Mean Shift Clustering: Overview

Purpose: Mean Shift Clustering is a non-parametric clustering technique that does not assume any prior knowledge of the number of clusters. It works by iteratively shifting data points towards the mode (the highest density of data points) of a density function, thus forming clusters around these modes. The purpose of Mean Shift Clustering is to discover the number and location of clusters in a dataset.

When It Is Used: Mean Shift is used in various scenarios, especially when:

- The number of clusters is unknown.
- Clusters have arbitrary shapes and sizes.
- A density-based approach is more appropriate than a distance-based approach (e.g., k-means).

Suitable Data Types: Mean Shift Clustering is suitable for:

- Continuous numerical data.
- Data with an underlying density structure where modes (peaks) represent clusters.
- Image segmentation tasks.
- Feature space analysis where density peaks correspond to cluster centers.

Examples of Real-life Applications:

1. **Image Segmentation:** Segmenting an image into regions with similar colors or textures.
2. **Object Tracking in Videos:** Tracking moving objects by identifying and following modes in the feature space.
3. **Anomaly Detection:** Identifying regions with low density as potential anomalies.
4. **Spatial Data Analysis:** Clustering geographic data points to identify regions with high density.

Implementation of Mean Shift Clustering on a Real-life Dataset

Let's implement Mean Shift Clustering using Python on a sample dataset. We'll use the `sklearn` library for the implementation and a real-life dataset for demonstration.

Dataset:

We'll use the "Mall Customer Segmentation Data" dataset, which includes customer information and spending scores. The goal is to identify clusters of customers with similar spending behavior.

Working Mechanism of Mean Shift Clustering

Mean Shift Clustering is a non-parametric and density-based clustering algorithm that seeks to find the modes (peaks) of a density function given a set of data points. Here's a step-by-step explanation of how it works:

1. **Initialization:**
   o Mean Shift starts with a set of data points.
   o Each data point is treated as a potential cluster center (also known as a "seed").
2. **Bandwidth Selection:**

   In the context of Mean Shift Clustering, the **bandwidth** is a crucial hyperparameter that determines the radius of the search window (also referred to as the kernel) around each data point. It defines the region within which the algorithm will search for points to calculate the mean shift vector.

   o A crucial parameter in Mean Shift is the bandwidth (also known as the window size or radius), denoted as hhh. This parameter defines the radius of the region (kernel) around each data point within which the algorithm searches for higher density.
   o The bandwidth can be selected using techniques like cross-validation, domain knowledge, or heuristic methods such as the "rule of thumb."
3. **Shifting Step:**
   o For each data point xix_ixi, the algorithm defines a window of radius hhh around xix_ixi.
   o Within this window, the algorithm computes the mean of the data points. This mean is the "mean shift vector."
   o The data point xix_ixi is then shifted towards this mean. Mathematically, the new position of xix_ixi is:
   $xinew=\sum xj\in windowK(xj-xi)\cdot xj\sum xj\in windowK(xj-xi)x\_i^{new} = \frac{\sum_{x_j \in \text{window}} K(x_j - x_i) \cdot x_j}{\sum_{x_j \in \text{window}} K(x_j - x_i)}xinew=\sum xj\in windowK(xj-xi)\sum xj\in windowK(xj-xi)\cdot xj$ where KKK is the kernel function (commonly a Gaussian kernel).
4. **Convergence:**
   o The shifting step is repeated iteratively for each data point until the shift is smaller than a predefined threshold (i.e., the points converge to a stable location).
   o Points that converge to the same mode are assigned to the same cluster.
5. **Cluster Formation:**
   o After convergence, each point is associated with a mode. Points with the same mode are grouped together to form clusters.
6. **Cluster Pruning (Optional):**
   o In practice, modes that are very close to each other can be merged to avoid having too many small clusters.
   o This step involves defining a distance threshold below which two modes are considered the same.

Visualization of the Mean Shift Process

To help visualize the Mean Shift process, consider a set of data points distributed in 2D space:

1. **Initial Data Points:**

- o Imagine a scatter plot of points representing customer data with "Annual Income" on the x-axis and "Spending Score" on the y-axis.
2. **Bandwidth Selection:**
   - o A circular window (kernel) is centered on each data point.
3. **Shifting Step:**
   - o For each point, calculate the centroid of all points within the window.
   - o Shift the point towards the centroid.
   - o Repeat this until convergence.
4. **Final Clusters:**
   - o Once all points have converged to their respective modes, identify clusters by grouping points that converge to the same mode.

Implementation

```
import pandas as pd

import numpy as np

from sklearn.cluster import MeanShift, estimate_bandwidth

from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt


# Load the dataset

url = "https://raw.githubusercontent.com/JWarmenhoven/Mall-
Customers/master/Mall_Customers.csv"

data = pd.read_csv(url)


# Select the features for clustering

features = data[['Annual Income (k$)', 'Spending Score (1-100)']]


# Standardize the features

scaler = StandardScaler()

features_scaled = scaler.fit_transform(features)


# Visualize the data before clustering

plt.figure(figsize=(10, 6))

plt.scatter(features['Annual Income (k$)'], features['Spending Score (1-100)'], c='gray', marker='o')
```

```python
plt.title('Data Before Clustering')

plt.xlabel('Annual Income (k$)')

plt.ylabel('Spending Score (1-100)')

plt.show()


# Estimate the bandwidth

bandwidth = estimate_bandwidth(features_scaled, quantile=0.2)


# Apply Mean Shift Clustering

mean_shift = MeanShift(bandwidth=bandwidth)

mean_shift.fit(features_scaled)


# Get the cluster labels

labels = mean_shift.labels_

cluster_centers = mean_shift.cluster_centers_


# Add the cluster labels to the original dataset

data['Cluster'] = labels


# Visualize the data after clustering

plt.figure(figsize=(10, 6))

unique_labels = np.unique(labels)

colors = plt.cm.viridis(np.linspace(0, 1, len(unique_labels)))


for k, col in zip(unique_labels, colors):

    class_member_mask = (labels == k)

    plt.plot(features['Annual Income (k$)'][class_member_mask], features['Spending Score (1-
100)'][class_member_mask], 'o', markerfacecolor=col, markeredgecolor='k', markersize=8,
label=f'Cluster {k}')
```

```
# Plot the cluster centers
```

for center in cluster_centers:

```
    plt.plot(center[0] * scaler.scale_[0] + scaler.mean_[0], center[1] * scaler.scale_[1] + scaler.mean_[1], 'x', markerfacecolor='red', markeredgecolor='k', markersize=12)
```

plt.title('Mean Shift Clustering of Customers')

plt.xlabel('Annual Income (k$)')

plt.ylabel('Spending Score (1-100)')

plt.legend()

plt.show()

Code Explanation

- `pandas as pd`: Imports the pandas library and assigns it the alias `pd` for data manipulation and analysis.
- `numpy as np`: Imports the numpy library and assigns it the alias `np` for numerical operations.
- `from sklearn.cluster import MeanShift, estimate_bandwidth`: Imports the Mean Shift clustering algorithm and a function to estimate the bandwidth from the scikit-learn library.
- `from sklearn.preprocessing import StandardScaler`: Imports the StandardScaler class to standardize features by removing the mean and scaling to unit variance.
- `import matplotlib.pyplot as plt`: Imports the matplotlib library for plotting and assigns it the alias `plt`.

- `url`: Specifies the URL of the CSV file containing the dataset.
- `data = pd.read_csv(url)`: Reads the CSV file from the specified URL into a pandas DataFrame named `data`.

`features`: Selects the 'Annual Income (k$)' and 'Spending Score (1-100)' columns from the DataFrame `data` for clustering.

- `scaler = StandardScaler()`: Creates an instance of the StandardScaler class.
- `features_scaled = scaler.fit_transform(features)`: Fits the StandardScaler to the selected features and transforms them to have zero mean and unit variance.

- `plt.figure(figsize=(10, 6))`: Creates a new figure with a specified size.
- `plt.scatter(features['Annual Income (k$)'], features['Spending Score (1-100)'], c='gray', marker='o')`: Plots a scatter plot of the data points with gray color and 'o' marker.
- `plt.title('Data Before Clustering')`: Sets the title of the plot.
- `plt.xlabel('Annual Income (k$)')`: Sets the label for the x-axis.

- `plt.ylabel('Spending Score (1-100)')`: Sets the label for the y-axis.
- `plt.show()`: Displays the plot.

`bandwidth = estimate_bandwidth(features_scaled, quantile=0.2)`: Estimates the bandwidth (window size) for Mean Shift clustering. The `quantile` parameter specifies the quantile to use for bandwidth estimation.

- `mean_shift = MeanShift(bandwidth=bandwidth)`: Creates an instance of the Mean Shift clustering algorithm with the estimated bandwidth.
- `mean_shift.fit(features_scaled)`: Fits the Mean Shift model to the standardized features.

- `labels = mean_shift.labels_`: Retrieves the cluster labels assigned to each data point.
- `cluster_centers = mean_shift.cluster_centers_`: Retrieves the coordinates of the cluster centers.

`data['Cluster'] = labels`: Adds a new column 'Cluster' to the original DataFrame `data`, containing the cluster labels for each data point.

- `plt.figure(figsize=(10, 6))`: Creates a new figure with a specified size.
- `unique_labels = np.unique(labels)`: Retrieves the unique cluster labels.
- `colors = plt.cm.viridis(np.linspace(0, 1, len(unique_labels)))`: Generates a list of colors from the 'viridis' colormap, one for each unique cluster label.

- `for k, col in zip(unique_labels, colors)`: Iterates over each unique cluster label and corresponding color.
- `class_member_mask = (labels == k)`: Creates a boolean mask to select data points belonging to the current cluster.
- `plt.plot(features['Annual Income (k$)'][class_member_mask], features['Spending Score (1-100)'][class_member_mask], 'o', markerfacecolor=col, markeredgecolor='k', markersize=8, label=f'Cluster {k}')`: Plots the data points of the current cluster with the specified color and marker properties. Adds a label for the cluster.

- `for center in cluster_centers`: Iterates over each cluster center.
- `plt.plot(center[0] * scaler.scale_[0] + scaler.mean_[0], center[1] * scaler.scale_[1] + scaler.mean_[1], 'x', markerfacecolor='red', markeredgecolor='k', markersize=12)`: Plots each cluster center as a red 'x' marker. The centers are transformed back to the original scale.

- `plt.title('Mean Shift Clustering of Customers')`: Sets the title of the plot.
- `plt.xlabel('Annual Income (k$)')`: Sets the label for the x-axis.
- `plt.ylabel('Spending Score (1-100)')`: Sets the label for the y-axis.
- `plt.legend()`: Adds a legend to the plot.
- `plt.show()`: Displays the plot.

Gaussian Mixture Models (GMM) in Unsupervised Learning

Purpose

Gaussian Mixture Models (GMM) are used in unsupervised learning primarily for clustering and density estimation. The purpose of GMMs is to model a distribution of data points that may come from multiple underlying Gaussian distributions. Each Gaussian component in the mixture represents a cluster, and the overall model captures the data's probabilistic structure.

When to Use GMMs

GMMs are particularly useful when:

1. **Data has subgroups**: The data is believed to come from several subgroups, each of which can be modeled by a Gaussian distribution.
2. **Flexibility**: More flexibility is needed than what k-means clustering can provide. Unlike k-means, GMM can capture elliptical clusters and varying sizes and densities of clusters.
3. **Soft clustering**: Instead of hard assignment of data points to clusters, GMM provides a probability (responsibility) that a data point belongs to each cluster.

Suitable Types of Data

GMMs are suitable for continuous data where the clusters can be assumed to have Gaussian distributions. Examples include:

- Customer segmentation based on purchasing behavior.
- Image segmentation where regions of an image may have different pixel intensity distributions.
- Anomaly detection where normal data points cluster around a Gaussian distribution, and anomalies deviate from this.

Real-life Examples

1. **Customer Segmentation**: Identifying different segments of customers based on purchase history and demographics.
2. **Image Segmentation**: Segmenting an image into different regions based on color or intensity.
3. **Anomaly Detection**: Detecting unusual patterns in financial transactions to flag potential fraud.

Working Mechanism of Gaussian Mixture Models (GMM)

Overview

Gaussian Mixture Models (GMM) assume that the data is generated from a mixture of several Gaussian distributions, each with its own mean and covariance. GMMs use the Expectation-Maximization (EM) algorithm to iteratively estimate the parameters of these Gaussian components and the probability that each data point belongs to each component.

1. **Gaussian Components**: Each component kkk is a Gaussian distribution with its own mean μk\mu_kμk and covariance matrix Σk\Sigma_kΣk.
2. **Mixing Coefficients**: Each component has a weight πk\pi_kπk representing the proportion of data points that belong to that component. The mixing coefficients sum to 1.

Expectation-Maximization (EM) Algorithm

The EM algorithm is used to fit the GMM to the data. It consists of two main steps:

1. **E-Step (Expectation Step)**:
   o Calculate the responsibility γik\gamma_{ik}γik, which is the probability that the iii-th data point xix_ixi belongs to the kkk-th Gaussian component.
   o

   \gamma_{ik} = \frac{\pi_k \mathcal{N}(x_i | \mu_k, \Sigma_k)}{\sum_{j=1}^K \pi_j \mathcal{N}(x_i | \mu_j, \Sigma_j)}] Here, N(xi|μk,Σk)\mathcal{N}(x_i | \mu_k, \Sigma_k)N(xi|μk,Σk) is the probability density function of the Gaussian distribution.

2. **M-Step (Maximization Step)**:
   o Update the parameters μk\mu_kμk, Σk\Sigma_kΣk, and πk\pi_kπk using the responsibilities calculated in the E-step.
   o Update the mean: μk=∑i=1Nγikxi∑i=1Nγik\mu_k = \frac{\sum_{i=1}^N \gamma_{ik} x_i}{\sum_{i=1}^N \gamma_{ik}}μk=∑i=1Nγik∑i=1Nγikxi
   o Update the covariance: Σk=∑i=1Nγik(xi−μk)(xi−μk)T∑i=1Nγik\Sigma_k = \frac{\sum_{i=1}^N \gamma_{ik} (x_i - \mu_k)(x_i - \mu_k)^T}{\sum_{i=1}^N \gamma_{ik}}Σk=∑i=1Nγik∑i=1Nγik(xi−μk)(xi−μk)T
   o Update the mixing coefficients: πk=∑i=1NγikN\pi_k = \frac{\sum_{i=1}^N \gamma_{ik}}{N}πk=N∑i=1Nγik
3. **Repeat**:
   o Iterate the E-step and M-step until convergence, which is when the change in the log-likelihood of the data given the model parameters becomes very small.

Step-by-Step Implementation

1. Generate synthetic data.
2. Visualize the data before applying GMM.
3. Apply GMM to the data.
4. Visualize the clustered data with labels.

   ```
   import numpy as np

   import matplotlib.pyplot as plt

   from sklearn.datasets import make_blobs
   ```

```python
from sklearn.mixture import GaussianMixture

import seaborn as sns


# Generate synthetic data

X, y_true = make_blobs(n_samples=300, centers=3, cluster_std=0.60, random_state=42)


# Visualize the original data

plt.figure(figsize=(10, 6))

sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=y_true, palette='viridis', legend=None)

plt.title('Original Data')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.show()

# Fit GMM to the data

gmm = GaussianMixture(n_components=3, random_state=42)

gmm.fit(X)

labels = gmm.predict(X)


# Retrieve GMM parameters

means = gmm.means_

covariances = gmm.covariances_

weights = gmm.weights_


# Print parameters
```

```
print("Means:\n", means)

print("Covariances:\n", covariances)

print("Weights:\n", weights)

# Visualize the clustered data

plt.figure(figsize=(10, 6))

sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=labels, palette='viridis', legend=None)

plt.scatter(means[:, 0], means[:, 1], s=300, c='red', marker='x')  # Mark cluster centers

for i, mean in enumerate(means):

    plt.text(mean[0], mean[1], f'Cluster {i+1}', fontsize=12, color='black', ha='center')

plt.title('GMM Clustering')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.show()
```

Explanation

- `import numpy as np`: Imports the NumPy library, which is used for numerical operations.
- `import matplotlib.pyplot as plt`: Imports Matplotlib's pyplot module for plotting.
- `from sklearn.datasets import make_blobs`: Imports the `make_blobs` function from scikit-learn to generate synthetic data.
- `from sklearn.mixture import GaussianMixture`: Imports the `GaussianMixture` class from scikit-learn for fitting the Gaussian Mixture Model.

  - `import seaborn as sns`: Imports the Seaborn library, which is used for creating more attractive statistical graphics.

`X, y_true = make_blobs(n_samples=300, centers=3, cluster_std=0.60, random_state=42)`: Generates synthetic data using `make_blobs`. Here:

- `n_samples=300`: Creates 300 data points.
- `centers=3`: Specifies that the data should be generated around 3 cluster centers.
- `cluster_std=0.60`: Sets the standard deviation of the clusters to 0.60, controlling the spread of the data points around the centers.

- `random_state=42`: Ensures reproducibility of the results by setting a seed for the random number generator.

- `plt.figure(figsize=(10, 6))`: Creates a new figure with a specified size.
- `sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=y_true, palette='viridis', legend=None)`: Creates a scatter plot of the original data.

  - `x=X[:, 0]`: Uses the first feature of the data points for the x-axis.
  - `y=X[:, 1]`: Uses the second feature of the data points for the y-axis.
  - `hue=y_true`: Colors the points based on their true cluster labels.
  - `palette='viridis'`: Uses the 'viridis' color palette.
  - `legend=None`: Omits the legend from the plot.

- `plt.title('Original Data')`: Sets the title of the plot.
- `plt.xlabel('Feature 1')`: Labels the x-axis.
- `plt.ylabel('Feature 2')`: Labels the y-axis.

  - `plt.show()`: Displays the plot.

- `gmm = GaussianMixture(n_components=3, random_state=42)`: Creates an instance of the `GaussianMixture` class with 3 components (clusters).

  - `n_components=3`: Specifies the number of Gaussian components.
  - `random_state=42`: Sets the random seed for reproducibility.

- `gmm.fit(X)`: Fits the Gaussian Mixture Model to the data `X` using the EM algorithm.

  - `labels = gmm.predict(X)`: Predicts the cluster labels for each data point in `X` based on the fitted model.

- `means = gmm.means_`: Retrieves the means of the Gaussian components.
- `covariances = gmm.covariances_`: Retrieves the covariance matrices of the Gaussian components.
- `weights = gmm.weights_`: Retrieves the weights (mixing coefficients) of the Gaussian components.
- `print("Means:\n", means)`: Prints the means of the Gaussian components.
- `print("Covariances:\n", covariances)`: Prints the covariance matrices.

  - `print("Weights:\n", weights)`: Prints the weights.

- `plt.figure(figsize=(10, 6))`: Creates a new figure with a specified size.
- `sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=labels, palette='viridis', legend=None)`: Creates a scatter plot of the clustered data.

- `x=X[:, 0]`: Uses the first feature of the data points for the x-axis.
- `y=X[:, 1]`: Uses the second feature of the data points for the y-axis.
- `hue=labels`: Colors the points based on their predicted cluster labels.
- `palette='viridis'`: Uses the 'viridis' color palette.
- `legend=None`: Omits the legend from the plot.

- `plt.scatter(means[:, 0], means[:, 1], s=300, c='red', marker='x')`: Plots the centers of the clusters.

  - `means[:, 0]`: Uses the x-coordinates of the cluster centers.
  - `means[:, 1]`: Uses the y-coordinates of the cluster centers.
  - `s=300`: Sets the size of the markers.
  - `c='red'`: Colors the markers red.
  - `marker='x'`: Uses 'x' markers.

- `for i, mean in enumerate(means)`: Iterates over each cluster center.

  - `plt.text(mean[0], mean[1], f'Cluster {i+1}', fontsize=12, color='black', ha='center')`: Adds text labels for each cluster center.
    - `mean[0]`: x-coordinate of the cluster center.
    - `mean[1]`: y-coordinate of the cluster center.
    - `f'Cluster {i+1}'`: Text label for the cluster.
    - `fontsize=12`: Font size of the text.
    - `color='black'`: Color of the text.
    - `ha='center'`: Horizontal alignment of the text.

- `plt.title('GMM Clustering')`: Sets the title of the plot.
- `plt.xlabel('Feature 1')`: Labels the x-axis.
- `plt.ylabel('Feature 2')`: Labels the y-axis.

  - `plt.show()`: Displays the plot.