What is Logistic Regression?

Despite the name, **Logistic Regression is a classification algorithm**, not a regression algorithm.
It is used when the **dependent variable (output)** is **categorical** — usually binary (e.g., 0/1, True/False, Yes/No).

💡 **Intuition**

Let's say you want to predict whether a student will **pass (1)** or **fail (0)** based on their **hours of study**.

If you used **Linear Regression**, your output might be something like:

y = 0.5 * (hours) + 1

But this could give results like **y = 5 or y = -2**, which are not valid probabilities.

So we need a function that maps **any real number → [0, 1]** range.

That's where the **Sigmoid Function** comes in.

🧮 Step 1: The Sigmoid (Logistic) Function

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

where

$$z = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$$

- If `z → ∞`, then `σ(z) → 1`
- If `z → -∞`, then `σ(z) → 0`

Thus, the output of Logistic Regression can be interpreted as a **probability**.

📈 Step 2: Hypothesis

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

where

- $h_\theta(x)$ = Predicted probability that $y = 1$
- w = weights (parameters)
- x = features

If

- $h_\theta(x) \geq 0.5 \rightarrow$ predict **1**
- $h_\theta(x) < 0.5 \rightarrow$ predict **0**

⚙️ Step 3: Cost (Loss) Function

In linear regression, we use **Mean Squared Error (MSE)**.
But in logistic regression, MSE doesn't work well because the function is non-linear.

We use **Log Loss (Binary Cross-Entropy Loss):**

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

where

- m = number of samples
- $y^{(i)}$ = actual label (0 or 1)
- $h_\theta(x^{(i)})$ = predicted probability

🧠 Intuition:

- If prediction is close to actual value → loss is small
- If prediction is wrong → loss is large

🧩 Step 4: Optimization (Gradient Descent)

We minimize the loss function using **Gradient Descent**:

$$w := w - \alpha \frac{\partial J(w)}{\partial w}$$

where

- α = learning rate
- $\frac{\partial J(w)}{\partial w}$ = gradient of cost w.r.t weights

This adjusts the weights step-by-step to minimize prediction error.

📊 Step 5: Decision Boundary

The model predicts probabilities, but we need a **decision**.

For binary classification:

$$\text{Predict } y = \begin{cases} 1 & \text{if } h_\theta(x) \geq 0.5 \\ 0 & \text{if } h_\theta(x) < 0.5 \end{cases}$$

The line (or surface) where $h_\theta(x) = 0.5$ is called the **Decision Boundary**.

Example in 2D:

$$w_0 + w_1 x_1 + w_2 x_2 = 0$$

This is a straight line that separates the two classes.

🧠 **Example in Python**

```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

```
# Example Data
X = np.array([[1], [2], [3], [4], [5]])  # hours of study
y = np.array([0, 0, 0, 1, 1])           # fail or pass

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Model
model = LogisticRegression()
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

print("Predictions:", y_pred)
print("Accuracy:", accuracy_score(y_test, y_pred))
```

📘 Extensions of Logistic Regression

Type	Description
Binary Logistic Regression	2 classes (e.g., Yes/No)
Multinomial Logistic Regression	More than 2 classes (uses softmax)
Regularized Logistic Regression	Adds L1/L2 penalties to prevent overfitting

---

⚖️ Advantages

- ✅ Simple and easy to implement
- ✅ Handles the problem of outliers
- ✅ Works well for linearly separable data
- ✅ Outputs probabilistic interpretation
- ✅ Efficient and widely used in real-world applications

⚠️ Limitations

❌ Doesn't handle non-linear relationships well (unless you use polynomial features)
❌ Sensitive to outliers
❌ Performance degrades if features are highly correlated or not scaled properly

---

🧭 Real-Life Applications

- Spam detection (spam / not spam)
- Disease prediction (positive / negative)
- Credit risk modeling (default / not default)
- Customer churn prediction

🧠 Concept Recap

Hypothesis Function:

$$h(x) = \frac{1}{1 + e^{-(w*x+b)}}$$

Loss Function (Binary Cross-Entropy):

$$Loss = -\frac{1}{m}\sum[y_i \log(h(x_i)) + (1 - y_i)\log(1 - h(x_i))]$$

Gradient Descent Updates:

$$w = w - \alpha \cdot \frac{dL}{dw}$$

$$b = b - \alpha \cdot \frac{dL}{db}$$