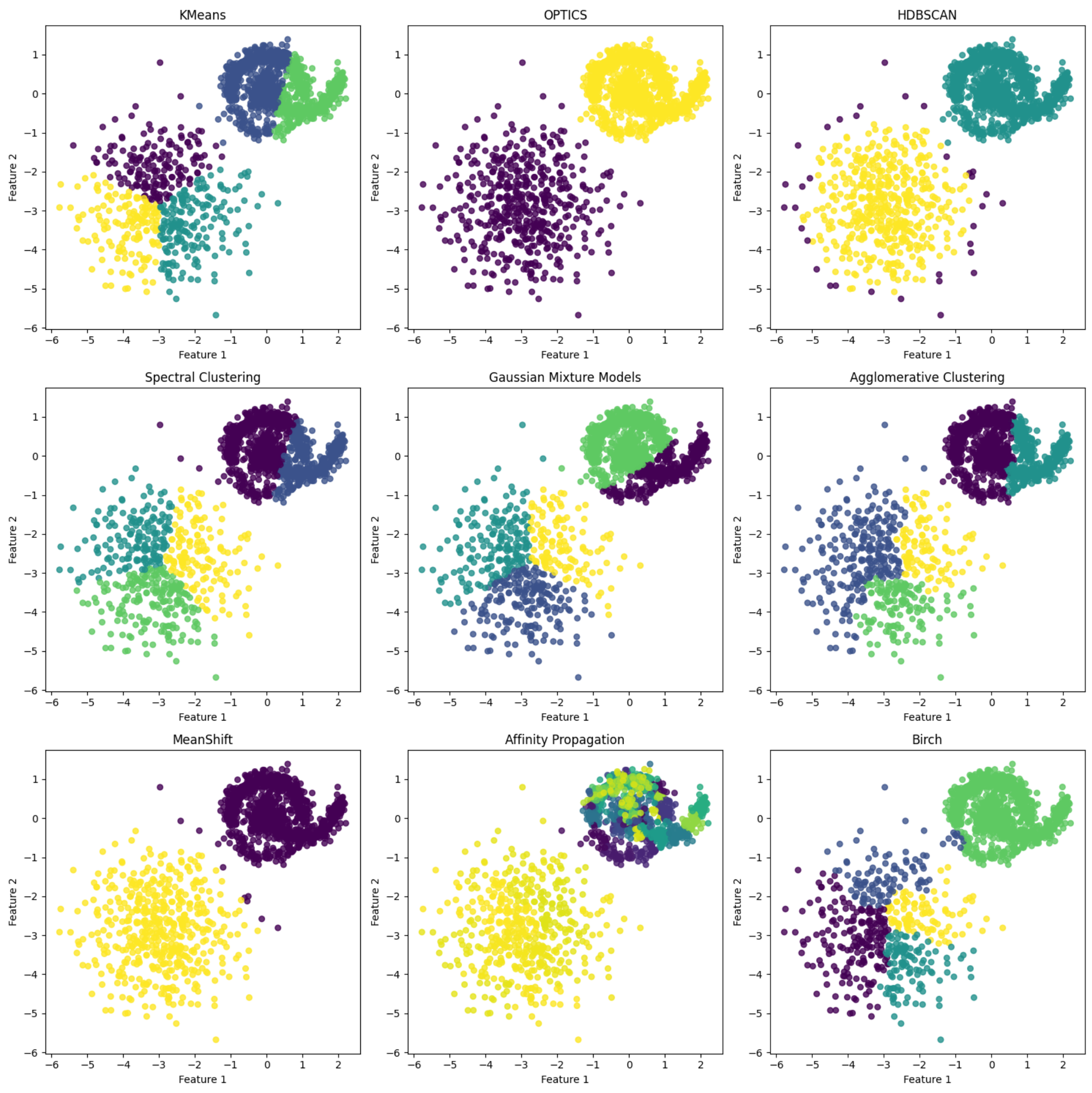
Generating synthetic datasets: Two circles with different densities (X_circle1 and X_circle2). Two moons with different densities (X_moons1 and X_moons2). One blob structure (X_blob). Labels are assigned to each dataset.

The rest of data preparation and algorithm implementation is similar.

[5]:
```python
# Generating the structures
num_samples_circle1 = 300
num_samples_circle2 = 200
num_samples_moons1 = 400
num_samples_moons2 = 300
```

```
num_samples_blob = 500

# First circle with higher density
X_circle1, _ = make_circles(n_samples=num_samples_circle1, factor=0.3, noise=0.
 ↪1)
y_circle1 = np.zeros(num_samples_circle1)

# Second circle with lower density
X_circle2, _ = make_circles(n_samples=num_samples_circle2, factor=0.4, noise=0.
 ↪15)
y_circle2 = np.ones(num_samples_circle2)

# First moon with higher density
X_moons1, _ = make_moons(n_samples=num_samples_moons1, noise=0.1)
y_moons1 = np.full(num_samples_moons1, 2)

# Second moon with lower density
X_moons2, _ = make_moons(n_samples=num_samples_moons2, noise=0.15)
y_moons2 = np.full(num_samples_moons2, 3)

# Blob structure
X_blob, _ = make_blobs(n_samples=num_samples_blob, centers=1, cluster_std=1.0)
y_blob = np.full(num_samples_blob, 4)

# Combine all structures into a single dataset
X = np.vstack((X_circle1, X_circle2, X_moons1, X_moons2, X_blob))
y = np.concatenate((y_circle1, y_circle2, y_moons1, y_moons2, y_blob))

# Plotting the original dataset
plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis', marker='o', s=30, alpha=0.8)
plt.title('Original Dataset with Five Different Structures')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.colorbar(label='Structure')

# Clustering algorithms
clustering_algorithms = [
    ('KMeans', KMeans(n_clusters=5, random_state=42)),
    ('OPTICS', OPTICS(min_samples=10, xi=0.05, min_cluster_size=0.1)),
    ('HDBSCAN', hdbscan.HDBSCAN(min_cluster_size=10)),
    ('Spectral Clustering', SpectralClustering(n_clusters=5,␣
 ↪assign_labels='discretize', random_state=42)),
    ('Gaussian Mixture Models', GaussianMixture(n_components=5,␣
 ↪random_state=42)),
    ('Agglomerative Clustering', AgglomerativeClustering(n_clusters=5)),
    ('MeanShift', MeanShift()),
    ('Affinity Propagation', AffinityPropagation()),
    ('Birch', Birch(n_clusters=5))
]

# Perform clustering analysis
plt.figure(figsize=(15, 15))
for i, (name, algorithm) in enumerate(clustering_algorithms, start=1):
    plt.subplot(3, 3, i)
    clustering_result = algorithm.fit_predict(X)
    plt.scatter(X[:, 0], X[:, 1], c=clustering_result, cmap='viridis',␣
 ↪marker='o', s=30, alpha=0.8)
    plt.title(name)
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')

plt.tight_layout()
plt.show()
```





Algorthm Descriptions and expected Behaviours for different data structures:

1. KMeans: KMeans is a centroid-based algorithm. It partitions the data into K clusters by iteratively assigning each data point to the nearest cluster centroid and then updating the centroids to the mean of the points in the cluster.

Expected Behavior: KMeans works well when clusters are spherical and of roughly similar size. It may struggle with non-spherical clusters or clusters with varying densities.

2. OPTICS (Ordering Points To Identify the Clustering Structure): OPTICS is a density-based algorithm that identifies clusters of varying density in large datasets.

Expected Behavior: OPTICS can find clusters of arbitrary shape and size and is robust to noise. It can identify clusters with irregular shapes and varying densities.

3. HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise):

HDBSCAN is a density-based algorithm that extends DBSCAN by incorporating a hierarchical approach to cluster identification.

Expected Behavior: HDBSCAN can find clusters of varying density and shape. It's robust to noise and can automatically determine the number of clusters.

4. Spectral Clustering: Spectral Clustering is a graph-based algorithm. It uses the eigenvalues of a similarity matrix to reduce the dimensionality of the data before clustering in fewer dimensions.

Expected Behavior: Spectral Clustering is effective for finding clusters in data with non-linear boundaries. It can find clusters of arbitrary shape and is not sensitive to outliers.

5. Gaussian Mixture Models (GMM): GMM is a probabilistic model that assumes that the data is generated from a mixture of several Gaussian distributions with unknown parameters.

Expected Behavior: GMM can find clusters of arbitrary shape. It is sensitive to the density of data points and can assign probabilities of data points to each cluster.

6. Agglomerative Clustering: Agglomerative Clustering is a hierarchical clustering algorithm that builds clusters by merging them together recursively.

Expected Behavior: Agglomerative Clustering can find clusters of arbitrary shape. It produces a dendrogram, which allows you to visualize the clustering process at different levels of granularity.

7. MeanShift: MeanShift is a centroid-based algorithm that doesn't require specifying the number of clusters. It automatically finds the number of clusters and their centroids.

Expected Behavior: MeanShift can find clusters of arbitrary shape and size. It's particularly useful when the number of clusters is not known a priori.

8. Affinity Propagation: Affinity Propagation is a graph-based algorithm that identifies 'exemplars' among data points and then assigns each data point to one of these exemplars.

Expected Behavior: Affinity Propagation can find clusters of arbitrary shape and size. It's particularly useful when the number of clusters is not known a priori.

9. Birch (Balanced Iterative Reducing and Clustering using Hierarchies): Birch is a hierarchical clustering algorithm. It incrementally and dynamically clusters incoming data points into a tree structure.

Expected Behavior: Birch is efficient for large datasets and can find clusters of arbitrary shape. It's particularly useful when memory resources are limited.