# Classification: Logistic Regression

## Why Linear Regression is not a good fit for **classification** problems:

1. **Linear Regression outputs continuous values**
    - Linear regression tries to predict a **number** (like 3.2, 7.8, etc.).
    - But in classification, we want **discrete categories** (e.g., "Spam" vs "Not Spam", or 0/1).
    - If we use linear regression, we'd need to forcefully set a threshold (e.g., if prediction ≥ 0.5 → class 1, else class 0).
    - This is not stable.

2. **Unbounded outputs**
    - A regression line can predict anything: **∞ to +∞**.
    - But probabilities must lie between **0 and 1**.
    - For example, if regression predicts 1.2 or -0.7, what does that mean in terms of probability? → Makes no sense.

3. **Not robust to outliers**
    - If there's even a few extreme data points, the regression line can get pulled badly.
    - This can flip classification decisions.
    - Logistic Regression (which is designed for classification) handles this better with the **sigmoid function**.

4. **Assumption of linearity may not hold**

- Linear regression assumes a straight-line relation between input and output.
- But class boundaries are often **non-linear**.
- Linear regression won't capture these well.

---

## 🔷 The Better Alternative → **Logistic Regression**

- Logistic Regression fixes these issues:
  - Uses a **sigmoid function** to squash outputs between 0 and 1.
  - Interprets outputs as **probabilities**.
  - Uses **log loss** instead of MSE, which works better for classification.

---

✅ In short:

- Linear Regression is great for **predicting continuous values**.
- But for **classification**, it's unstable and unbounded.
- Logistic Regression (or other classifiers) is the right tool.

---

# Logistic Regression

Perfect question 🎯 . Let's build it step by step so it's crystal clear.

---

## 🔷 Logistic Regression Explained

### 1. What is Logistic Regression?

Despite the name, **Logistic Regression is not for regression tasks** — it's actually a **classification algorithm**.

It is mainly used when the target variable is **binary** (yes/no, spam/not spam, 0/1).

---

### 2. The Core Idea

- Linear regression predicts a continuous value:

$$y = w^T x + b$$

- But for classification, we want a **probability** between **0 and 1**.

👉 Logistic Regression fixes this by applying the **sigmoid function** (also called the logistic function) to the linear output:

$$P(y = 1|x) = \sigma(w^T x + b) = \frac{1}{1 + e^{-(w^T x + b)}}$$

- Output is always between **0 and 1** ✅ (interpretable as probability).

---

## 3. Decision Rule

- After we get the probability, we apply a **threshold** (usually 0.5):
  - If $P(y = 1|x) \geq 0.5$ → predict **class 1**
  - If $P(y = 1|x) < 0.5$ → predict **class 0**

Example:

- If the model outputs 0.87 → predict **1**
- If model outputs 0.22 → predict **0**

---

## 4. Why not use MSE like Linear Regression?

- For classification, MSE isn't a good loss function.
- Logistic regression instead uses **Log Loss / Cross-Entropy Loss**:

$$\text{Loss} = -\big[y \log(p) + (1 - y) \log(1 - p)\big]$$

This penalizes wrong predictions more effectively.

---

## 5. Why is Logistic Regression Useful for Binary Classification?

- Outputs probabilities, not just classes.
- Easy to interpret: "There's a 78% chance this email is spam."

- Works well as a **baseline model** before moving to more complex classifiers (like decision trees, SVMs, or neural networks).
- Computationally efficient and interpretable.

---

✅ **In short:**

Logistic regression = Linear model + Sigmoid + Cross-entropy loss

👉 Makes it a perfect tool for binary classification tasks.

---

# Loss function

## 🧠 Step 1: What Logistic Regression Is Doing

In **logistic regression**, we're not predicting hard class labels (0 or 1) directly.

Instead, we predict the **probability** that a sample belongs to class 1:

$$\hat{y} = P(y = 1|x) = \sigma(w^T x) = \frac{1}{1 + e^{-w^T x}}$$

where:

- $\hat{y}$ = predicted probability that output = 1
- $w^T x$ = linear combination of features
- $\sigma$ = sigmoid function, which maps any real number to (0, 1)

---

## ⚙️ Step 2: How We Want the Model to Behave

We want:

- $\hat{y}$ to be **close to 1** when actual y = 1
- $\hat{y}$ to be **close to 0** when actual y = 0

So, we need a **loss function** that:

- **Penalizes** wrong predictions heavily
- **Rewards** correct predictions

- Is **smooth and differentiable** (for gradient descent)

---

## 📉 Step 3: Defining the Loss for One Example

We define the **loss for a single training example** as:

$$L(y, \hat{y}) = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})]$$

Why this formula?

Let's see both cases:

### Case 1: y = 1

Then the loss becomes:

$$L = -\log(\hat{y})$$

→ If $\hat{y}$ = 1, loss = 0 (perfect prediction)

→ If $\hat{y}$ = 0.1, loss = large (bad prediction)

### Case 2: y = 0

Then the loss becomes:

$$L = -\log(1 - \hat{y})$$

→ If $\hat{y}$ = 0, loss = 0 (perfect prediction)

→ If $\hat{y}$ = 0.9, loss = large (bad prediction)

So this formula **automatically handles both cases**.

---

## 📦 Step 4: The Overall Cost Function

We take the average loss across all training examples:

$$J(w) = -\frac{1}{n} \sum_{i=1}^{n} \left[ y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \right]$$

This is called the **Binary Cross-Entropy Loss** or **Log Loss**.

---

## 🎯 Step 5: Why It Makes Sense (Intuition)

You can think of it as measuring **how surprised the model is** by the true label:

- If the model predicts a probability close to the truth, it's **not surprised** → small loss.
- If the model predicts the wrong class confidently, it's **very surprised** → large loss.

This idea comes from **information theory**, where:

> Cross-entropy measures the "distance" between the true distribution (labels) and predicted probability distribution.

---

## 🧩 Step 6: Gradient and Learning

To minimize this loss, we use **gradient descent**.

The gradient of the loss with respect to weights www is:

$$\frac{\partial J}{\partial w} = \frac{1}{n} \sum_i (\hat{y}_i - y_i) x_i$$

Notice:

- $(\hat{y} - y)$ shows how far our prediction is from the truth.
- If the prediction is too high, the weights decrease.
- If the prediction is too low, weights increase.

This is why the logistic regression update rule looks similar to linear regression — but with **sigmoid probabilities** instead of raw outputs.

---

## 📊 Step 7: Visual Intuition

| $\hat{y}$ | y | $Loss - [y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})]$ | Interpretation |
|---|---|---|---|
| 0.99 | 1 | 0.01 | Very good prediction |
| 0.6 | 1 | 0.51 | Moderate prediction |
| 0.1 | 1 | 2.30 | Very bad (model was confident but wrong) |
| 0.1 | 0 | 0.10 | Good prediction |
| 0.9 | 0 | 2.30 | Very bad (wrong with high confidence) |

## ✅ Summary

| Concept | Explanation |
|---|---|
| **Loss for one sample** | $L = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})]$ |
| **Total cost function** | $J(w) = \frac{1}{n} \sum L_i$ |
| **Purpose** | Measure how well predicted probabilities match true labels |
| **Optimization method** | Gradient Descent (minimize cross-entropy) |
| **Effect** | Penalizes confident wrong predictions heavily; rewards correct predictions smoothly |

# Logistic Regression vs PLA

### PLA:

The perceptron predicts:

$$\hat{y} = \text{sign}(w^T x)$$

It updates weights only when there is a misclassification:

$$w \leftarrow w + \eta(y - \hat{y})x$$

✅ Works only if the data is **linearly separable**.

❌ If data overlaps (not perfectly separable), it **never converges**.

## Logistic Regression:

The model predicts a probability using the **sigmoid function**:

$$P(y=1|x) = \sigma(w^T x) = \frac{1}{1+e^{-w^T x}}$$

The decision boundary is at P(y=1|x) = 0.5.

Weights are learned by minimizing **cross-entropy (log loss)**:

$$J(w) = -\sum [y \log(\hat{y}) + (1-y)\log(1-\hat{y})]$$

✅ Works **even if the data is not linearly separable**.

✅ Gives **probabilistic confidence** of predictions.

✅ Uses **gradient descent** for optimization.

| Feature | PLA | Logistic Regression |
|---|---|---|
| **Type of classifier** | Hard classifier | Probabilistic classifier |
| **Learning rule** | Updates only on mistakes | Updates based on all samples (via gradient descent) |
| **Loss function** | No explicit loss (just mistake-driven) | Cross-entropy (smooth and differentiable) |
| **Convergence** | Only guaranteed if data is linearly separable | Always converges (to best possible fit) |
| **Output** | ±1 (hard decision) | [0,1] probability (soft decision) |
| **Interpretability** | No direct probability interpretation | Clear probability meaning (e.g., “80% chance of class 1”) |
| **Robustness to noise** | Poor | Much better (thanks to probabilistic modeling) |

# Intuition

Think of it this way:

- **PLA** says: “Is this point correctly classified or not?” (binary yes/no feedback)
- **Logistic Regression** says: “How confident am I in this classification?” and adjusts accordingly.

This *smooth adjustment* allows logistic regression to learn more gracefully — it doesn’t overreact to individual mistakes like PLA does.

---