8 Artificial Intelligence I

Hui Xu, xuh@fudan.edu.cn

Learning Objective:

- Learn the principles of clustering and common algorithms, including K-Means and DBSCAN.
- Understand the principles of regression, with a focus on linear regression.
- Grasp the fundamentals of classification, including logistic regression and KNN.

8.1 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 specific tasks (*e.g.,* face recognition and recommendation systems); strong (general) AI, which aims to perform a wide range of human intellectual tasks (with large language models (LLMs) representing a step toward this goal); and superintelligent AI, a theoretical AI surpassing human intelligence. By technique, AI includes symbolic AI, which uses rules and logic to represent knowledge and solve problems (*e.g.,* expert systems); and machine learning, which learns patterns from data to make predictions without being explicitly programmed for each task. This lecture focuses on machine learning.

MF 20006

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.

8.2 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.

8.2.1 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 sum of distances between data points and their assigned centroids. K-Means follows an iterative procedure similar to the *Expectation-Maximization (EM)* approach.

The algorithm proceeds as follows:

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

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

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

4) *Repeat:* Continue steps 2) and 3) until the centroids converge or a maximum number of iterations is reached.

In practice, clustering algorithms such as K-Means rely on a measure of 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:

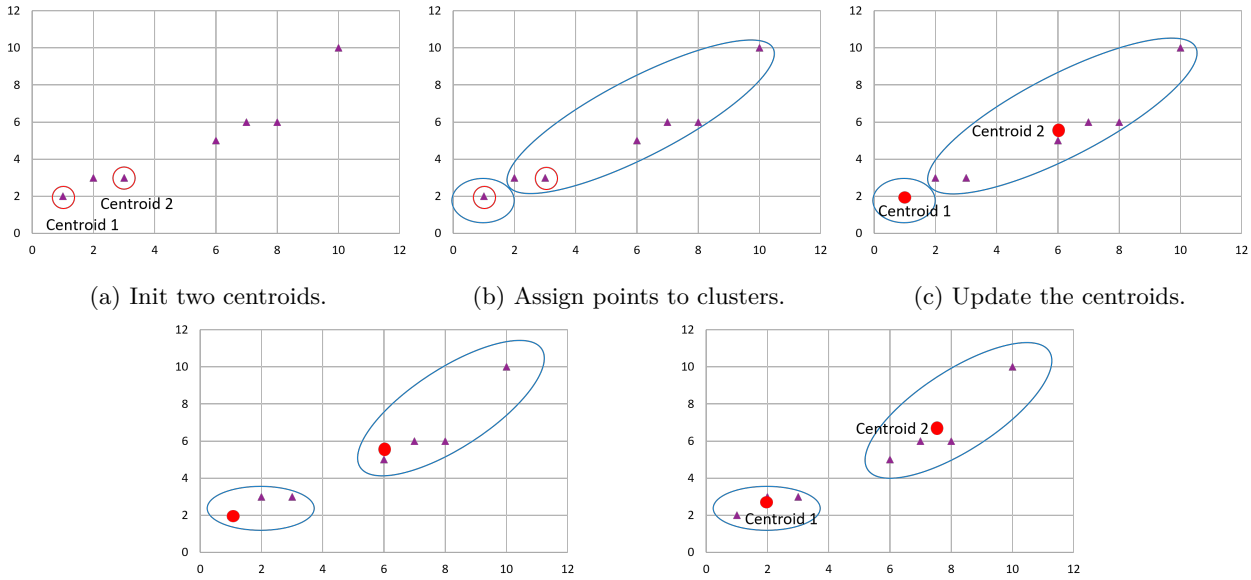
$$\|\mathbf{x} - \mathbf{y}\|_2 = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}.$$

- *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 8.1 illustrates an example of K-Means clustering with two clusters using Euclidean distance. In Figure 8.1a, two points, $(1, 2)$ and $(3, 3)$, are randomly selected as the initial centroids. Figure 8.1b shows how each point is assigned to the nearest centroid. Figure 8.1c shows the updated centroids. Centroid 1 remains unchanged, while Centroid 2 moves to $(6, 5.5)$. In Figure 8.1d, the points are reassigned based on

MIT 2006



(a) Init two centroids. (b) Assign points to clusters. (c) Update the centroids.

(d) Assign points to clusters. (e) Update the centroids.

Figure 8.1: Demonstration K-Means with two classes.

the updated centroids, resulting in two additional points being assigned to Cluster 1. Figure 8.1e shows the new centroids, $(2, 8/3)$ and $(31/4, 27/4)$. Since the cluster assignments no longer change, the algorithm converges.

While K-Means is simple and widely used, it has several limitations. The number of clusters K must be specified in advance. The algorithm is sensitive to outliers, and different initial centroids may lead to different clustering results. Additionally, K-Means works best when clusters are compact and roughly spherical in shape.

Next, we introduce a density-based clustering algorithm, *DBSCAN*, which can identify clusters of arbitrary shapes and handle noise.

8.2.2 DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a clustering algorithm that groups points in dense regions and identifies isolated points in low-density regions as noise. Unlike K-Means, DBSCAN does not require specifying the number of clusters in advance.

The algorithm proceeds as follows:

1) *Select a point:* Pick an unvisited point.

2) *Find neighbors:* Find all points within distance ε, including the point itself.

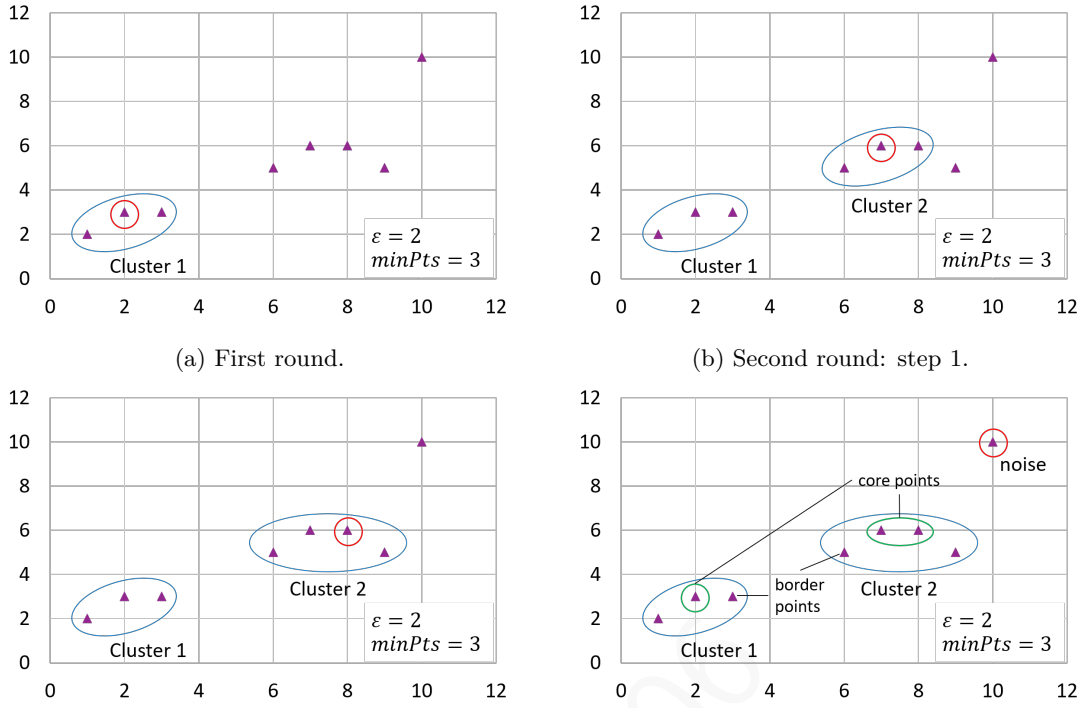
3) *Check density:* If the neighborhood contains at least *minPts* points, start a new cluster. Otherwise, mark the point as noise.

4) *Expand the cluster:* Add neighboring points to the cluster and recursively expand from those that are core points.

5) *Repeat:* Continue until all points have been visited.

DBSCAN distinguishes between two types of points within a cluster. *Core points* have at least *minPts* points (including themselves) within distance ε. *Border points* lie within the neighborhood of a core point

but do not have enough neighbors to be core points. Points that do not belong to any cluster are classified as *noise*.



(a) First round.

(b) Second round: step 1.

(c) Second round: step 2.

(d) Third round.

Figure 8.2: Demonstration of DBSCAN.

Figure 8.2 illustrates an example of DBSCAN with $\varepsilon = 2$ and $minPts = 3$. The algorithm starts from a randomly selected point $(2, 3)$ in Figure 8.2a. Since two other points lie within distance 2, the selected point is a core point, and Cluster 1 is formed. Next, the algorithm selects another unvisited point. As shown in Figure 8.2b, two other points lie within its neighborhood, allowing a new cluster to be formed. Figure 8.2c shows that another point can be reached through one of its neighboring core points. As a result, Cluster 2 contains four points. Finally, one unvisited point remains in Figure 8.2d. Since it has no other points within distance ε and is not reachable from any core point, it is classified as *noise*. The clustering process is then complete.

Compared with K-Means, DBSCAN can discover clusters of arbitrary shapes and identify noise. However, its performance depends on the choice of ε and $minPts$, and it may struggle with clusters of varying densities.

8.3 Regression

A *regression task* aims to predict a continuous-valued output based on one or more input features. Figure 8.3 illustrates 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 for a new input, such as $x = 12$.

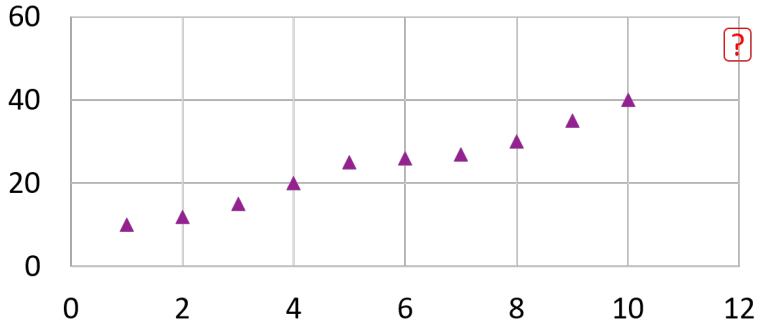


Figure 8.3: Demonstration of a regression task.

8.3.1 Regression Models

Linear Regression

Linear regression models the relationship between input features and a continuous target variable using a function that is linear in its parameters:

$$\hat{y} = \mathbf{x}^T \mathbf{w} + b,$$

where $\mathbf{x}$ represents the input features, $\mathbf{w}$ represents the model coefficients, and b is the bias term.

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

$$L(\mathbf{w}, b) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \|X\mathbf{w} + b\mathbf{1} - Y\|_2^2,$$

where X is the input feature matrix, Y is the vector of target values, and $\mathbf{1}$ is a vector of ones.

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

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

Polynomial Regression

A linear function of the original input features may not capture nonlinear relationships effectively. Polynomial regression addresses this limitation by introducing polynomial features:

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

Regression for Periodic Data

Polynomial regression can capture smooth nonlinear relationships but is generally unsuitable for modeling repeating patterns, such as sine waves, over extended intervals. For periodic data, alternative

MF2006

models can be used, such as *Fourier series expansion*:

$$\hat{y} = a_0 + \sum_{k=1}^{K} \left(a_k \cos(kx) + b_k \sin(kx) \right).$$

Other approaches, such as kernel-based regression, can also model nonlinear and periodic patterns using appropriate kernel functions.

8.3.2 Parameter Estimation

This section introduces two approaches to estimating regression model parameters: an analytical solution using the *normal equation* and an iterative approach using *gradient descent*.

Normal Equation (Least Squares)

The least squares method estimates model parameters by minimizing the sum of squared errors. For models that are linear in their parameters, the optimal parameters can be obtained analytically.

For simplicity, consider linear regression. Assume that the input matrix X has been augmented with a column of ones so that the bias term b is included in the parameter vector w. The model becomes:

$$\hat{Y} = Xw$$

and the loss function simplifies to:

$$L(w) = \|Xw - Y\|_2^2$$

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, we use the following matrix differentiation rules:

$$\frac{\partial (w^T A w)}{\partial w} = (A + A^T) w, \qquad \frac{\partial (a^T w)}{\partial w} = a,$$

where A is a constant square matrix and a is a constant vector, both independent of w. If A is symmetric, the first derivative simplifies to $2Aw$.

Since $X^T X$ is symmetric, we have

$$\frac{\partial (w^T X^T X w)}{\partial w} = 2X^T X w.$$

Similarly,

$$\frac{\partial (-2Y^T X w)}{\partial w} = -2X^T Y, \qquad \frac{\partial (Y^T Y)}{\partial w} = 0.$$

Combining these results gives the gradient:

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

To minimize the loss, we set the gradient to zero:

$$2X^T (Xw - Y) = 0.$$

Rearranging the equation:

$$X^T X w = X^T Y.$$

This is known as the *normal equation*. If X^TX is invertible, multiplying both sides by $(X^TX)^{-1}$ gives:

$$w^* = (X^TX)^{-1}X^TY.$$

This provides the optimal parameters that minimize the sum of squared errors. The same approach can also be applied to polynomial regression. Although polynomial regression models nonlinear relationships between inputs and outputs, its parameters can still be estimated using the normal equation after expanding the input features to include polynomial terms.

Example. Consider a linear regression problem with three training samples:

$$(1, 2), \quad (2, 3), \quad (3, 4).$$

We aim to fit a linear model $\hat{y} = wx + b$. By incorporating the bias term into the parameter vector, we obtain:

$$X = \begin{pmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \end{pmatrix}, \quad Y = \begin{pmatrix} 2 \\ 3 \\ 4 \end{pmatrix}, \quad w = \begin{pmatrix} w_1 \\ b \end{pmatrix}.$$

Using the normal equation:

$$X^TX = \begin{pmatrix} 14 & 6 \\ 6 & 3 \end{pmatrix}, \qquad X^TY = \begin{pmatrix} 20 \\ 9 \end{pmatrix}.$$

Thus, we solve:

$$\begin{pmatrix} 14 & 6 \\ 6 & 3 \end{pmatrix} \begin{pmatrix} w_1 \\ b \end{pmatrix} = \begin{pmatrix} 20 \\ 9 \end{pmatrix}.$$

This gives:

$$w_1 = 1, \qquad b = 1.$$

Therefore, the fitted regression model is:

$$\hat{y} = x + 1.$$

MF 2006

Gradient Descent

Gradient descent is a widely used iterative optimization method that updates model parameters in the direction of the negative gradient of the loss function. Unlike the normal equation, it does not require a closed-form solution and can be applied to a wide range of models.

The parameters are updated iteratively according to:

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

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

To illustrate how the gradient is computed, consider a linear regression model with d input features. For simplicity, we represent the bias term b separately:

$$\hat{y} = w_1x_1 + w_2x_2 + \cdots + w_dx_d + b$$

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

$$\nabla_w L(w) = \begin{pmatrix} \frac{\partial L}{\partial w_1} \\ \frac{\partial L}{\partial w_2} \\ \vdots \\ \frac{\partial L}{\partial w_d} \end{pmatrix}$$

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:

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

For a single training sample, consider the squared error loss:

$$L(w, b) = (\hat{y} - y)^2$$

The derivative of the loss with respect to the prediction is:

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

Combining these results:

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

Similarly, the gradient with respect to all weights is:

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

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

The derivative with respect to the bias term is:

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

Gradient descent repeatedly updates the model parameters using these gradients until the loss converges or a stopping criterion is reached.

Example. Consider three training samples:

$$(1, 2), \quad (2, 3), \quad (3, 4).$$

We aim to fit the linear regression model:

$$\hat{y} = wx + b.$$

The batch gradients with respect to w and b are:

$$\frac{\partial L}{\partial w} = 2 \sum_{i=1}^{3} (wx_i + b - y_i)x_i,$$

$$\frac{\partial L}{\partial b} = 2 \sum_{i=1}^{3} (wx_i + b - y_i).$$

Suppose the initial parameters are $w^0 = 0$ and $b^0 = 0$, and the learning rate is $\eta = 0.01$.

MT 2000

First iteration. The initial predictions are all zero, so the gradients are:

$$\begin{aligned}\frac{\partial L}{\partial w} &= 2[(-2)\cdot 1 + (-3)\cdot 2 + (-4)\cdot 3] \\ &= -40, \\ \frac{\partial L}{\partial b} &= 2(-2-3-4) = -18.\end{aligned}$$

Applying the first gradient descent update:

$$\begin{aligned}w^1 &= 0 - 0.01(-40) = 0.4, \\ b^1 &= 0 - 0.01(-18) = 0.18.\end{aligned}$$

The updated model is:

$$\hat{y} = 0.4x + 0.18.$$

Second iteration. Using the updated parameters, the predictions are:

$$\hat{y}_1 = 0.58, \quad \hat{y}_2 = 0.98, \quad \hat{y}_3 = 1.38.$$

Recomputing the gradients:

$$\begin{aligned}\frac{\partial L}{\partial w} &= 2[(-1.42)\cdot 1 + (-2.02)\cdot 2 + (-2.62)\cdot 3] \\ &= -26.64, \\ \frac{\partial L}{\partial b} &= 2(-1.42-2.02-2.62) = -12.12.\end{aligned}$$

Applying the second gradient descent update:

$$\begin{aligned}w^2 &= 0.4 - 0.01(-26.64) = 0.6664, \\ b^2 &= 0.18 - 0.01(-12.12) = 0.3012.\end{aligned}$$

The updated model becomes:

$$\hat{y} = 0.6664x + 0.3012.$$

The process continues by recomputing the gradients and updating the parameters until convergence. With a suitable learning rate, the parameters gradually approach the optimal values $w = 1$ and $b = 1$, yielding the same solution as the normal equation.

MF2006

8.4 Classification

Classification is a supervised learning task that involves predicting a discrete label or category for a given input. Classification tasks can be divided into *binary classification*, where there are two possible classes (*e.g.*, spam or not spam), and *multiclass classification*, where there are more than two possible classes (*e.g.*, recognizing different animal species). Figure 8.4 illustrates a binary classification problem with two classes, A and B.

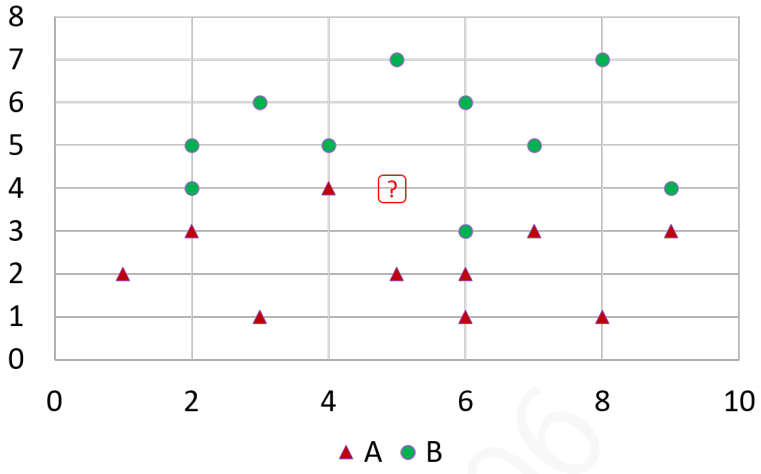


Figure 8.4: Demonstration of a classification task.

8.4.1 Logistic Regression

Logistic regression is a widely used model for binary classification. Similar to linear regression, it first computes a linear combination of the input features:

$$z = x^T w + b,$$

where x is the input feature vector, w is the weight vector, b is the bias term, and z is the *logit*, an unbounded linear score.

The logit is then passed through the *sigmoid* (logistic) function to obtain the probability of belonging to class 1:

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

Finally, the predicted class is determined using a threshold (typically 0.5):

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

Although classification can be performed directly using the logit, the sigmoid function provides class probabilities that are useful for assessing prediction confidence and computing the cross-entropy loss during training.

8.4.2 K-Nearest Neighbors (KNN)

The *k-Nearest Neighbors* (KNN) algorithm is a simple non-parametric method for classification, supporting both binary and multiclass cases. Given a new input sample, KNN finds the k closest training

samples in feature space, typically using Euclidean distance, and predicts the class by majority voting among these neighbors.

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

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

Unlike logistic regression, KNN does not learn an explicit prediction function during training. Instead, it stores the training data and performs distance calculations when making predictions.

KNN makes no explicit assumption about the underlying data distribution, but its performance depends strongly on the choice of k and the distance metric. A small k may lead to overfitting, while a large k may lead to underfitting. Since KNN relies on distances, feature scaling is often important. Additionally, prediction can be computationally expensive for large datasets.

MF 20006