1. Introduction to Clustering

1.1 What is Clustering?

1.1.1 Definition and Purpose

Clustering is an unsupervised machine learning technique that involves grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar to each other than to those in other groups (clusters). The main goal of clustering is to identify the intrinsic grouping in a set of unlabelled data.

Purpose of Clustering:

- **Discover Patterns**: Identify natural groupings in data to discover hidden patterns.
- **Data Reduction**: Simplify large datasets by categorizing them into clusters.
- **Insight Generation**: Provide insights for decision-making and exploratory data analysis.
- **Preprocessing Step**: Serve as a preprocessing step for other algorithms.

1.1.2 Difference Between Clustering and Classification

- **Clustering**:
    - **Type**: Unsupervised learning.
    - **Labels**: No predefined labels; groups data based on similarity.
    - **Goal**: Discover the inherent structure in data without prior knowledge.
    - **Example**: Grouping customers based on purchasing behavior.
- **Classification**:
    - **Type**: Supervised learning.
    - **Labels**: Uses predefined labels to train the model.
    - **Goal**: Predict the category of new observations based on trained data.
    - **Example**: Classifying emails as spam or non-spam.

1.2 Use Cases of Clustering

1.2.1 Applications in Various Domains

- **Customer Segmentation**:
    - **Description**: Group customers based on their behaviors and characteristics.

    - **Purpose**: Tailor marketing strategies, personalize customer experiences, and improve customer service.

- **Image Compression**:
    - **Description**: Reduce the number of colors in an image by clustering similar colors.
    - **Purpose**: Compress images to save storage space while maintaining image quality.
- **Document Clustering**:
    - **Description**: Organize a large collection of documents into groups based on content similarity.
    - **Purpose**: Improve information retrieval and facilitate topic modeling.
- **Anomaly Detection**:
    - **Description**: Identify outliers or unusual patterns in data.
    - **Purpose**: Detect fraud, network intrusions, or equipment failures.
- **Genomic Data Analysis**:
    - **Description**: Group genes or proteins with similar expression patterns.
    - **Purpose**: Understand biological processes and disease mechanisms.
- **Social Network Analysis**:
    - **Description**: Identify communities or groups within a social network.
    - **Purpose**: Analyze the structure and dynamics of social interactions.
- **Market Basket Analysis**:
    - **Description**: Find groups of items that frequently co-occur in transactions.
    - **Purpose**: Enhance recommendation systems and optimize inventory management.

Clustering is a versatile tool used across various fields to explore and analyze data, uncovering hidden structures and generating valuable insights.

2. k-Means Clustering

2.1 Understanding k-Means Clustering

2.1.1 Basic Concept and Intuition

k-Means Clustering is an iterative algorithm that partitions a dataset into k distinct, non-overlapping subsets (clusters). The algorithm aims to minimize the variance within each cluster. The basic idea is to define k centroids, one for each cluster, and assign each data point

to the nearest centroid, then update the centroids based on the mean of the assigned points. This process is repeated until the centroids no longer change significantly.

2.1.2 Choosing the Number of Clusters (k)

Choosing the right number of clusters k is crucial for effective clustering. Methods to determine k include:

- **Elbow Method**: Plot the within-cluster variance against the number of clusters and look for an "elbow" point where the variance reduction slows down.
- **Silhouette Score**: Measures how similar a data point is to its own cluster compared to other clusters. A higher score indicates better-defined clusters.
- **Cross-Validation**: Use a validation set to evaluate clustering performance for different values of k.

2.1.3 Objective Function (Minimizing Within-Cluster Variance)

The objective function of k-means clustering is to minimize the sum of squared distances between each data point and the centroid of the cluster to which it belongs

$$\text{Objective Function} = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2$$

where μ_i is the centroid of cluster C_i, and x is a data point in C_i.

2.2 Algorithm Steps

2.2.1 Initialization of Centroids

Randomly initialize k centroids or use techniques like k-means++ to select initial centroids that are spread out.

2.2.2 Assignment Step

Assign each data point to the nearest centroid based on the Euclidean distance.

2.2.3 Update Step

Update the centroids by calculating the mean of all data points assigned to each centroid.

2.2.4 Convergence Criteria

The algorithm converges when the centroids no longer change significantly between iterations or when a maximum number of iterations is reached.

3. Hierarchical Clustering

3.1 Understanding Hierarchical Clustering

3.1.1 Basic Concept and Intuition

Hierarchical clustering is a method of cluster analysis that seeks to build a hierarchy of clusters. It does not require specifying the number of clusters in advance and can be visualized using a dendrogram, which shows the arrangement of the clusters produced by the algorithm. The main idea is to create a tree-like structure of nested clusters.

3.1.2 Types of Hierarchical Clustering

- **Agglomerative (Bottom-Up)**: Starts with each data point as a single cluster and merges the closest pairs of clusters iteratively until all points are in a single cluster.
- **Divisive (Top-Down)**: Starts with all data points in a single cluster and splits them iteratively into smaller clusters until each data point is in its own cluster.

3.2 Algorithm Steps for Agglomerative Clustering

3.2.1 Initialization

- Begin with each data point as its own cluster.

3.2.2 Merging Clusters Based on Distance Metrics

- **Single-Link (Minimum Distance)**: Distance between two clusters is defined as the shortest distance between points in the two clusters.
- **Complete-Link (Maximum Distance)**: Distance between two clusters is defined as the longest distance between points in the two clusters.
- **Average-Link (Mean Distance)**: Distance between two clusters is defined as the average distance between all pairs of points in the two clusters.

3.2.3 Creating a Dendrogram

- A dendrogram is a tree-like diagram that records the sequences of merges or splits. It shows how clusters are nested and the distance at which clusters are merged.

4. DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

4.1 Understanding DBSCAN

4.1.1 Basic Concept and Intuition

DBSCAN is a clustering algorithm that groups together points that are closely packed together while marking as outliers points that lie alone in low-density regions. Unlike k-means, DBSCAN does not require the number of clusters to be specified in advance and can find arbitrarily shaped clusters.

4.1.2 Key Parameters

- **Epsilon (ε)**: The maximum distance between two points for one to be considered as in the neighbourhood of the other. It defines the radius of the neighbourhood around a point.
- **minPts**: The minimum number of points required to form a dense region (i.e., a cluster).

4.1.3 Identifying Core, Border, and Noise Points

- **Core Points**: Points that have at least minPts points within a distance of ε.
- **Border Points**: Points that are within ε distance of a core point but do not have enough neighbours to be a core point themselves.
- **Noise Points**: Points that are neither core points nor border points.

4.2 Algorithm Steps

4.2.1 Initializing Clusters

Start with an unvisited point and check if it forms a dense region. If yes, it is the starting point of a new cluster.

4.2.2 Expanding Clusters Based on Density

Expand the cluster by including all points within ε distance of any core point in the cluster. Repeat this process until no more points can be added to the cluster.

4.2.3 Handling Noise Points

Points that do not belong to any cluster are labelled as noise points.

5. Model Evaluation and Metrics for Clustering

5.1 Evaluation Metrics for Clustering

5.1.1 Inertia (Within-Cluster Sum of Squares)

Inertia measures the compactness of clusters. It is the sum of squared distances between each point and its closest centroid

$$\text{Inertia} = \sum_{i=1}^{n} \min_{\mu_j \in C} \|x_i - \mu_j\|^2$$

Lower inertia values indicate more compact clusters.

5.1.2 Silhouette Score

The silhouette score measures how similar a point is to its own cluster compared to other clusters. It ranges from -1 to 1, where a higher score indicates better-defined clusters

$$\bar{s(i)} = \frac{b(i)-a(i)}{\max(a(i),b(i))}$$

where a(i) is the average distance to points in the same cluster, and b(i) is the average distance to points in the nearest cluster.

5.1.3 Davies-Bouldin Index

The Davies-Bouldin index measures the average similarity ratio of each cluster with its most similar cluster. Lower values indicate better clustering

$$\mathrm{DB} = \frac{1}{k} \sum_{i=1}^{k} \max_{j \neq i} \left(\frac{\sigma_i + \sigma_j}{d(c_i, c_j)} \right)$$

where σ_i is the average distance of points in cluster i to its centroid, and $d(c_i, c_j)$ is the distance between centroids i and j.

Model Evaluation in PyTorch

5.2.1 Calculating Evaluation Metrics

```
from sklearn.metrics import silhouette_score, davies_bouldin_score

# Calculate Inertia
def calculate_inertia(X, labels):
    unique_labels = torch.unique(labels)
    inertia = 0.0
    for label in unique_labels:
        if label == -1:  # Skip noise points
            continue
        cluster_points = X[labels == label]
        centroid = cluster_points.mean(dim=0)
        inertia += torch.sum((cluster_points - centroid) ** 2).item()
    return inertia

# Calculate Silhouette Score
def calculate_silhouette_score(X, labels):
    return silhouette_score(X.numpy(), labels.numpy())

# Calculate Davies-Bouldin Index
def calculate_davies_bouldin_index(X, labels):
    return davies_bouldin_score(X.numpy(), labels.numpy())

# Calculate evaluation metrics
inertia = calculate_inertia(X_tensor, labels)
silhouette = calculate_silhouette_score(X_tensor, labels)
davies_bouldin = calculate_davies_bouldin_index(X_tensor, labels)

print(f'Inertia: {inertia:.2f}')
print(f'Silhouette Score: {silhouette:.2f}')
print(f'Davies-Bouldin Index: {davies_bouldin:.2f}')
```

5.2.2 Visualizing Clustering Results

```
# Plot the clusters
def plot_clusters(X, labels):
    unique_labels = torch.unique(labels)
    colors = plt.get_cmap("viridis", len(unique_labels))
    plt.figure(figsize=(8, 6))
    for k in unique_labels:
        if k == -1:
            color = 'k'
            marker = 'x'
            edgecolor = 'none'
        else:
            color = colors(k / len(unique_labels))
            marker = 'o'
            edgecolor = 'k'
        class_member_mask = (labels == k)
        xy = X[class_member_mask]
        plt.scatter(xy[:, 0], xy[:, 1], c=[color], marker=marker, edgecolor=edgecolor, s=50, label=f'Cluster {k}' if k != -1 else 'Noise')
    plt.title('DBSCAN Clustering')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.legend()
    plt.show()

plot_clusters(X_tensor.numpy(), labels)
```

Explanation

1. **Calculating Inertia**: The calculate_inertia function calculates the sum of squared distances between points and their cluster centroids. This metric helps evaluate the compactness of clusters.
2. **Calculating Silhouette Score**: The calculate_silhouette_score function uses the silhouette_score function from sklearn.metrics to compute the average silhouette score of all samples.
3. **Calculating Davies-Bouldin Index**: The calculate_davies_bouldin_index function uses the davies_bouldin_score function from sklearn.metrics to compute the Davies-Bouldin index.
4. **Visualizing Clustering Results**: The plot_clusters function visualizes the clustering results using scatter plots. Each cluster is represented by a different color, and noise points are marked with an 'x'.

This implementation demonstrates how to calculate and interpret key evaluation metrics for clustering and visualize the clustering results using PyTorch and Matplotlib.