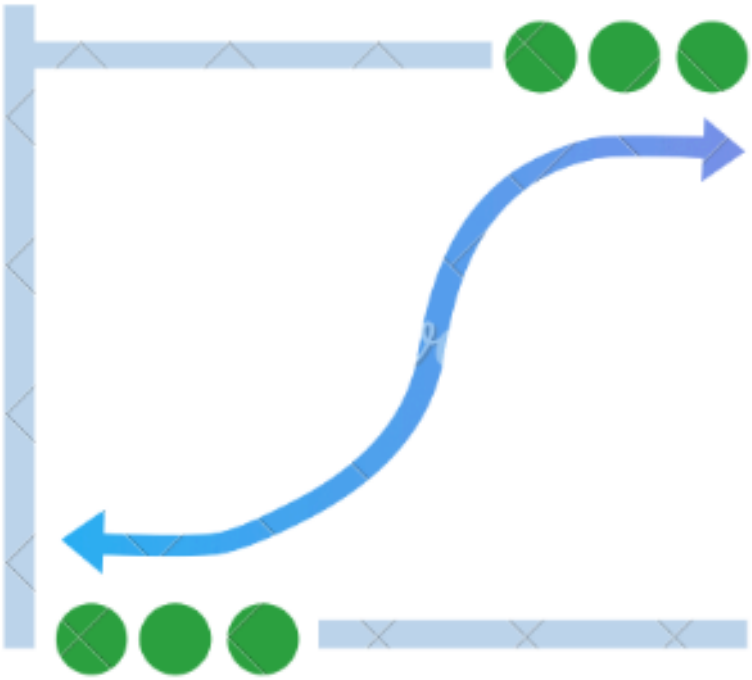
Course3_Module1

Logistic Regression

What is Logistic Regression



Logistic Regression is a statistical model used for binary classification problems. Despite its name, it's a classification algorithm rather than a regression algorithm. It predicts the probability that an instance belongs to a particular class.

Mathematically, logistic regression estimates the probability of a binary outcome based on one or more predictor variables. The model can be written as:

$$P(Y=1|X) = \frac{e^{\beta_0+\beta_1X_1+\beta_2X_2+...+\beta_nX_n}}{1+e^{\beta_0+\beta_1X_1+\beta_2X_2+...+\beta_nX_n}}$$

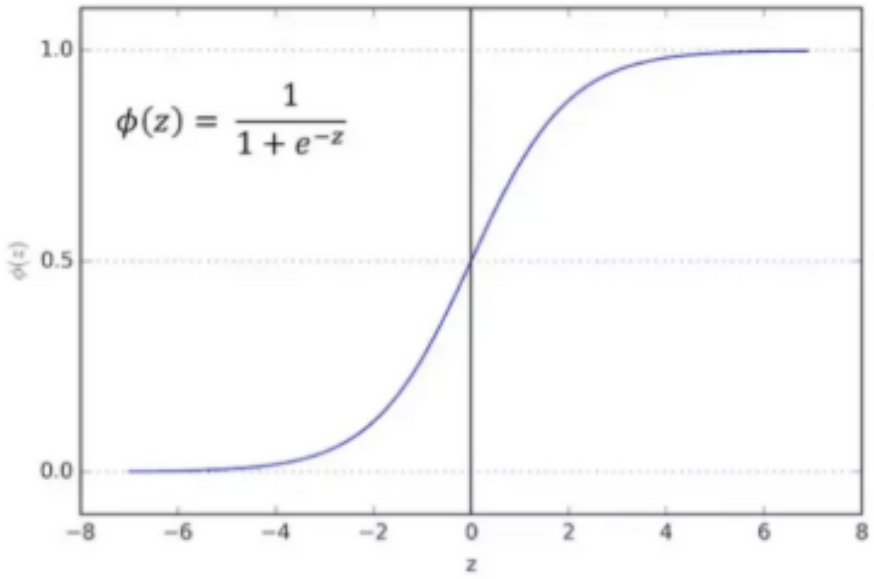
This can be simplified to:

$$P(Y=1|X) = \frac{1}{1+e^{-(\beta_0+\beta_1X_1+\beta_2X_2+...+\beta_nX_n)}}$$

Where:

- Y is the binary outcome (0 or 1)
- X represents the input features
- β represents the model parameters (coefficients)

Sigmoid Function



The sigmoid function (also known as the logistic function) is the core of logistic regression. It maps any real-valued number to a value between 0 and 1, which can be interpreted as a probability.

The sigmoid function is defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Where $z = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_n X_n$, which is the linear combination of the input features.

Key properties of the sigmoid function:

- Output is always between 0 and 1
- S-shaped curve (sigmoidal)
- Centered at z = 0, where $\sigma(0) = 0.5$
- As z approaches positive infinity, $\sigma(z)$ approaches 1

- As z approaches negative infinity, $\sigma(z)$ approaches 0

Why not use Linear Regression instead of Logistic Regression?

While linear regression might seem like a potential solution for classification problems, it has several limitations:

1. Unbounded outputs

Linear regression produces outputs that can range from $-\infty$ to $+\infty$, whereas probabilities must be between 0 and 1. The sigmoid function in logistic regression ensures that the output is properly bounded.

$$\text{Linear regression: } y = \beta_0 + \beta_1 X_1 + ... + \beta_n X_n \in (-\infty, +\infty)$$

$$\text{Logistic regression: } P(Y = 1|X) = \sigma(\beta_0 + \beta_1 X_1 + ... + \beta_n X_n) \in (0, 1)$$

2. Linear relationship assumption

Linear regression assumes a linear relationship between the dependent and independent variables. In classification, the relationship is typically non-linear, which the sigmoid function can model effectively.

3. Sensitivity to outliers

Linear regression is more sensitive to outliers compared to logistic regression, which can affect the decision boundary significantly.

Why not use Linear Regression for classification?

1. Thresholding issues

When using linear regression for classification, we'd need to set a threshold to convert the continuous output to a binary decision. The choice of this threshold is arbitrary and can lead to poor classification performance.

2. Violated assumptions

Linear regression assumes that errors are normally distributed with constant variance. In binary classification problems, these assumptions are violated because the response variable can only take two values (0 or 1).

3. Suboptimal decision boundaries

Linear regression minimizes the sum of squared errors, which is not the appropriate objective for classification tasks. This can result in suboptimal decision boundaries.

Consider a binary classification example where most data points are at the extremes (clearly class 0 or clearly class 1). Linear regression would try to fit a line through these points, whereas logistic regression would create a more appropriate S-shaped decision boundary.

Pros and Cons of Logistic Regression

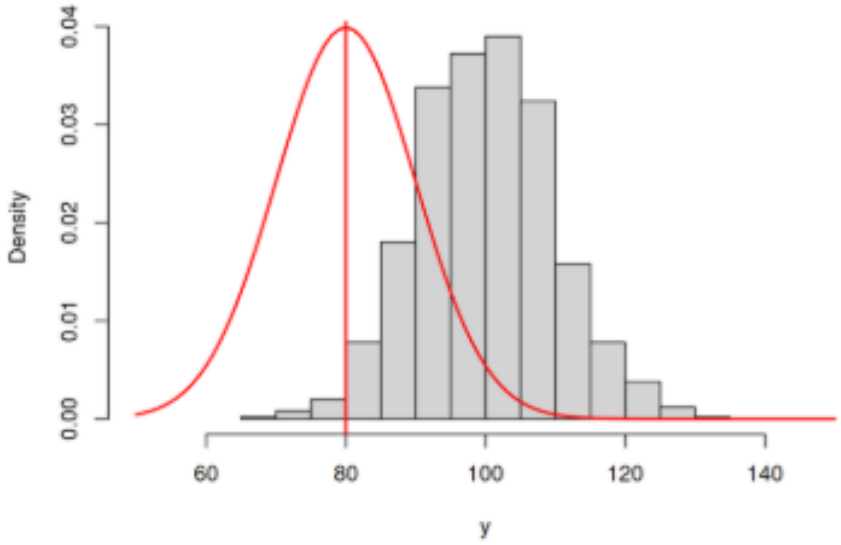
Pros:

- Simple and interpretable: The coefficients directly represent the log-odds of the outcome.
- Efficient training: Requires less computational resources compared to more complex models.
- Works well for linearly separable data.
- Outputs well-calibrated probabilities: The probabilities from logistic regression are usually reliable.
- Less prone to overfitting in high-dimensional spaces when regularization is used.
- No assumptions about distributions of classes in feature space.

Cons:

- Limited to linear decision boundaries: Cannot capture complex non-linear relationships without feature engineering.
- Struggles with imbalanced datasets without proper adjustments.
- May have convergence issues when features are highly correlated.
- Requires careful feature selection and engineering.
- Assumes independence among features (though this can be partially addressed).
- Not suitable for problems requiring high accuracy in complex domains.

What is MLE (Maximum Likelihood Estimation)



Maximum Likelihood Estimation (MLE) is a method used to estimate the parameters of a statistical model. In logistic regression, MLE finds the model parameters (coefficients) that maximize the likelihood of observing the given data.

For logistic regression, the likelihood function measures how likely the observed binary outcomes are, given the current model parameters and input features.

The likelihood function for n independent observations can be written as:

$$L(\beta) = \prod_{i=1}^{n} P(Y_i|X_i, \beta)^{Y_i} \cdot (1 - P(Y_i|X_i, \beta))^{1-Y_i}$$

Where:

- Y_i is the observed outcome (0 or 1) for the i-th data point
- X_i represents the input features for the i-th data point
- $P(Y_i|X_i, \beta)$ is the predicted probability for the i-th data point
- β represents the model parameters

We want to find the values of β that maximize this function. For computational convenience, we typically maximize the log-likelihood instead:

$$\log L(\beta) = \sum_{i=1}^{n} [Y_i \log(P(Y_i|X_i, \beta)) + (1 - Y_i) \log(1 - P(Y_i|X_i, \beta))]$$

Loss Function of Logistic Regression

In practice, we typically minimize the negative log-likelihood, which is called the log loss or cross-entropy loss:

$$J(\beta) = -\frac{1}{n}\sum_{i=1}^{n}[Y_i \log(h_\beta(X_i)) + (1 - Y_i)\log(1 - h_\beta(X_i))]$$

Where $h_\beta(X_i)$ is the predicted probability $P(Y_i = 1|X_i, \beta)$ for the i-th data point.

Let's break down this loss function:

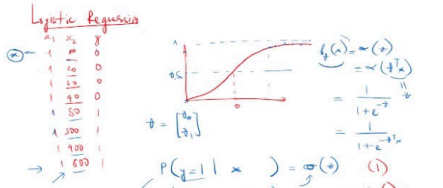
- When $Y_i = 1$, the second term becomes 0, and we're left with $-log(h_\beta(X_i))$
- When $Y_i = 0$, the first term becomes 0, and we're left with $-log(1 - h_\beta(X_i))$

This means:

- For positive examples $Y = 1$, we want $h_\beta(X)$ to be close to 1 to minimize $-log(h_\beta(X))$
- For negative examples $Y = 0$, we want $h_\beta(X)$ to be close to 0 to minimize $-log(1 - h_\beta(X))$

The loss function penalizes confident wrong predictions more heavily than less confident ones, which aligns with our goal of obtaining accurate probabilities.

Logistic Regression



$h_\theta(x) = \sigma(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$

$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$

$P(y=1 \mid x) = \sigma(\theta)$ (1)

$P(y=0 \mid x) = 1 - \sigma(\theta)$ (2)

$P(y=1 \mid \begin{bmatrix} 1 \\ 800 \end{bmatrix}) = 0.9 \ (90\%)$

$\rightarrow P(y=0 \mid \begin{bmatrix} 1 \\ 100 \end{bmatrix}) = 1 - 0.9 = 0.1 = 10\%$

Hàng $P(y \mid x) = \sigma(z)^y (1 - \sigma(z))^{1-y}$

$y=0$ Trường hợp 2 TH trên

$P(y=0 \mid x) = \sigma(z)^0 (1-\sigma(z))^{1-0} = 1 - \sigma(z)$

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

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

$P(y^{(2)} \mid x^{(2)}) = \sigma(z^{(2)})^{y^{(2)}} (1 - \sigma(z^{(2)}))^{1-y^{(2)}}$

$P(y^{(6)} \mid x^{(6)}) = \sigma(z^{(6)})^{y^{(6)}} (1 - \sigma(z^{(6)}))^{1-y^{(6)}}$

Những xs này tới nhất có thể

Tích tất cả những xs trên trường dùng là tới nhất có thể

(Maximum Likelihood Estimation)

$P(y^{(1)} \mid x^{(1)}) \dots P(y^{(6)} \mid x^{(6)})$ Lới nhất

$$\prod_{i=1}^{m=6} P(y^{(i)} \mid x^{(i)})$$

$$= \prod_{i=1}^{m=6} \sigma(z^{(i)})^{y^{(i)}} (1 - \sigma(z^{(i)}))^{1-y^{(i)}}$$

$$L(\theta) = \prod_{i=1}^{m} \sigma(z^{(i)})^{y^{(i)}} (1 - \sigma(z^{(i)}))^{1-y^{(i)}}$$

$$= \prod_{i=1}^{m} h_\theta(x^{(i)})^{y^{(i)}} (1 - h_\theta(x^{(i)}))^{1-y^{(i)}}$$

$$LL(\theta) = \log \prod_{i=1}^{m} h_\theta(x^{(i)})^{y^{(i)}} (1 - h_\theta(x^{(i)}))^{1-y^{(i)}}$$

$$= \sum_{i=1}^{m} \log \left(h_\theta(x^{(i)})^{y^{(i)}} (1 - h_\theta(x^{(i)}))^{1-y^{(i)}} \right)$$

$$= \sum_{i=1}^{m} \left[\log h_\theta(x^{(i)})^{y^{(i)}} + \log (1 - h_\theta(x^{(i)}))^{1-y^{(i)}} \right]$$

$$= \sum_{i=1}^{m} \left[y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)})) \right]$$

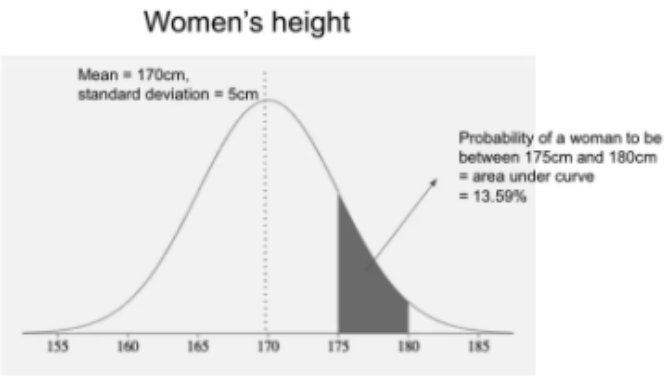
$J(\theta)$

Minimize $J(\theta) = -LL(\theta)$

Compare Likelihood and Probability

While often used interchangeably in casual conversation, likelihood and probability have distinct mathematical meanings in statistics:

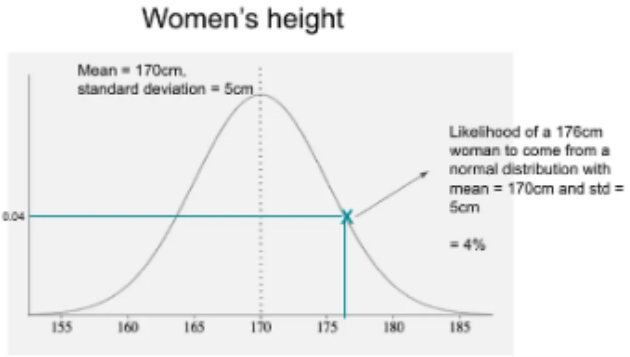
Probability



Probability describes the chance of an outcome occurring given fixed model parameters.

- Notation: $P(data|parameters)$
- Probability measures the chance of observing certain data given that we know the model parameters.
- In logistic regression, probability answers: "Given these coefficient values, what's the chance of observing $Y = 1$?"
- Probability is a function of the data, with parameters fixed.

Likelihood



Likelihood describes how likely particular parameter values are, given observed outcomes.

- Notation: $L(parameters|data)$
- Likelihood measures how plausible different parameter values are, given the observed data.
- In logistic regression, likelihood answers: "Given our observed Y values, how plausible are these coefficient values?"
- Likelihood is a function of the parameters, with data fixed.

Key Differences:

- Direction of conditioning: Probability conditions on parameters; likelihood conditions on data.
- Purpose: Probability predicts future data; likelihood evaluates parameter plausibility.
- In model fitting, we maximize likelihood (find most plausible parameters) rather than probability.

Gradient Descent of Loss Function

Gradient Descent is an optimization algorithm used to minimize the loss function in logistic regression. The goal is to find the model parameters (coefficients) that minimize the log loss function.

The Loss Function

First, let's recall our loss function for logistic regression:

$$J(\beta) = -\frac{1}{n}\sum_{i=1}^{n}[Y_i \log(h_\beta(X_i)) + (1-Y_i)\log(1-h_\beta(X_i))]$$

Where:

- $h_\beta(X_i) = \sigma(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_n X_n)$ is the sigmoid function applied to the linear combination of inputs
- X_i represents the feature vector for the i-th training example
- Y_i is the true label (0 or 1) for the i-th training example
- β represents the model parameters we're trying to optimize

The Gradient Descent Algorithm

Gradient descent works by iteratively updating the parameters in the direction of steepest descent of the loss function:

$$\beta_j := \beta_j - \alpha \cdot \frac{\partial J(\beta)}{\partial \beta_j}$$

Where:

- α is the learning rate (step size)
- $\partial J(\beta)/\partial \beta_j$ is the partial derivative of the loss function with respect to parameter β_j

Computing the Gradient Using the Chain Rule

To find the gradient, we need to calculate the partial derivatives of $J(\beta)$ with respect to each parameter β_j. This requires the chain rule since our loss function involves a composition of functions.

Step 1: Express $h_\beta(X_i)$ in terms of the sigmoid function:

$$h_\beta(X_i) = \sigma(z_i) = \frac{1}{1 + e^{-z_i}}$$

Where $z_i = \beta_0 + \beta_1 X_{i,1} + \beta_2 X_{i,2} + ... + \beta_n X_{i,n}$ is the linear predictor.

Step 2: Calculate the derivative of the sigmoid function:

$$\frac{d\sigma(z)}{dz} = \sigma(z)(1 - \sigma(z))$$

Step 3: Calculate the partial derivative of z_i with respect to β_j:

$$\frac{\partial z_i}{\partial \beta_j} = X_{i,j}$$

Note that $X_{i,0} = 1$ for the intercept term β_0.

Step 4: Calculate the partial derivative of the loss function with respect to $h_\beta(X_i)$:

$$\frac{\partial J(\beta)}{\partial h_\beta(X_i)} = -\frac{1}{n}\left(\frac{Y_i}{h_\beta(X_i)} - \frac{1 - Y_i}{1 - h_\beta(X_i)}\right)$$

Step 5: Apply the chain rule to find the partial derivative of $J(\beta)$ with respect to β_j:

$$\frac{\partial J(\beta)}{\partial \beta_j} = \sum_{i=1}^{n} \frac{\partial J(\beta)}{\partial h_\beta(X_i)} \cdot \frac{\partial h_\beta(X_i)}{\partial z_i} \cdot \frac{\partial z_i}{\partial \beta_j}$$

Substituting the expressions from steps 2, 3, and 4:

$$\frac{\partial J(\beta)}{\partial \beta_j} = \sum_{i=1}^{n} \left(-\frac{1}{n} \left(\frac{Y_i}{h_\beta(X_i)} - \frac{1 - Y_i}{1 - h_\beta(X_i)} \right) \right) \cdot h_\beta(X_i)(1 - h_\beta(X_i)) \cdot X_{i,j}$$

Step 6: Simplify the expression:

$$\frac{\partial J(\beta)}{\partial \beta_j} = -\frac{1}{n} \sum_{i=1}^{n} \left(\frac{Y_i(1 - h_\beta(X_i)) - (1 - Y_i)h_\beta(X_i)}{h_\beta(X_i)(1 - h_\beta(X_i))} \right) \cdot h_\beta(X_i)(1 - h_\beta(X_i)) \cdot X_{i,j}$$

$$\frac{\partial J(\beta)}{\partial \beta_j} = -\frac{1}{n} \sum_{i=1}^{n} (Y_i - h_\beta(X_i)) \cdot X_{i,j}$$

$$\frac{\partial J(\beta)}{\partial \beta_j} = \frac{1}{n} \sum_{i=1}^{n} (h_\beta(X_i) - Y_i) \cdot X_{i,j}$$

This final expression gives us the gradient of the loss function with respect to each parameter β_j.

Gradient Descent Algorithm Steps

Now, we can implement the gradient descent algorithm for logistic regression:

- Initialize parameters β to small random values or zeros
- Repeat until convergence (or for a fixed number of iterations):
  - Compute predictions: $h_\beta(X_i)$ for all training examples
  - Compute the gradient: $\nabla J(\beta) = (1/n) \sum (h_\beta(X_i) - Y_i) \cdot X_i, j$ for each parameter β_j
  - Update parameters: $\beta_j := \beta_j - \alpha \cdot \nabla J(\beta)_j$
- Return the optimized parameters β

In vectorized form, the gradient descent update can be written as:

$$\beta := \beta - \alpha \cdot \frac{1}{n} X^T (h_\beta(X) - Y)$$

Where:

- X is the design matrix (each row is a training example, each column is a feature)
- Y is the vector of true labels
- $h_\beta(X)$ is the vector of predicted probabilities

Confusion Matrix and Error Metrics

The confusion matrix is a table that describes the performance of a classification model on test data for which the true values are known. For binary classification, it has the following structure:

	Predicted Negative	Predicted Positive
Actual Negative	True Negative (TN)	False Positive (FP)
Actual Positive	False Negative (FN)	True Positive (TP)

From the confusion matrix, we can derive several performance metrics:

Accuracy

The proportion of correctly classified instances:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Limitation: Not reliable for imbalanced datasets where one class is much more frequent than the other.

Precision

The proportion of positive identifications that were actually correct:

$$\text{Precision} = \frac{TP}{TP + FP}$$

High precision indicates a low false positive rate. It answers: "Of all instances predicted as positive, how many were actually positive?"

Recall (Sensitivity)

The proportion of actual positives that were correctly identified:

$$\text{Recall} = \frac{TP}{TP + FN}$$

High recall indicates a low false negative rate. It answers: "Of all actual positive instances, how many did we correctly identify?"

F1 Score

The harmonic mean of precision and recall:

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

F1 score balances precision and recall and is useful when classes are imbalanced.