Clustering

Clustering is a type of unsupervised learning that involves grouping a set of objects in such a way that objects in the same group (or cluster) are more similar to each other than to those in other groups.

k-Means Clustering

- **Concept**: k-Means is one of the simplest and most popular clustering algorithms. It partitions the dataset into `k` clusters, where each data point belongs to the cluster with the nearest mean.
- **Algorithm**:
    1. Initialize `k` cluster centroids randomly.
    2. Assign each data point to the nearest centroid.
    3. Recalculate the centroids as the mean of all points assigned to each cluster.
    4. Repeat steps 2 and 3 until convergence (i.e., centroids no longer change significantly).
- **Strengths**: Simple and fast for small to medium-sized datasets.
- **Weaknesses**: Sensitive to initial centroid positions, may converge to a local minimum, not suitable for non-globular clusters or clusters of different sizes.

Hierarchical Clustering

- **Concept**: Hierarchical clustering creates a hierarchy of clusters using either a top-down (divisive) or bottom-up (agglomerative) approach.
- **Algorithm**:
    - **Agglomerative**:
        1. Start with each data point as its own cluster.
        2. Merge the closest pair of clusters.
        3. Repeat until all points are in a single cluster.
    - **Divisive**:
        1. Start with all points in one cluster.
        2. Recursively split clusters until each point is its own cluster.
- **Strengths**: No need to specify the number of clusters in advance, produces a dendrogram for visualization.
- **Weaknesses**: Computationally intensive, especially for large datasets, sensitive to noise and outliers.

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

- **Concept**: DBSCAN groups together points that are closely packed and marks points that are isolated in low-density regions as outliers.
- **Algorithm**:
    1. Define parameters `ε` (epsilon, the maximum radius of the neighborhood) and `MinPts` (minimum number of points in the neighborhood).
    2. Classify points as core points, reachable points, or outliers.

    3. Expand clusters from core points.
- **Strengths**: Can find arbitrarily shaped clusters, robust to noise and outliers.
- **Weaknesses**: Not suitable for datasets with varying densities, sensitive to parameter selection.

Dimensionality Reduction

Dimensionality reduction involves reducing the number of random variables under consideration, making the data easier to visualize and often improving algorithm performance.

Principal Component Analysis (PCA)

- **Concept**: PCA transforms the data to a new coordinate system where the greatest variance lies on the first axis, the second greatest variance on the second axis, and so on.
- **Algorithm**:
    1. Standardize the data.
    2. Calculate the covariance matrix.
    3. Calculate the eigenvalues and eigenvectors of the covariance matrix.
    4. Sort eigenvalues and eigenvectors.
    5. Select the top `k` eigenvectors to form a new feature space.
- **Strengths**: Reduces complexity, improves computational efficiency, captures the most important variance in the data.
- **Weaknesses**: Linear method, may not capture complex structures, sensitive to outliers.

t-Distributed Stochastic Neighbor Embedding (t-SNE)

- **Concept**: t-SNE is a non-linear dimensionality reduction technique that is particularly well-suited for visualizing high-dimensional data in 2 or 3 dimensions.
- **Algorithm**:
    1. Compute pairwise similarities between points in the high-dimensional space.
    2. Define a probability distribution over pairs of points.
    3. Define a similar distribution in the lower-dimensional space.
    4. Minimize the Kullback-Leibler divergence between the two distributions.
- **Strengths**: Excellent for visualization, preserves local structure.
- **Weaknesses**: Computationally intensive, difficult to interpret, sensitive to hyperparameters.

Linear Discriminant Analysis (LDA)

- **Concept**: LDA is a supervised dimensionality reduction technique that aims to find a linear combination of features that best separate two or more classes.
- **Algorithm**:
    1. Compute the within-class and between-class scatter matrices.
    2. Compute the eigenvalues and eigenvectors of the scatter matrices.
    3. Select the top eigenvectors to form a new feature space.
- **Strengths**: Effective for classification tasks, maximizes class separability.

- **Weaknesses**: Assumes normal distribution and equal covariance, limited to linear boundaries.

Semi-Supervised Learning

Semi-supervised learning combines a small amount of labeled data with a large amount of unlabeled data during training.

Methods Combining Labeled and Unlabeled Data

- **Self-Training**: Use a model trained on labeled data to label the unlabeled data, and then retrain the model on the combined data.
- **Co-Training**: Train two models on different views of the data and use each to label the unlabeled data for the other.
- **Graph-Based Methods**: Represent data as a graph and use label propagation algorithms to spread labels through the graph.
- **Generative Models**: Use generative models like variational autoencoders (VAEs) to learn the underlying distribution of the data and generate pseudo-labels.

Reinforcement Learning

Reinforcement learning (RL) involves an agent that learns to make decisions by taking actions in an environment to maximize cumulative reward.

Q-Learning

- **Concept**: Q-Learning is a model-free RL algorithm that learns the value of actions in a given state to find an optimal policy.
- **Algorithm**:
    1. Initialize Q-values arbitrarily.
    2. For each episode:
        - For each step in the episode:
            - Choose an action using an exploration-exploitation strategy.
            - Take the action and observe the reward and next state.
            - Update the Q-value using the Bellman equation.
- **Strengths**: Simple, effective for discrete action spaces.
- **Weaknesses**: Inefficient for large state spaces, slow convergence.

Deep Q-Networks (DQNs)

- **Concept**: DQNs combine Q-Learning with deep neural networks to handle high-dimensional state spaces.
- **Algorithm**:
    1. Use a neural network to approximate the Q-function.
    2. Use experience replay to store and sample past experiences.
    3. Use target networks to stabilize training.

    4. Update the Q-network using the Q-learning update rule.
- **Strengths**: Handles high-dimensional inputs like images.
- **Weaknesses**: Requires large amounts of data and computational resources, sensitive to hyperparameters.

Policy Gradient Methods

- **Concept**: Policy gradient methods optimize the policy directly by maximizing the expected reward.
- **Algorithm**:
    1. Parameterize the policy with a neural network.
    2. Collect trajectories by running the policy in the environment.
    3. Compute the gradient of the expected reward with respect to the policy parameters.
    4. Update the policy parameters using gradient ascent.
- **Strengths**: Effective for continuous action spaces, can learn stochastic policies.
- **Weaknesses**: High variance in gradient estimates, sample inefficient.

Multi-Armed Bandits

- **Concept**: Multi-armed bandits are a simpler form of RL where an agent repeatedly chooses from a set of actions (arms) to maximize cumulative reward.
- **Algorithm**:
    1. **Epsilon-Greedy**: Choose a random action with probability ε, otherwise choose the best-known action.
    2. **UCB (Upper Confidence Bound)**: Select actions based on a balance of exploitation and exploration.
    3. **Thompson Sampling**: Use probability matching to sample actions according to their likelihood of being optimal.
- **Strengths**: Simple, effective for online learning scenarios.
- **Weaknesses**: Limited to static environments, doesn't handle delayed rewards well.

Markov Decision Processes (MDPs)

- **Concept**: MDPs provide a mathematical framework for modeling decision-making with states, actions, rewards, and state transitions.
- **Algorithm**:
    1. Define the state space, action space, reward function, and transition probabilities.
    2. Solve for the optimal policy using dynamic programming methods like Value Iteration or Policy Iteration.
- **Strengths**: Provides a solid theoretical foundation, can handle stochastic environments.
- **Weaknesses**: Computationally expensive for large state/action spaces, assumes full knowledge of the environment.