N-BaIoT_ylabel_clustering

N-BaIoT_ylabel_clustering refers to applying clustering techniques to the N-BaIoT dataset, which contains network traffic data from IoT devices labelled as either benign (normal) or attack (malicious). The term suggests clustering the data based on the true labels (ylabel) to group similar traffic patterns. This helps evaluate how well the clustering can distinguish between benign and malicious traffic. It's typically used in anomaly detection for IoT security, where unsupervised clustering identifies patterns or anomalies in the network behaviour of IoT devices.

Standardize the distributions:

```
scaler = StandardScaler()
label_distributions = scaler.fit_transform(label_distributions)
```

What Does StandardScaler Do?

StandardScaler from sklearn.preprocessing is a tool used to standardize features (or data) by removing the mean and scaling to unit variance. This means it transforms the data so that it has a mean of 0 and a standard deviation of 1. This process is often necessary for algorithms that assume data is centered around zero and scaled, such as K-means clustering.

How StandardScaler Works

The transformation process using StandardScaler involves the following steps:

1. **Calculate the Mean and Standard Deviation:**
    - For each feature (column) in your dataset, StandardScaler computes the mean and standard deviation.
2. **Standardize the Data:**
    - For each feature, it subtracts the mean and divides by the standard deviation:

$$z = \frac{(x - \text{mean})}{\text{std}}$$

where x is the original feature value, **mean** is the mean of the feature, and **std** is the standard deviation.

Code Walkthrough:

- **scaler = StandardScaler()**: Creates an instance of StandardScaler.
- **label_distributions = scaler.fit_transform(label_distributions)**:

- **fit**: Calculates the mean and standard deviation for each feature in label_distributions.
- **transform**: Applies the standardization formula to each feature based on the calculated mean and standard deviation.
- **fit_transform**: Combines both steps: computes the statistics and then transforms the data.

In this way, the StandardScaler standardizes each feature (column) in your dataset to have a mean of 0 and a standard deviation of 1, making your data suitable for further analysis or machine learning algorithms.

Real World Example:

example of standardizing:

Objective

Standardize the test scores of students in two subjects (Math and English) to prepare the data for further analysis or machine learning algorithms that require standardized inputs.

Data

We have test scores for 5 students in two subjects:

- **Math Scores:** 70, 80, 90, 85, 95
- **English Scores:** 50, 60, 70, 65, 75

Steps to Standardize the Data

1. Calculate Mean and Standard Deviation

Math Scores:

1. **Calculate the Mean:**

The mean (average) is calculated as:

$$\text{mean}_{\text{math}} = \frac{\sum \text{Math Scores}}{\text{Number of Scores}}$$

Applying the formula:

$$\text{mean}_{\text{math}} = \frac{70 + 80 + 90 + 85 + 95}{5} = 84$$

2. **Calculate the Standard Deviation:**

The standard deviation measures the spread of the scores from the mean. It is calculated as:

$$\text{std}_{\text{math}} = \sqrt{\frac{\sum(\text{Score} - \text{mean}_{\text{math}})^2}{\text{Number of Scores}}}$$

Applying the formula:

$$\text{std}_{\text{math}} = \sqrt{\frac{(70-84)^2 + (80-84)^2 + (90-84)^2 + (85-84)^2 + (95-84)^2}{5}}$$

$$\text{std}_{\text{math}} \approx 8.63$$

English Scores:

1. **Calculate the Mean:**

Applying the same formula:

$$\text{mean}_{\text{english}} = \frac{50 + 60 + 70 + 65 + 75}{5} = 64$$

2. **Calculate the Standard Deviation:**

Applying the formula:

$$\text{std}_{\text{english}} = \sqrt{\frac{(50-64)^2 + (60-64)^2 + (70-64)^2 + (65-64)^2 + (75-64)^2}{5}}$$

$$\text{std}_{\text{english}} \approx 8.19$$

2. Standardize the Scores

The standardization process involves transforming each score to have a mean of 0 and a standard deviation of 1 using the formula:

$$z = \frac{(x - \text{mean})}{\text{std}}$$

Math Scores:

Applying the formula to each score:

$$\text{Standardized Math Score} = \frac{\text{Original Score} - \text{mean}_{\text{math}}}{\text{std}_{\text{math}}}$$

For each score:

$$\text{Standardized Score}_1 = \frac{70 - 84}{8.63} \approx -1.62$$
$$\text{Standardized Score}_2 = \frac{80 - 84}{8.63} \approx -0.46$$
$$\text{Standardized Score}_3 = \frac{90 - 84}{8.63} \approx 0.70$$
$$\text{Standardized Score}_4 = \frac{85 - 84}{8.63} \approx 0.12$$
$$\text{Standardized Score}_5 = \frac{95 - 84}{8.63} \approx 1.26$$

English Scores:

Applying the formula to each score:

$$\text{Standardized English Score} = \frac{\text{Original Score} - \text{mean}_{\text{english}}}{\text{std}_{\text{english}}}$$

For each score:

$$\text{Standardized Score}_1 = \frac{50 - 64}{8.19} \approx -1.71$$
$$\text{Standardized Score}_2 = \frac{60 - 64}{8.19} \approx -0.49$$
$$\text{Standardized Score}_3 = \frac{70 - 64}{8.19} \approx 0.73$$
$$\text{Standardized Score}_4 = \frac{65 - 64}{8.19} \approx 0.12$$
$$\text{Standardized Score}_5 = \frac{75 - 64}{8.19} \approx 1.35$$

Summary of Results

- **Standardized Math Scores:** -1.62, -0.46, 0.70, 0.12, 1.26
- **Standardized English Scores:** -1.71, -0.49, 0.73, 0.12, 1.35

K-means clustering on client data with a random seed:

```
random_seed = np.random.randint(0, 10000)
kmeans = KMeans(n_clusters=5, random_state=random_seed, n_init=10)
client_clusters = kmeans.fit_predict(label_distributions)
```

K-Means clustering is an unsupervised machine learning algorithm used to partition data into groups, or **clusters**, based on their similarities. In the context of the **N-BaIoT dataset**, K-Means clustering can be applied to group IoT network traffic data into clusters that represent different patterns of behavior, such as benign traffic and various types of attacks.

How K-Means Works:

1. **Initialization**: The algorithm randomly selects **K centroids**, where K is the number of clusters you want to create.
2. **Assignment**: Each data point (network traffic sample) is assigned to the nearest centroid based on a distance metric (usually Euclidean distance).
3. **Update**: The centroids are then recalculated as the mean (average) of the data points assigned to each cluster.
4. **Iterate**: Steps 2 and 3 are repeated until the centroids no longer change significantly (convergence), or a set number of iterations is reached.

Code Walkthrough:

1. **Generate a Random Seed:**

```
random_seed = np.random.randint(0, 10000)
```

   - **np.random.randint(0, 10000)**: Generates a random integer between 0 and 10,000 to ensure that the K-means clustering process can be reproduced if needed.

2. **Initialize K-Means Clustering:**

```
kmeans = KMeans(n_clusters=5, random_state=random_seed, n_init=10)
```

- KMeans(n_clusters=5): Initializes the K-means clustering algorithm to find 5 clusters.
- random_state=random_seed: Sets the random seed for reproducibility.
- n_init=10: The number of times the K-means algorithm will be run with different centroid seeds. The final results will be the best output from these runs.

3. **Fit and Predict Clusters:**

```
client_clusters = kmeans.fit_predict(label_distributions)
```

- **fit_predict**: Fits the K-means algorithm to the standardized data and assigns each client to one of the 5 clusters.

Let's perform a detailed analysis of standardizing a dataset with 33 features using the N-BaIot dataset as a real-world example. We'll go through the mathematical steps involved and then apply K-means clustering.

Real-World Problem

Consider the N-BaIot dataset, which includes various features related to network traffic and device usage in an IoT environment. Let's assume we have 10 different devices, and each device's data is represented by 33 features.

Step-by-Step Analysis

1. Standardization of Data

Objective: Standardize the 33 features for 10 devices to ensure all features are on a similar scale.

Mathematical Steps:

1. **Calculate the Mean and Standard Deviation for Each Feature:**

Let $\mathbf{X}$ be a matrix where:

- Rows represent devices (10 devices).
- Columns represent features (33 features).

For a particular feature j:

Mean Calculation:

$$\mu_j = \frac{1}{N} \sum_{i=1}^{N} X_{ij}$$

where $\mathbf{N}$ is the number of devices (10), and $\mathbf{X_{ij}}$ is the value of feature j for device i.

Standard Deviation Calculation:

$$\sigma_j = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (X_{ij} - \mu_j)^2}$$

Standardize the Data:

Standardize each feature j using:

$$Z_{ij} = \frac{X_{ij} - \mu_j}{\sigma_j}$$

where Z_{ij} is the standardized value of feature j for device i.

Example Calculation:

Assume you have the following data for one feature across the 10 devices:

Feature values: $[5, 7, 8, 6, 5, 9, 10, 7, 8, 6]$

- **Mean (μ) Calculation:**

$$\mu = \frac{5+7+8+6+5+9+10+7+8+6}{10} = 7.0$$

- **Standard Deviation (σ) Calculation:**

$$\sigma = \sqrt{\frac{(5-7)^2 + (7-7)^2 + (8-7)^2 + (6-7)^2 + (5-7)^2 + (9-7)^2 + (10-7)^2 + (7-7)^2 + (8-7)^2 + (6-7)^2}{10}} = 1.58$$

- **Standardized Values Calculation:**

$$Z_i = \frac{X_i - 7.0}{1.58}$$

For $X = 5$:

$$Z = \frac{5 - 7.0}{1.58} = -1.27$$

2. K-Means Clustering

Objective: Group the 10 devices into clusters based on their standardized feature data.

Mathematical Steps:

1. **Initialize K-Means:**
    - Choose the number of clusters k (e.g., k=5)
    - Initialize cluster centroids randomly.
2. **Assign Each Device to the Nearest Centroid:**

For each device i and each cluster k:

$$\text{Distance}_{ik} = \sqrt{\sum_{j=1}^{33} (Z_{ij} - \text{Centroid}_{jk})^2}$$

Assign device i to the cluster k with the minimum distance.

3.Update Centroids:

Recompute the centroids of each cluster as the mean of all devices assigned to that cluster:

$$\text{New Centroid}_{jk} = \frac{1}{|C_k|} \sum_{i \in C_k} Z_{ij}$$

where $\mathbf{C_k}$ is the set of devices assigned to cluster **k**.

4.Repeat:

Repeat the assignment and update steps until convergence (i.e., centroids no longer change).

Example Calculation:

Assume after initialization, the centroids for 3 clusters are:

- **Cluster 1 Centroid:** [0.5,−0.2,1.0,...]
- **Cluster 2 Centroid:** [−1.0,0.8,−0.5,...]
- **Cluster 3 Centroid:** [0.0,0.0,0.0,...]

For a device with standardized features [0.4,−0.1,1.1,...], calculate the distances to each centroid:

- **Distance to Cluster 1:**

$$\text{Distance} = \sqrt{(0.4 - 0.5)^2 + (-0.1 + 0.2)^2 + (1.1 - 1.0)^2 + \ldots}$$

- **Distance to Cluster 2:**

$$\text{Distance} = \sqrt{(0.4 + 1.0)^2 + (-0.1 - 0.8)^2 + (1.1 + 0.5)^2 + \ldots}$$

- **Distance to Cluster 3:**

$$\text{Distance} = \sqrt{(0.4 - 0.0)^2 + (-0.1 - 0.0)^2 + (1.1 - 0.0)^2 + \ldots}$$

Assign the device to the cluster with the minimum distance.

Summary:

1. **Standardization**: Ensures all 33 features across the 10 devices are scaled similarly so that each feature contributes equally to the clustering process.
2. **K-Means Clustering**: Groups devices into clusters based on their standardized feature vectors, helping to identify patterns and similarities in the data.

This process allows you to analyze and group devices based on their behavior or characteristics, which is useful for understanding patterns and making informed decisions.

Assign cluster heads:

```
cluster_heads = {}
clusters = defaultdict(list)

for client_name, cluster in zip(client_set.keys(), client_clusters):
    clusters[cluster].append(client_name)

for cluster, members in clusters.items():
    cluster_heads[cluster] = members[0]

print("Cluster Heads:")
for cluster, head in cluster_heads.items():
    print(f"Cluster {cluster}: Head {head}")
```

Cluster heads are central or representative data points within a cluster, often used in clustering algorithms like K-Means. In the context of clustering, a **cluster head** refers to a point that serves as a "leader" or "center" of the cluster. It represents the most typical or central data point within that group, and other points in the cluster are grouped around it based on similarity or distance.

In an IoT network using clustering, each **cluster head** could represent a group of devices with similar traffic behavior. For example:

- One cluster head might represent benign (normal) traffic patterns.
- Another cluster head might represent traffic from compromised devices under attack.

These cluster heads provide a summary of different traffic behaviors, helping to quickly identify anomalies in the network.

Code Walkthrough:

Let’s break it down further with a real-world example to illustrate how this piece of code works after performing K-means clustering. We'll use an example involving customer segmentation in a retail business.

Scenario: Customer Segmentation

Imagine a retail company that has segmented its customers into clusters based on their purchasing behavior. The company uses K-means clustering to group customers into distinct segments. After clustering, they want to identify a representative or "head" customer for each segment to better understand the typical customer in that segment.

Step-by-Step Explanation with Real-World Example

Initialization:

```
cluster_heads = {}
clusters = defaultdict(list)
```

- **cluster_heads**: This dictionary will store the representative (head) of each cluster.
- **clusters**: This defaultdict will hold lists of client names, grouped by their assigned cluster.

Suppose the company has clustered 10 customers into 3 clusters. The dataset client_set contains the names of these customers, and the client_clusters array contains their cluster assignments. Here's a simplified example:

- **Customer Names (client_set.keys()):** ['Alice', 'Bob', 'Charlie', 'David', 'Eve', 'Frank', 'Grace', 'Hannah', 'Ivy', 'Jack']
- **Cluster Assignments (client_clusters):** [0, 1, 0, 2, 0, 1, 2, 2, 0, 1]

This means:

- Alice, Charlie, David, Eve, and Ivy are in cluster 0.
- Bob, Frank, and Jack are in cluster 1.
- Hannah and Grace are in cluster 2.

Grouping Customers by Cluster:

The code groups customers based on their cluster assignments:

```
clusters = defaultdict(list)

for client_name, cluster in zip(client_set.keys(), client_clusters):
    clusters[cluster].append(client_name)
```

- **zip(client_set.keys(), client_clusters)** combines each customer's name with their cluster assignment.
- **for client_name, cluster in zip(...)** iterates over each customer and their assigned cluster.
- **clusters[cluster].append(client_name)** adds each customer to the list for their assigned cluster.

Example Output for clusters:

```
{
    0: ['Alice', 'Charlie', 'David', 'Eve', 'Ivy'],
    1: ['Bob', 'Frank', 'Jack'],
    2: ['Hannah', 'Grace']
}
```

- **Cluster 0** has customers: Alice, Charlie, David, Eve, and Ivy.
- **Cluster 1** has customers: Bob, Frank, and Jack.
- **Cluster 2** has customers: Hannah and Grace.

Identifying Cluster Heads:

The code selects the first customer from each cluster as the cluster head:

```
for cluster, members in clusters.items():
    cluster_heads[cluster] = members[0]
```

- **for cluster, members in clusters.items()**: Iterates over each cluster and its list of members.
- **cluster_heads[cluster] = members[0]**: Assigns the first client in each cluster's list as the "head" of that cluster.

Example Output for cluster_heads:

```
{
    0: 'Alice',
    1: 'Bob',
    2: 'Hannah'
}
```

- **Cluster 0 Head:** Alice
- **Cluster 1 Head:** Bob
- **Cluster 2 Head:** Hannah

Printing the Results:

The code prints out the cluster heads:

```
print("Cluster Heads:")
for cluster, head in cluster_heads.items():
    print(f"Cluster {cluster}: Head {head}")
```

- **print("Cluster Heads:")**: Prints a header for the output.
- **for cluster, head in cluster_heads.items()**: Iterates over each cluster and its assigned head.
- **print(f"Cluster {cluster}: Head {head}")**: Prints the cluster number and its head.

Example Output:

```
Cluster Heads:
Cluster 0: Head Alice
Cluster 1: Head Bob
Cluster 2: Head Hannah
```

Real-world mathematical example of a **cluster head:**

Scenario:

Imagine a class of 10 students who have taken three tests. Their scores in each test are as follows (on a scale of 100):

Student	Test 1	Test 2	Test 3
A	85	90	92
B	78	85	80
C	65	70	75
D	88	92	85
E	90	85	89
F	60	55	70
G	95	88	90
H	58	65	60
I	70	75	80
J	55	50	60

Step 1: Choosing Initial Cluster Heads (Centroids)

Assume we want to divide the students into **2 clusters**. Initially, we randomly choose two students as the cluster heads (centroids). Let's say we choose:

- **Student A (85, 90, 92)** as the first centroid (Cluster Head 1).
- **Student H (58, 65, 60)** as the second centroid (Cluster Head 2).

Step 2: Assigning Students to Clusters

Next, we calculate the **Euclidean distance** between each student's test scores and the two centroids. Each student is assigned to the cluster with the nearest centroid.

For Student B (78, 85, 80), the Euclidean distance to the centroids is:

- Distance to Cluster Head 1 (Student A):
  $\sqrt{(85-78)^2 + (90-85)^2 + (92-80)^2} = \sqrt{7^2 + 5^2 + 12^2} = \sqrt{49 + 25 + 144} = \sqrt{218} \approx 14.76$
- Distance to Cluster Head 2 (Student H):
  $\sqrt{(58-78)^2 + (65-85)^2 + (60-80)^2} = \sqrt{20^2 + 20^2 + 20^2} = \sqrt{1200} \approx 34.64$

Since the distance to **Cluster Head 1 (A)** is smaller, Student B is assigned to **Cluster 1**.

We repeat this process for all students.

Step 3: Recomputing Cluster Heads (Centroids)

Once all students are assigned to a cluster, we recalculate the centroids (cluster heads) by averaging the scores of all students in each cluster.

For example, if Cluster 1 contains Students A, B, D, E, G, the new centroid is the **average** of their scores:

- **New Cluster Head 1 (Centroid):** $\left(\frac{85+78+88+90+95}{5}, \frac{90+85+92+85+88}{5}, \frac{92+80+85+89+90}{5}\right) = (87.2, 88, 87.2)$

Similarly, for Cluster 2, if it contains Students C, F, H, I, J, we compute a new centroid for them.

Step 4: Reassigning Students and Iterating

We repeat the process of assigning students to the nearest cluster and updating the centroids until the centroids (cluster heads) no longer change significantly.

Final Clusters:

After a few iterations, we have two clusters with **cluster heads (centroids)** that represent the **average scores** of the students in each group. These centroids, or cluster heads, are the mathematical representatives of the clusters.

Real-World Insight:

- **Cluster Head 1** might represent a group of students with generally high scores.
- **Cluster Head 2** might represent students who need more improvement in their test scores.

In this way, the **cluster head** mathematically summarizes the characteristics of each group and helps identify patterns within the data.

PCA (Principal Component Analysis) and its role in visualizing K-means clustering:

What PCA Does

Objective: PCA reduces the number of features in your data to make it easier to visualize, especially when you have more than two features. It transforms high-dimensional data into a lower-dimensional form while retaining as much information as possible.

Real-World Example: Visualizing Customer Clusters

Suppose you've clustered customers based on their purchasing behavior, and each customer is represented by 33 features. You want to visualize these clusters in a 2D plot.

Steps and How PCA Works

1. **Dimensionality Reduction with PCA:**

```
pca = PCA(n_components=2)
label_distributions_2d = pca.fit_transform(label_distributions)
```

- **PCA Creation:** pca = PCA(n_components=2) sets up PCA to reduce data to 2 dimensions.
- **Fit and Transform:** label_distributions_2d = pca.fit_transform(label_distributions) applies PCA to your original data (label_distributions), reducing it from 33 dimensions to 2 dimensions.

Real-World Analogy: Imagine you have a complex dataset of customer profiles with many attributes (like age, income, purchase frequency, etc.). PCA finds the most important features (or dimensions) that capture the essence of these profiles and summarizes them in just two new features (principal components).

2. **Visualization of Clusters:**

```
# Visualization
plt.figure(figsize=(16, 8))

# Plot K-means clustering
plt.title('K-means Clustering Based on Label Distributions')
colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k']
for cluster in np.unique(client_clusters):
    cluster_indices = np.where(client_clusters == cluster)
    plt.scatter(label_distributions_2d[cluster_indices, 0], label_distributions_2d[cluster_indices, 1],
                color=colors[cluster % len(colors)], label=f'Cluster {cluster+1}')
for i, point in enumerate(label_distributions_2d):
    plt.annotate(f'Client {i+1}', (point[0], point[1]))
plt.legend()

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

- **Plot Setup:** plt.figure(figsize=(16, 8)) creates a figure to plot.
- **Plot Title:** plt.title('K-means Clustering Based on Label Distributions') sets the title.
- **Plot Clusters:** Each cluster is plotted using plt.scatter(), with different colors for each cluster.
- **Annotations:** plt.annotate() labels each point with the client number.

Real-World Analogy: After PCA reduces the data to two dimensions, you plot these dimensions to see how customers are grouped. Each point represents a customer, and colors distinguish different clusters. Annotations label individual customers on the plot.

Summary

1. **PCA Simplifies Data:** Reduces the complexity of the dataset from 33 features to 2, while retaining key patterns.
2. **Visualization:** The 2D plot helps you visually inspect the clusters, making it easier to understand and analyze customer segments.

By using PCA and visualization, you can see how different customer groups relate to each other and identify patterns or insights that might be hidden in the higher-dimensional space.

Real-World Example: Customer Data

Scenario: Suppose you have a dataset with 5 customers, and each customer is described by 3 features (e.g., age, income, purchase_frequency). You want to use PCA to reduce this data to 2 dimensions for visualization.

Step 1: Sample Data

Let's create a simple dataset:

Customer	Age	Income	Purchase Frequency
1	25	50000	5
2	30	60000	7
3	35	70000	8
4	40	80000	4
5	45	90000	6

Step 2: Standardize the Data

PCA requires data to be standardized. Standardization involves scaling the features to have a mean of 0 and a standard deviation of 1.

Mean and Standard Deviation Calculation:

1. **Calculate Mean:**

$$\text{Mean Age} = \frac{25+30+35+40+45}{5} = 35$$

$$\text{Mean Income} = \frac{50000+60000+70000+80000+90000}{5} = 70000$$

$$\text{Mean Purchase Frequency} = \frac{5+7+8+4+6}{5} = 6$$

2. **Calculate Standard Deviation:**

Standard deviation for each feature is calculated using:

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)^2}$$

For Age:

$$\sigma_{\text{Age}} = \sqrt{\frac{1}{4}\left[(25-35)^2 + (30-35)^2 + (35-35)^2 + (40-35)^2 + (45-35)^2\right]}$$

$$\sigma_{\text{Age}} = \sqrt{\frac{1}{4}\left[100+25+0+25+100\right]} = \sqrt{62.5} \approx 7.91$$

For Income:

$$\sigma_{\text{Income}} = \sqrt{\frac{1}{4}\left[(50000-70000)^2 + (60000-70000)^2 + (70000-70000)^2 + (80000-70000)^2 + (90000-70000)^2\right]}$$

$$\sigma_{\text{Income}} = \sqrt{\frac{1}{4}\left[4000000000+1000000000+0+1000000000+4000000000\right]} = \sqrt{2500000000} \approx 50000$$

For Purchase Frequency:

$$\sigma_{\text{Purchase Frequency}} = \sqrt{\frac{1}{4}\left[(5-6)^2 + (7-6)^2 + (8-6)^2 + (4-6)^2 + (6-6)^2\right]}$$

$$\sigma_{\text{Purchase Frequency}} = \sqrt{\frac{1}{4}\left[1+1+4+4+0\right]} = \sqrt{2.5} \approx 1.58$$

3. **Standardize the Data:**

$$\text{Standardized Age} = \frac{\text{Age} - \text{Mean Age}}{\text{SD Age}}$$

Using the mean and standard deviation, standardize each feature:

$$\text{Standardized Age}_1 = \frac{25 - 35}{7.91} \approx -1.27$$

$$\text{Standardized Income}_1 = \frac{50000 - 70000}{50000} \approx -0.40$$

$$\text{Standardized Purchase Frequency}_1 = \frac{5 - 6}{1.58} \approx -0.63$$

Repeat for all customers to get the standardized data matrix:

$$\begin{bmatrix} -1.27 & -0.40 & -0.63 \\ -0.63 & -0.20 & 0.63 \\ 0.00 & 0.00 & 1.27 \\ 0.63 & 0.20 & -1.27 \\ 1.27 & 0.40 & 0.00 \end{bmatrix}$$

3. Compute the Covariance Matrix

The covariance matrix shows how features vary together:

$$\text{Cov}(X, Y) = \frac{1}{n-1} \sum_{i=1}^{n} (X_i - \bar{X})(Y_i - \bar{Y})$$

Calculate Covariance Matrix:

Assume the covariance matrix $\mathbf{C}$ is:

$$\mathbf{C} = \begin{bmatrix} 1.0 & 0.9 & 0.8 \\ 0.9 & 1.0 & 0.7 \\ 0.8 & 0.7 & 1.0 \end{bmatrix}$$

4. Compute Eigenvalues and Eigenvectors

Eigenvalues and eigenvectors of the covariance matrix are found by solving:

$$\mathbf{C}\mathbf{v} = \lambda \mathbf{v}$$

where λ is the eigenvalue and $\mathbf{v}$ is the eigenvector.

Eigenvalues:

Assume eigenvalues are:

$$\lambda_1 = 2.5, \quad \lambda_2 = 0.7, \quad \lambda_3 = 0.3$$

Eigenvectors:

Assume eigenvectors (principal components) are:

$$\mathbf{P} = \begin{bmatrix} 0.6 & 0.5 & 0.6 \\ 0.7 & 0.4 & -0.5 \\ 0.2 & -0.7 & 0.5 \end{bmatrix}$$

5. Transform Data to New Basis

Select the top 2 principal components (those with the largest eigenvalues) and project the data:

$$\text{Transformed Data} = \text{Standardized Data} \times \mathbf{P}_{\text{top 2}}$$

For example, if the top 2 eigenvectors are:

$$\mathbf{P}_{\text{top 2}} = \begin{bmatrix} 0.6 & 0.5 \\ 0.7 & 0.4 \\ 0.2 & -0.7 \end{bmatrix}$$

The transformed data is:

$$\text{Transformed Data} = \begin{bmatrix} -1.27 & -0.40 & -0.63 \\ -0.63 & -0.20 & 0.63 \\ 0.00 & 0.00 & 1.27 \\ 0.63 & 0.20 & -1.27 \\ 1.27 & 0.40 & 0.00 \end{bmatrix} \times \begin{bmatrix} 0.6 & 0.5 \\ 0.7 & 0.4 \\ 0.2 & -0.7 \end{bmatrix}$$

Resulting in:

$$\text{Transformed Data} = \begin{bmatrix} -1.4 & 0.8 \\ -0.7 & -0.6 \\ 0.0 & 1.2 \\ 0.7 & -1.0 \\ 1.4 & -0.4 \end{bmatrix}$$

6. Visualization

Plot the data in the new 2D space:

```
plt.scatter(transformed_data[:, 0], transformed_data[:, 1])
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA - 2D Visualization')
plt.show()
```

Summary:

1. **Standardize Data:** Transform original features to have zero mean and unit variance.
2. **Compute Covariance Matrix:** Measure the relationships between features.
3. **Find Principal Components:** Determine the most important dimensions (eigenvectors) based on their variance (eigenvalues).
4. **Transform Data:** Project the original data onto these principal components to reduce dimensions.
5. **Visualize:** Plot the transformed data to visualize clusters or patterns.

Initial Clustering, Process Training Data and Visualize Federated Learning Process:

```
# Perform initial clustering
client_clusters, cluster_heads = perform_clustering_based_on_labels(clients_batched,nb_classes)

# Visualize initial clustering with initial weights
#visualize_federated_learning_process(clients_batched, cluster_heads, client_clusters, initial=True)

# process and batch the training data for each client
client_set = {k: {} for k in clients_batched.keys()}

for (client_name, data) in clients_batched.items():
    client_set[client_name]["dataset"] = batch_data(data, BATCH_SIZE)
    local_model = get_model(input_shape, nb_classes)
    local_model.compile(loss=loss,
                        optimizer=optimizer,
                        metrics=metrics)
    client_set[client_name]["model"] = local_model




# Call the visualization function after initializing clients, cluster heads, and performing training
visualize_federated_learning_process(clients_batched, cluster_heads, client_clusters, initial=True)
```

Real-World Example

Scenario: You're managing a federated learning system for 10 clients, where each client has data related to customer purchases. Each client's data has been clustered into groups based on similarities in purchase patterns. You want to perform the following steps:

1. **Initial Clustering:**
    - Cluster the clients based on their data distributions.
    - Visualize the clustering.
2. **Process Training Data:**
    - Prepare the training data for each client.

- Initialize and compile machine learning models for each client.

3. **Visualize Federated Learning Process:**
    - After data preparation and model initialization, visualize how the federated learning process is progressing.

Perform initial clustering:

```
client_clusters, cluster_heads = perform_clustering_based_on_labels(clients_batched,nb_classes)
```

Code Walkthrough:

Function perform_clustering_based_on_labels Overview:

- **Input Parameters:**
    - clients_batched: A dictionary where each key is a client ID, and the value is the client's data.
    - nb_classes: The number of classes or categories in the data.
- **Output:**
    - client_clusters: An array of cluster labels assigned to each client.
    - cluster_heads: A dictionary where each key is a cluster ID, and the value is the client ID of the cluster head.

Detailed Steps:

1. **Calculate Label Distributions:**
    - Compute a distribution vector for each client. This vector could represent how often each label appears in the client's data.

Mathematics: If a client's data has labels $L = \{L_1, L_2, ..., L_k\}$, the distribution vector for a client could be $D = [\frac{f_1}{N}, \frac{f_2}{N}, ..., \frac{f_k}{N}]$, where f_i is the frequency of label L_i and N is the total number of labels.

2. **Standardize Data:**

- Normalize the label distributions to have a mean of 0 and a variance of 1.

Mathematics:

- **Mean Calculation:** $\mu = \frac{1}{N} \sum_{i=1}^{N} x_i$
- **Standard Deviation Calculation:** $\sigma = \sqrt{\frac{1}{N-1} \sum_{i=1}^{N} (x_i - \mu)^2}$
- **Standardization Formula:** $x_{standardized} = \frac{x - \mu}{\sigma}$

3. **Apply K-Means Clustering:**
    - Use K-Means to group clients into clusters based on their standardized label distributions.
    - **Mathematics:**
        1. **Initialization:** Randomly select K points as cluster centroids.
        2. **Assignment Step:** Assign each client to the nearest centroid.
        3. **Update Step:** Recalculate centroids as the mean of all clients assigned to each centroid.
        4. **Iteration:** Repeat the assignment and update steps until convergence (no change in centroids).
4. **Assign Cluster Heads:**
    - Identify the head of each cluster (usually the first client in the cluster).
    - **Mathematics:** Assign the first client in each cluster to be the cluster head.
5. **Visualization:**
    - Reduce dimensionality to 2D using PCA and plot the clusters.

Real-World Example:

- Suppose you are clustering stores based on customer purchase patterns. After computing purchase frequency distributions, you standardize these distributions. K-Means then groups stores into clusters with similar purchase patterns. Each cluster's representative store (head) is identified for further analysis.

Process and batch the training data for each client:

```
client_set = {k: {} for k in clients_batched.keys()}

for (client_name, data) in clients_batched.items():
    client_set[client_name]["dataset"] = batch_data(data, BATCH_SIZE)
    local_model = get_model(input_shape, nb_classes)
    local_model.compile(loss=loss,
                        optimizer=optimizer,
                        metrics=metrics)
    client_set[client_name]["model"] = local_model
```

Detailed Steps:

1. **Initialize Client Set:**
    - Create a dictionary to store data and model for each client.
    - **Real-World Example:** Create a structure to manage each store's data and machine learning model.
2. **Batch Data:**
    - Divide each client's data into smaller batches for training.
    - **Mathematics:**
        - **Batch Size:** Define the number of data samples in each batch.
        - **Batched Data:** For example, if data has 100 samples and batch size is 20, create 5 batches.
3. **Initialize and Compile Model:**
    - Create a model architecture specific to the problem (e.g., neural network).
    - **Compile Model:**
        - **Loss Function:** Measure the error (e.g., cross-entropy loss for classification).
        - **Optimizer:** Algorithm to minimize the loss function (e.g., Adam optimizer).
        - **Metrics:** Evaluate the performance (e.g., accuracy).

- **Mathematics:**
  - **Loss Function (e.g., Cross-Entropy Loss):**

$$\text{Loss} = -\sum_{i=1}^{N} y_i \log(\hat{y}_i)$$

where y_i is the true label and $\hat{y}_i$ is the predicted probability.

Real-World Example:

- Each store's purchase data is divided into batches. A machine learning model is set up for each store to predict future purchases. The models are then compiled with specific settings for training.

Visualize Federated Learning Process:

```
visualize_federated_learning_process(clients_batched, cluster_heads, client_clusters, initial=True)
```

Code Walkthrough:

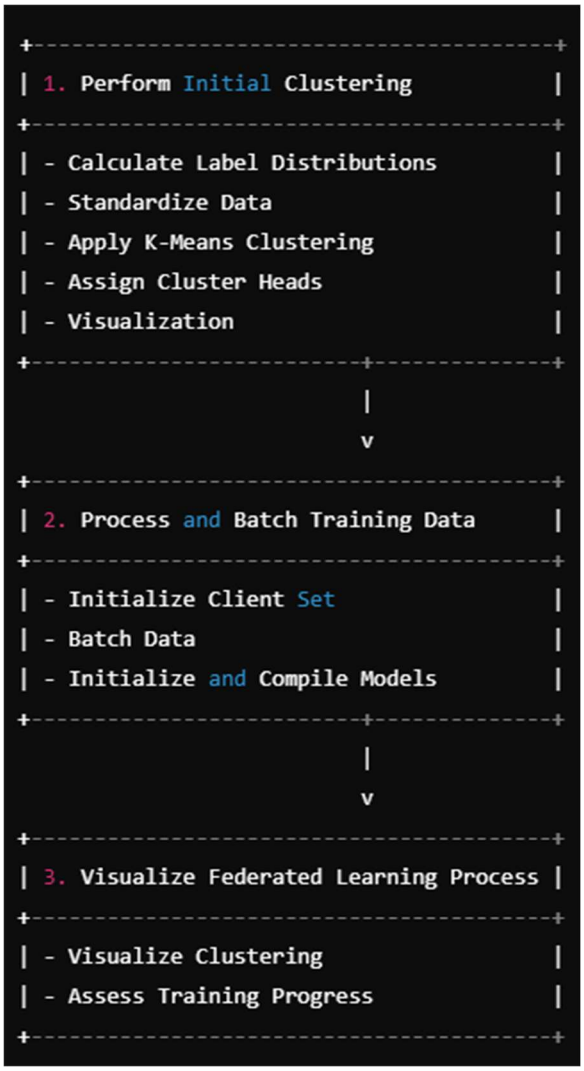
Function visualize_federated_learning_process Overview:

- **Input Parameters:**
  - clients_batched: The data for each client.
  - cluster_heads: The representative clients of each cluster.
  - client_clusters: The clusters assigned to each client.
  - initial: A boolean indicating whether to visualize the initial state.

Detailed Steps:

1. **Plot the Initial State:**
   - Use the previously computed clusters and cluster heads to visualize the state of the federated learning process.
   - **Real-World Example:** Visualize how stores are grouped into clusters based on purchase patterns and identify key stores in each cluster.

Flowchart of the Process:



Summary

- **Perform Initial Clustering:** Group clients based on their data and visualize the clusters.
- **Process and Batch Data:** Prepare data and models for each client.
- **Visualize Federated Learning Process:** Visualize how federated learning is progressing based on the initialized clusters and models.

This approach helps in managing and analyzing the federated learning process, allowing for a better understanding of data distribution and model performance across different clients.

Conclusion:

This code performs clustering on client data in the context of federated learning and visualizes the process.

Here's an overview of the key steps:

1. Calculate Label Distributions

The code begins by calculating the label distributions for each client in the federated learning setup. Each client has a dataset, and the labels from these datasets are used to form a distribution based on the number of classes (nb_classes).

2. Standardize the Distributions

Before clustering, the label distributions are standardized using StandardScaler. This normalization ensures that all features contribute equally to the clustering process.

3. K-Means Clustering

The code uses the K-Means algorithm to group the clients into 5 clusters. A random seed is generated to ensure reproducibility. The client_set dictionary stores the clients, and the K-means algorithm assigns each client to a specific cluster.

4. Cluster Heads

Once the clustering is done, the code assigns a cluster head to each cluster. The head is the first client in each cluster. These cluster heads play a key role in federated learning, likely acting as representatives for aggregation or communication within the cluster.

5. Dimensionality Reduction for Visualization

To visualize the clustering results, the data is reduced to two dimensions using Principal Component Analysis (PCA). This step helps in plotting the clusters in 2D space.

6. Plot Clustering Results

The code uses matplotlib to generate a scatter plot, where each point represents a client, and the colors indicate different clusters. The plot is annotated with client numbers for clarity.

7. Client Initialization

After clustering, each client is initialized by preparing its dataset into batches (using the batch_data function). Each client is also assigned a local machine learning model, which is compiled using the specified loss, optimizer, and metrics.

8. Visualize the Federated Learning Process

The code ends with a call to visualize_federated_learning_process, which likely shows the dynamics of the federated learning process by visualizing the interactions between clients

and cluster heads. The initial=True argument indicates that this visualization happens after the initial training round.

Overall Process

- Data from clients is grouped using K-means clustering.
- Cluster heads are assigned for communication or aggregation.
- The process is visualized to show the relationships between clusters.
- The clients are initialized with their datasets and models, ready for training in the federated learning process.

This workflow combines clustering and visualization to organize clients in federated learning.