9 Artificial Intelligence I

Hui Xu, xuh@fudan.edu.cn

Learning Objective:

- Understand the principles and techniques of clustering, including K-Means and DBSCAN algorithms.
- Understand the concept and methodology of regression, with a focus on linear regression.
- Understand the fundamentals of classification, particularly using logistic regression and KNN.

9.1 Overview

Artificial Intelligence (AI) is a branch of computer science that enables machines to perform tasks requiring human intelligence, such as reasoning, learning, problem-solving, perception, and language understanding. AI can be classified by capability or by technique. By capability, there is weak (narrow) AI, which specializes in a single task (*e.g.,* face recognition, recommendation systems); strong (general) AI, which can perform any human intellectual task; and superintelligent AI, a theoretical AI surpassing human intelligence. By technique, AI includes symbolic AI, which uses rules and logic to represent knowledge; expert systems, practical applications of symbolic AI with domain-specific knowledge; and machine learning, which learns patterns from data to make predictions without explicit programming. This lecture focuses on machine learning.

Machine Learning Machine learning is a branch of artificial intelligence that enables systems to learn patterns from data and make predictions or decisions without being explicitly programmed. It can be broadly categorized into supervised learning and unsupervised learning. Supervised learning involves learning from labeled data, where the correct outputs are known, while unsupervised learning deals with discovering hidden patterns or structures in unlabeled data.

The common process in machine learning typically involves several steps:

1) *Data Collection:* Gather labeled or representative data that captures the problem you are trying to solve.
2) *Data Preprocessing:* Handle missing data, scale features, encode categorical variables, and perform other transformations to prepare the dataset for training.
3) *Model Selection:* Choose an appropriate model for the specific task. Examples include linear regression, neural networks, *etc.*
4) *Training the Model:* Fit the model to the training data by minimizing an error metric, such as mean squared error for regression or cross-entropy loss for classification.
5) *Model Evaluation:* Assess the performance of the trained model on a separate test set using metrics like accuracy for classification or mean squared error for regression.

Next, we will introduce three typical tasks in machine learning. These include *clustering*, which is an unsupervised learning task, and *regression* and *classification*, which are supervised learning tasks.

9.2 Clustering

Clustering is an unsupervised machine learning task that groups similar data points together based on their features. The goal is to discover the inherent structure or patterns in the dataset.

9.2.1 K-Means

K-Means is a popular clustering algorithm that partitions data into K clusters based on similarity. Each cluster is represented by its *centroid*, which is the mean of all points in the cluster. The algorithm iteratively updates cluster assignments and centroids to minimize the variance within clusters. K-Means can be understood as an instance of the *Expectation-Maximization (EM)* approach.

The algorithm proceeds as follows:

1) *Initialize the centroids:* Randomly select K data points as the initial centroids.

2) *Expectation (E) step:* Assign each data point to the closest centroid.

3) *Maximization (M) step:* Recompute each centroid as the mean of all points assigned to it.

4) *Repeat:* Continue steps 2) and 3) until the centroids do not change or a threshold is reached.

In practice, clustering algorithms such as K-Means rely on a measure of similarity or distance between data points. The distance can be calculated using various norms, including:

- *L1 norm (Manhattan distance):* The sum of absolute differences between the coordinates of two points. For points $\mathbf{x} = (x_1, \ldots, x_d)$ and $\mathbf{y} = (y_1, \ldots, y_d)$, it is defined as

$$\|\mathbf{x} - \mathbf{y}\|_1 = \sum_{i=1}^{d} |x_i - y_i|.$$

- *L2 norm (Euclidean distance):* The square root of the sum of squared differences, representing the straight-line distance between two points:

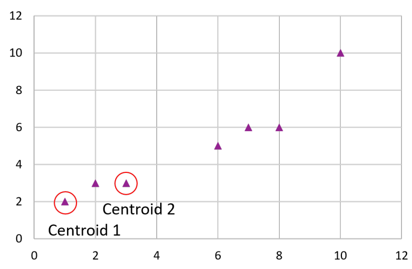
$$\|\mathbf{x} - \mathbf{y}\|_2 = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}.$$

- *Lp norm:* A generalized distance measure, where $p \geq 1$:

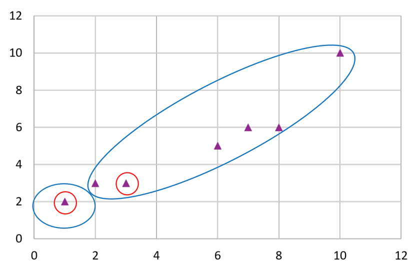
$$\|\mathbf{x} - \mathbf{y}\|_p = \left(\sum_{i=1}^{d} |x_i - y_i|^p \right)^{1/p}.$$

Figure 9.1 demonstrates an example of 2-Means clustering using Euclidean distance. In Figure 9.1a, two points, $(1, 2)$ and $(3, 3)$, are randomly chosen as the initial centroids. Next, Figure 9.1b shows the assignment of each point to a cluster based on the Euclidean distance to the two centroids. Figure 9.1c recomputes the centroids. Centroid 1 remains unchanged because no points are assigned to its cluster, while Centroid 2 is updated to $(6, 5.5)$. In Figure 9.1d, each point is reassigned based on the new centroids. As a result, two more points are assigned to cluster 1. Figure 9.1e shows the recomputed centroids after this reassignment, (2, 8/3) and (31/4, 27/4). Since the cluster assignments no longer change, the algorithm converges and the clustering is complete.

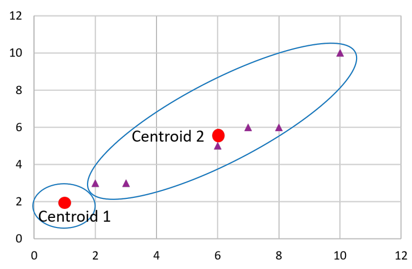
While K-Means is simple and widely used, it has several limitations. The number of clusters K must be known in advance. The algorithm is sensitive to outliers, and poor initial centroids can lead to bad clusters. Additionally, when using Euclidean distance, the clusters tend to be roughly circular in shape. Next, we introduce a more robust clustering approach, *DBSCAN*.



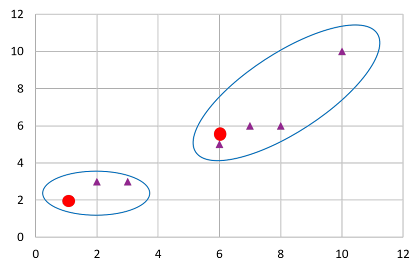
(a) Init two centroids.



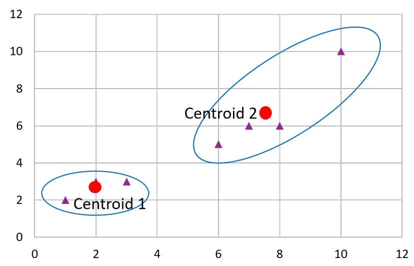
(b) Assign points to clusters.



(c) Update the centroids.



(d) Assign points to clusters.



(e) Update the centroids.

Figure 9.1: Demonstration K-Means with two classes.

9.2.2 DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a clustering algorithm that groups together points that are closely packed and marks points that lie alone in low-density regions as noise. The algorithm works as follows:

1) Pick an unvisited point.

2) Find all points within ε (its neighborhood).

3) If it has $\geq minPts$ neighbors, start a new cluster.

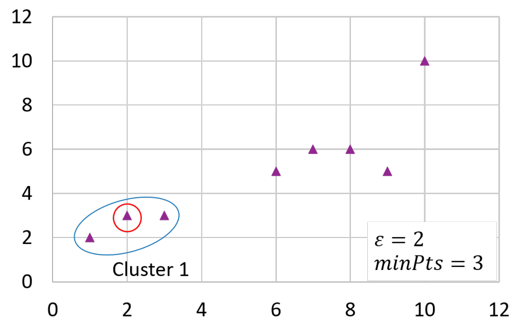
4) Expand the cluster by visiting all its neighbors recursively.

5) If it has $< minPts$, mark it as *noise*.

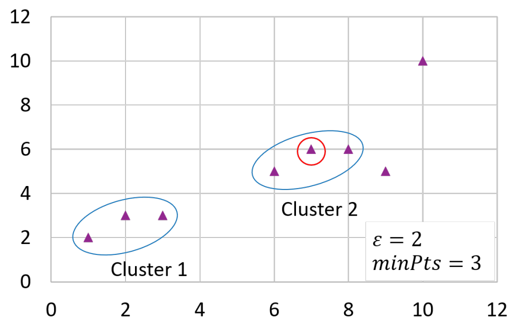
6) Repeat until all points are visited.

Within each cluster, there are two types of points. *Core points* have at least *minPts* points (including itself) within a distance ε. *Border points* lie within the neighborhood of a core point but do not themselves have enough neighbors to be core points.

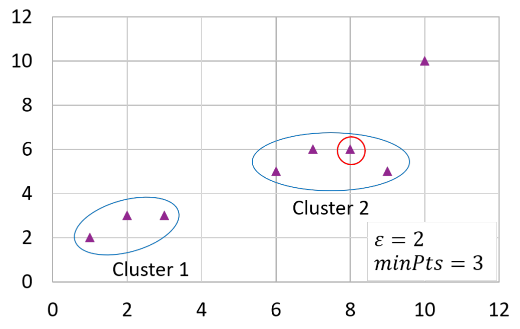
Figure 9.2 demonstrates an example with $\varepsilon = 2$ and $minPts = 3$. The algorithm starts from a randomly selected point $(2, 3)$ in Figure 9.2a. There are two points within distance 2. Since the number of neighbors meets the threshold, Cluster 1 is formed. Next, the algorithm starts a new round with another unvisited point. As shown in Figure 9.2b, two points are within its neighborhood. Figure 9.2c further demonstrates that, from one of these neighbors, another point can be reached. As a result, Cluster 2 is formed with four points. Finally, one unvisited point remains in Figure 9.2d. Since it has no neighbors within distance ε, it is marked as *noise*. The process is then complete.



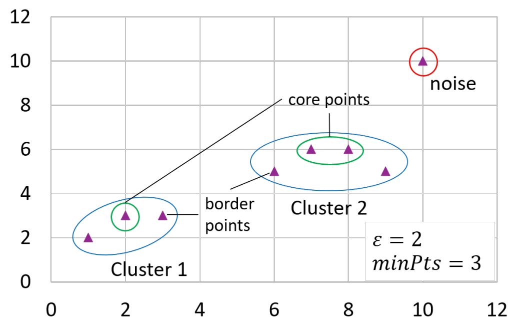
(a) First round.



(b) Second round: step 1.



(c) Second round: step 2.



(d) Third round.

Figure 9.2: Demonstration of DBSCAN.

9.3 Regression

A *regression task* concerns predicting a continuous-valued output based on one or more input features. Figure 9.3 demonstrates an example. Given a set of data points where the horizontal axis represents the input and the vertical axis represents the output, the goal is to learn a function that predicts the output value for a new input, such as $x = 12$.

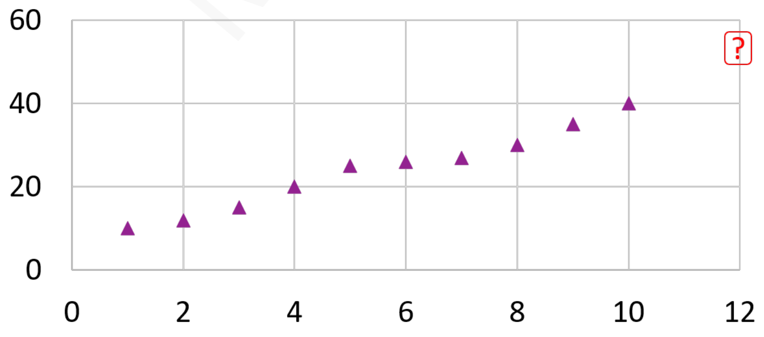


Figure 9.3: Demonstration of a regression task.

9.3.1 Regression Models

Linear Regression

Linear regression models the relationship between input features x and a continuous target variable y using a linear function:

$$\hat{y} = xw + b$$

where w represents the model coefficients and b is the bias term.

The model parameters are typically learned by minimizing the *sum of squared errors* (SSE):

$$L(w,b) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \|(Xw + b) - Y\|^2$$

The goal is to find the parameters that minimize this loss:

$$(w^*, b^*) = \arg\min_{w,b} L(w,b)$$

Polynomial Regression

Linear regression cannot fit curved or nonlinear data effectively. Polynomial regression extends linear regression by adding nonlinear terms of the input variable:

$$\hat{y} = w_0 + w_1 x + w_2 x^2 + \cdots + w_d x^d$$

More

Polynomial regression captures smooth nonlinear relationships but struggles with periodic data (*e.g.,* sine waves). Even high-degree polynomials tend to oscillate and cannot model repeating patterns effectively. For periodic or oscillatory patterns, alternative models are more suitable, such as *Fourier series expansion* ($\hat{y} = a_0 + \sum_{k=1}^{K} \big(a_k \cos(kx) + b_k \sin(kx)\big)$) or other kernel methods.

9.3.2 Optimization

This section introduces two approaches for optimizing model parameters: *least squares* and *gradient descent.*

Least Squares

The least squares method is applicable when the model is linear in parameters and the dataset is not too large. For simplicity, let us use linear regression as an example and assume that the input matrix X has been augmented with a column of ones so that the bias term b is included in w. Then the model becomes:

$$\hat{Y} = Xw$$

and the loss function simplifies to:

$$L(w) = \|Xw - Y\|^2$$

Expanding the quadratic form:

$$\begin{aligned} L(w) &= (Xw - Y)^T (Xw - Y) \\ &= w^T X^T X w - 2Y^T X w + Y^T Y \end{aligned}$$

Taking the derivative of $L(w)$ with respect to w:

$$\nabla_w L(w) = 2X^T (Xw - Y)$$

Setting the derivative to zero for minimization:

$$X^T (Xw - Y) = 0$$

Solving for w gives the *normal equation*:

$$w^* = (X^T X)^{-1} X^T Y$$

This yields a closed-form solution for the optimal parameters, provided $X^T X$ is invertible (*i.e.,* X has full column rank). Although the polynomial regression model is nonlinear in x, it is still linear in parameters w_i. Hence, it can also be solved by the least squares approach, after expanding the input features to include polynomial terms.

Gradient Descent

Gradient descent is a commonly-used optimization approach in practice because it is applicable to a wide range of models, including linear, polynomial, and nonlinear regression. Instead of finding a global optimum, it is a greedy approach that iteratively updates parameters in the direction of the steepest descent of the loss function, choosing the update that locally reduces the loss the most.

The parameters are updated iteratively according to:

$$w^{t+1} := w^t - \eta \nabla_w L(w)$$

where η is the learning rate and $\nabla_w L(w)$ is the gradient of the loss with respect to w.

Consider a linear regression model for d features:

$$\hat{y} = w_1 x_1 + w_2 x_2 + \cdots + w_d x_d + b$$

The gradient of the loss $L(w)$ with respect to the weight vector $w = (w_1, w_2, \ldots, w_d)$ is the vector of partial derivatives:

$$\nabla_w L(w) = \left(\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \ldots, \frac{\partial L}{\partial w_d} \right)$$

For one weight w_i, using the chain rule:

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_i}$$

Since $\hat{y} = \sum_{j=1}^{d} w_j x_j + b$, we have

$$\frac{\partial \hat{y}}{\partial w_i} = x_i$$

If we use the squared error loss $L = (\hat{y} - y)^2$, then

$$\frac{\partial L}{\partial \hat{y}} = \frac{\partial (\hat{y} - y)^2}{\partial \hat{y}} = 2(\hat{y} - y)$$

Combining these results:

$$\frac{\partial L}{\partial w_i} = 2(\hat{y} - y)\, x_i$$

Similarly, the gradient with respect to all weights is

$$\nabla_w L(w) = 2(\hat{y} - y)\, x$$

where $x = (x_1, x_2, \ldots, x_d)^T$ is the input feature vector.

9.4 Classification

Classification is another supervised learning task that involves predicting a discrete label or category for a given input. Depending on the number of possible labels, classification can be classified as *binary classification*, where there are two possible classes (*e.g.*, spam or not spam), or *multiclass classification*, where there are more than two possible classes (*e.g.*, animals). Figure 9.4 illustrates a binary classification problem with two classes, A and B.

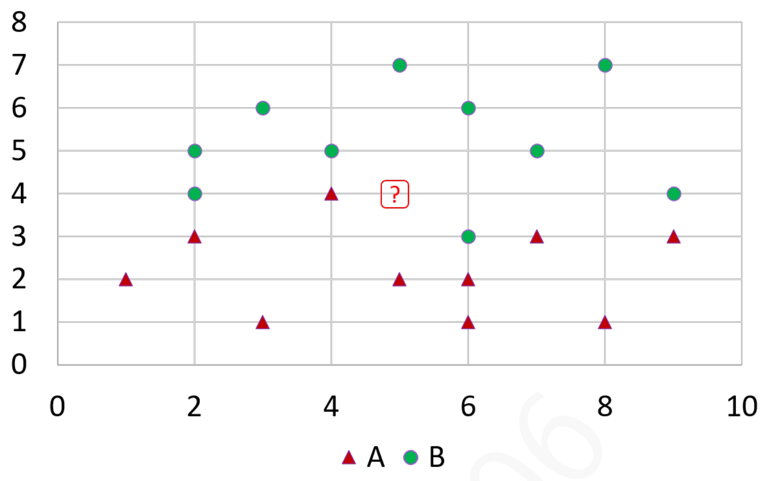


Figure 9.4: Demonstration of a classification task.

9.4.1 Logistic Regression

Logistic regression is a binary classifier built upon a regression model, such as linear regression. Instead of directly predicting a continuous value, it models the probability that the label y belongs to class 1. Given an input vector x, the model first computes a linear combination:

$$\hat{y} = xw + b,$$

where $\hat{y}$ is the *logit*, representing the unbounded linear score. This score is then passed through the *sigmoid* (logistic) function to obtain a probability:

$$P(y = 1 \mid x) = \sigma(\hat{y}) = \frac{1}{1 + e^{-\hat{y}}}.$$

Finally, a decision is made based on a threshold (typically 0.5):

$$\tilde{y} = \begin{cases} 1, & \text{if } P(y = 1 \mid x) > 0.5, \\ 0, & \text{otherwise.} \end{cases}$$

In summary, logistic regression converts a regression score into a probability and uses it to perform binary classification.

9.4.2 KNN

The *k-Nearest Neighbors* (KNN) algorithm is a simple yet powerful non-parametric method for classification, including both binary and multiclass cases. Given a new input sample, KNN finds the k training samples that are closest to it in feature space (typically measured by Euclidean distance), and predicts the label by majority voting among these neighbors.

$$\hat{y} = \text{mode}\Big(\{y_i \mid x_i \in \mathcal{N}_k(x)\}\Big)$$

where $\mathcal{N}_k(x)$ denotes the k nearest neighbors of x, and $\text{mode}(\cdot)$ returns the most frequent label among them.

KNN makes no explicit assumption about the underlying data distribution, but its performance depends strongly on the choice of k and the distance metric. Smaller k may lead to overfitting, while larger k tends to smooth the decision boundary.