# Logistic Regression For Binomial Classification

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

Problem:
You have data of students that studied for certain hours and their pass or fail status.

| Study Hours | Pass/Fail |
|---|---|
| 1.0 | 0 |
| 2.5 | 0 |
| 3.0 | 0 |
| 4.5 | 1 |
| 5.0 | 1 |
| 6.0 | 1 |

Here,
0 -> Fail
1-> Pass

Design a machine learning model that can determine pass/fail status of a student who scored 3.4.

# Logistic Regression For Binomial Classification

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

This problem solved by python and scikit-learn

| Study Hours (X) | Pass/Fail (y) |
|---|---|
| 1.0 | 0 |
| 2.5 | 0 |
| 3.0 | 0 |
| 4.5 | 1 |
| 5.0 | 1 |
| 6.0 | 1 |

```python
from sklearn.linear_model import LogisticRegression
import numpy as np

# Data (Study Hours and Pass/Fail)
X = np.array([[1], [2.5], [3], [4.5], [5], [6]])   # Feature
y = np.array([ 0,    0,    0,    1,    1,   1])   # Target

# Create and train the model
model = LogisticRegression()
model.fit(X, y)

# Predict probabilities for new students
hours = np.array([[3.4],])
probs = model.predict_proba(hours)

# Display the probabilities of passing (class 1)
for h, p in zip(hours.flatten(), probs[:, 1]):
    print(f"Study Hours: {h}, Probability of Passing: {p:.2f}")
```

```
Study Hours: 3.4, Probability of Passing: 0.41
```

# Logistic Regression For Binomial Classification

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

Sometime we have to predict whether a data point belongs to certain category. Category can be

**1. Binomial:** There can be only two possible types of categories
Example: Customer is going to **buy or not buy;**
Student either **Pass or Fail;**
email is **Spam or Not Spam;**
Patient has **Cancer or no-cancer**

**2. Multinomial:** There can be 3 or more possible types of categories
Example: Image is either **cat, dog, or sheep**;
Iris flower can be either **setosa, versicolor or virginica**

We need a model that can classify data points in classes/categories:
This is where **Logistic Regression** come.
**Logistic Regression** is a model that predicts **probability** ( a number between 0 and 1) and then applies a **threshold** (usually 0.5) to determine which class the data belongs to.

# Logistic Regression For Binomial Classification

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

To perform binomial classification, we use **sigmoid function**:

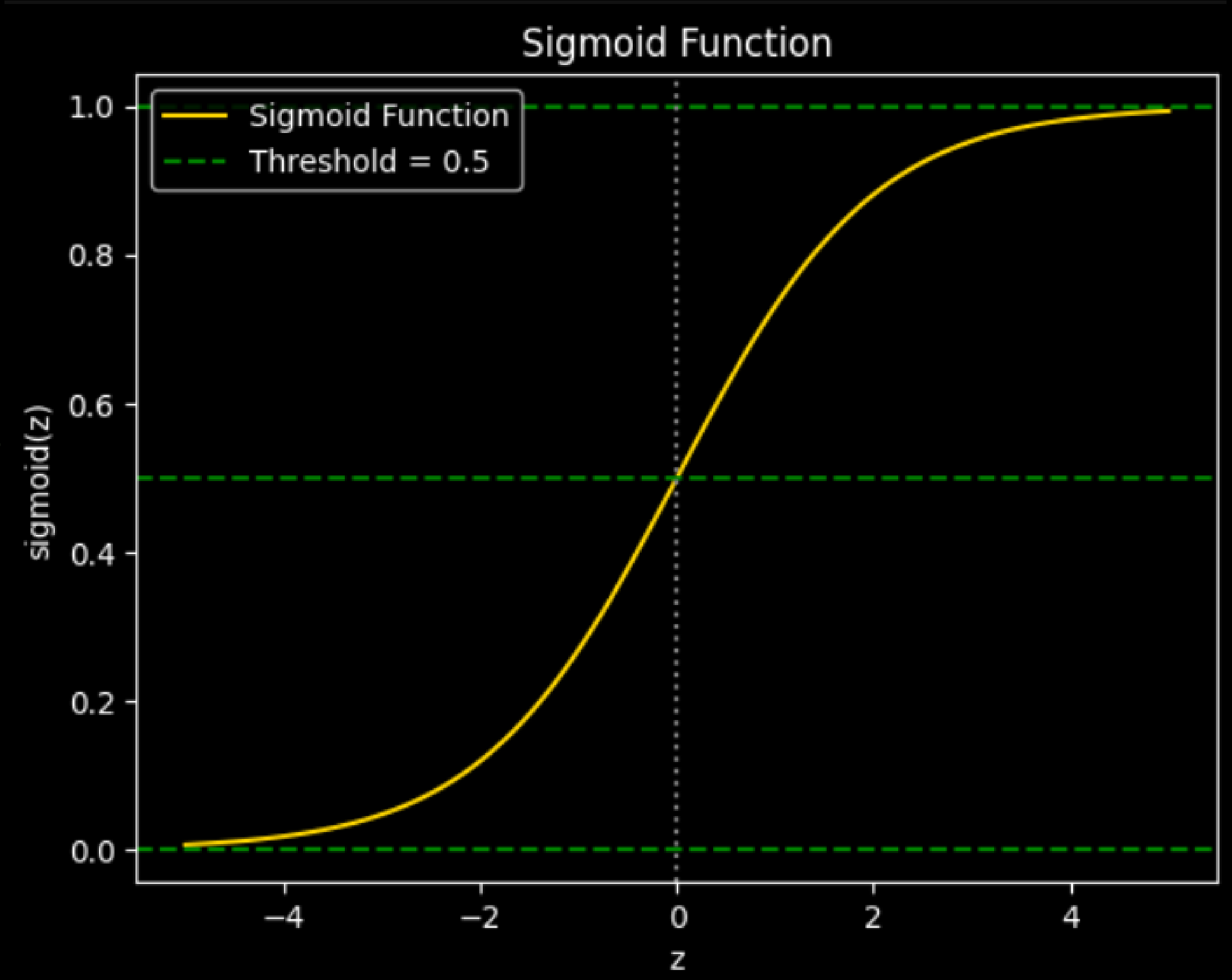
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Important features of sigmoid function:

- Maps any real value of z to a range between **0 and 1**.
- Often used in **logistic regression** to convert linear score z into probability. i.e. The probability of obtaining z
- It is used to classify objects in 2 classes
- e is Euler Number, 2.71828…It is an irrational number

Example: Calculate probability of z=2

$$\sigma(2) = \frac{1}{1 + e^{-2}} \quad = \frac{1}{1 + 0.1353} \quad \approx 0.8808$$



# Logistic Regression For Binomial Classification

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

$$x \longrightarrow z = b_0 + b_1 x \longrightarrow \sigma(z) = \frac{1}{1+e^{-z}}$$

Step-by-Step Intuition

1. Logistic regression first calculates a **linear score** (In our example, x is StudyHours):

$$z = b_0 + b_1 \times \text{StudyHours}$$

The bias $b_0$ and weight $b_1$ are calculated during training from data .

2. Then applies the **sigmoid function** to convert z into a probability:

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

The output is a probability of **Pass (1)** and is between 0 and 1.

3. If $P(Pass) > 0.5$ → predict **Pass (1)**
   If $P(Pass) \leq 0.5$ → predict **Fail (0)**

# Logistic Regression For Binomial Classification

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

$$x \longrightarrow z = b_0 + b_1 x \longrightarrow \sigma(z) = \frac{1}{1+e^{-z}}$$

Example: Assuming that from training data, we have calculated bias $b_0 = -4.35$ and weight $b_1 = 1.17$. and we want to find whether the student who scored 3.4, did he/she pass.

Step-by-Step:

1. Logistic regression first calculates a **linear score** (x is StudyHours):

$$z = b_0 + b_1 \times \text{StudyHours}$$

z = -4.35 + 1.17 x 3.4 = - 0.372

2. Then applies the **sigmoid function** to convert z into a probability:

$$\sigma(z) = \frac{1}{1+e^{-z}} \qquad \sigma(-0.372) = \frac{1}{1+e^{0.372}} \qquad \sigma(-0.372) = 0.41 = 41\ \%$$

This is the probability of **Pass (1).**

3. Here, $P(Pass) \leq 0.5$ →predict **Fail (0)**

# Recap On Work Flow Of Logistic Regression

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

Machine Learning Model Before Training

$b_0 =?\ and\ b_1 = ?$

Machine Learning Model After Training

**Training Data** →

*Calculates*
$b_0 = -4.35\ and$
$b_1 = 1.17$

**Feed x = 3.4** →

$x \rightarrow z \rightarrow \sigma(z)$

$3.4 \rightarrow -0.372 \rightarrow 0.41$

# Logistic Regression with Many Features

Let's say you have many features: study hours, sleep time before the exam, previous score, etc

When you have multiple features:

$$X = (x_1, x_2, x_3, \dots, x_{\mathrm{n}}) = (\text{study hours}, \text{sleep time}, \text{previous score})$$

Logistic regression models the probability that the output $y = 1$.

Step1: For **many features**, the model first computes:

$$\mathrm{z} = \ b_0 + b_1 x_1 + b_2 x_2 + \ b_2 x_2 + \ \dots + \ b_{\mathrm{n}} x_n$$

where,
$b_0$ = bias/intercept
$b_1$, $b_2$, …., $b_{1n}$ are called weights. These are calculated during training from the training data.
$x_1$, $x_2$, …., $x_{1n}$ are called features

Step2: Then applies the **sigmoid function** to convert z into a probability:

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

Step3: If $P(1) > 0.5$ → predict **class 1**
If $P(1) \leq 0.5$ → predict **class 0**

# Logistic Regression with Many Features

Python code that calculates the probability for data that has 3 features:

| X1 | X2 | X3 | y |
|---|---|---|---|
| 2.5 | 1.3 | 0.5 | 1 |
| 1.0 | 3.5 | 2.2 | 0 |
| 3.2 | 0.9 | 1.4 | 1 |
| 4.1 | 2.2 | 0.1 | 1 |
| 0.9 | 4.5 | 3.1 | 0 |
| 3.9 | 1.2 | 0.4 | 1 |

```python
# logistic regression for many features
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression

# Features: X1, X2, X3
X = np.array([
    [2.5, 1.3, 0.5],
    [1.0, 3.5, 2.2],
    [3.2, 0.9, 1.4],
    [4.1, 2.2, 0.1],
    [0.9, 4.5, 3.1],
    [3.9, 1.2, 0.4]
])

# Target values (binary)
y = np.array([1, 0, 1, 1, 0, 1])

# Train logistic regression
model = LogisticRegression()
model.fit(X, y)

# Predict probability for a new sample
x_new = np.array([[3.0, 1.5, 0.7]])

prob = model.predict_proba(x_new)
print("Probability y=1:", prob[0][1])
```

```
Probability y=1: 0.9006659215352736
```

# ADVANCED

# Logistic Regression

**Assumptions of Logistic Regression**

**1. Independent observations**:
There should be no correlation or dependence between the input samples.

**2. Linearity relationship between independent variables and log odds**:
The model assumes a linear relationship between the independent variables and the log odds of the dependent variable which means the predictors affect the log odds in a linear way.

**3. No outliers**:
The dataset should not contain extreme outliers as they can distort the estimation of the logistic regression coefficients.

**4. Large sample size**:
It requires a sufficiently large sample size to produce reliable and stable results.

# Logistic Regression: Find the bias $b_0$ and weight $b_1$

## Goal of Logistic Regression:

We're modeling the probability that an outcome $y = 1$ (e.g., student passes) given input $x$ (e.g., study hours):

$$P(y = 1|x) = \frac{1}{1 + e^{-(b_0 + b_1 x)}}$$

where

- $b_0$ = intercept (bias term)
- $b_1$ = slope (how much study hours influence probability)

Step 1: The Model's Prediction

For any student with study hours $x_i$:

$$\hat{y}_i = P(y_i = 1|x_i) = \frac{1}{1 + e^{-(b_0 + b_1 x_i)}}$$

and

$$1 - \hat{y}_i = P(y_i = 0|x_i)$$

# Logistic Regression: Find the bias $b_0$ and weight $b_1$

Step 2: Likelihood Function

We want the model to assign **high probability to the actual outcomes** in the training data.

For all $n$ data points:

$$L(b_0, b_1) = \prod_{i=1}^{n} [\hat{y}_i]^{y_i} [1 - \hat{y}_i]^{(1-y_i)}$$

- If $y_i = 1$, the term becomes $\hat{y}_i$
- If $y_i = 0$, the term becomes $1 - \hat{y}_i$

This is called the **likelihood function** — it represents how likely our parameters $b_0$, $b_1$ make the data we observe.

# Logistic Regression: Find the bias $b_0$ and weight $b_1$

Step 3: Log-Likelihood Function

It's easier to work with logs (to turn the product into a sum):

$$\ell(b_0, b_1) = \sum_{i=1}^{n} \left[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)\right]$$

This is the **log-likelihood function** — it's what we want to **maximize** to find the best $b_0$ and $b_1$.

Step 4: Optimization ( Finding $b_0$ and $b_1$ )

To find optimal parameters:

$$\text{maximize } \ell(b_0, b_1)$$

or equivalently,

$$\text{minimize } -\ell(b_0, b_1)$$

which is called the **log loss (or cross-entropy loss)**.

Or minimize the **Binary cross-entropy cost function**

$$J = -\frac{1}{n}\ell(b_0, b_1)$$

# Logistic Regression: Find the bias $b_0$ and weight $b_1$

Step 5: Gradient Descent Intuition

We use **iterative optimization methods** — typically **Gradient Descent** or **Newton-Raphson,** to solve above problem

We start with random guesses for $b_0, b_1$, and iteratively update:

$$b_j := b_j + \alpha \frac{\partial \ell}{\partial b_j} \qquad\qquad b_j := b_j + \alpha \sum_{i=1}^{n} (y_i - \hat{y}_i) x_{ij}$$

where

- $\alpha$ = learning rate (step size)
- $(y_i - \hat{y}_i)$ = prediction error

Over many iterations, the parameters move in the direction that **increases the likelihood** (i.e., reduces log loss).

# Logistic Regression: Find the bias $b_0$ and weight $b_1$

Question: Why we do not use derivative to solve ?

In logistic regression:

$$\hat{y}_i = \frac{1}{1 + e^{-(b_0 + b_1 x_i)}}$$

We maximize the **log-likelihood**:

$$\ell(b_0, b_1) = \sum_i [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

Now, take derivatives:

$$\frac{\partial \ell}{\partial b_0} = \sum_i (y_i - \hat{y}_i)$$

$$\frac{\partial \ell}{\partial b_1} = \sum_i (y_i - \hat{y}_i) x_i$$

If we try to set these equal to zero, we get:

$$\sum_i (y_i - \hat{y}_i) = 0, \quad \sum_i (y_i - \hat{y}_i) x_i = 0$$

But $\hat{y}_i$ itself is a nonlinear function of $b_0$ and $b_1$ (because of the exponential).

So these equations are **nonlinear** in the parameters — there's no algebraic way to isolate $b_0, b_1$.

They're coupled inside an exponential and a fraction.

Hence, **no closed-form solution**.

# Logistic Regression: Find the bias $b_0$ and weight $b_1$

**Step-by-step numeric example:** Below is an example that shows how $b_0$ and $b_1$ change using **gradient ascent on the log-likelihood** (equivalently: gradient updates that *increase* the log-likelihood).
I use a tiny toy dataset of **3 students** so I can show each arithmetic step on the paper.

- Data:
  $x = [1, 2, 3]$ (study hours)
  $y = [0, 0, 1]$ (0 = fail, 1 = pass)
- Model:
  $\hat{y}_i = \sigma(z_i), \quad z_i = b_0 + b_1 x_i$
  $\sigma(z) = \dfrac{1}{1 + e^{-z}}$
- Gradients of log-likelihood (for all data points):

$$g_{b_0} = \sum_{i=1}^{n} (y_i - \hat{y}_i), \qquad g_{b_1} = \sum_{i=1}^{n} (y_i - \hat{y}_i) x_i$$

- Gradient ascent update:

$$b_j \leftarrow b_j + \alpha \, g_{b_j}$$

  (We use gradient **ascent** to *maximize* log-likelihood. If you prefer minimizing negative log-likelihood, update signs flip.)
- Hyperparameters / initial values:

$$b_0^{(0)} = 0, \quad b_1^{(0)} = 0, \quad \alpha = 0.1$$

# Logistic Regression: Find the bias $b_0$ and weight $b_1$

**Iteration 1 (initial $b_0 = 0, b_1 = 0$)**

1. Compute linear scores $z_i$:

$$z = [0 + 0 \cdot 1,\ 0 + 0 \cdot 2,\ 0 + 0 \cdot 3] = [0, 0, 0]$$

2. Sigmoid (predicted probabilities) $\hat{y}_i = \sigma(z_i)$:

$$\hat{y} = [\sigma(0), \sigma(0), \sigma(0)] = [0.5, 0.5, 0.5]$$

3. Errors $e_i = y_i - \hat{y}_i$:

$$e = [0 - 0.5,\ 0 - 0.5,\ 1 - 0.5] = [-0.5, -0.5, 0.5]$$

4. Gradients:

$$g_{b_0} = \sum e_i = -0.5$$

$$g_{b_1} = \sum e_i x_i = (-0.5) \cdot 1 + (-0.5) \cdot 2 + (0.5) \cdot 3 = -0.5 - 1 + 1.5 = 0.0$$

5. Update parameters:

$$b_0 \leftarrow 0 + 0.1 \times (-0.5) = -0.05$$

$$b_1 \leftarrow 0 + 0.1 \times 0.0 = 0.0$$

**After iteration 1:** $b_0 = -0.05,\ b_1 = 0.0$

# Logistic Regression: Find the bias $b_0$ and weight $b_1$

**Iteration 2 (start $b_0 = -0.05, b_1 = 0$)**

1. **Linear scores:**

$$z = [-0.05,\ -0.05,\ -0.05]$$

2. **Sigmoid:**

$$\sigma(-0.05) \approx 0.4875026035157896$$

So

$$\hat{y} \approx [0.4875026035,\ 0.4875026035,\ 0.4875026035]$$

3. **Errors:**

$$e \approx [-0.4875026035,\ -0.4875026035,\ 0.5124973965]$$

4. **Gradients:**

$$g_{b_0} = \sum e_i \approx -0.4625078105473689$$

$$g_{b_1} = \sum e_i x_i \approx (-0.4875026) \cdot 1 + (-0.4875026) \cdot 2 + (0.5124974) \cdot 3 \approx 0.07498437890526222$$

5. **Update:**

$$b_0 \leftarrow -0.05 + 0.1 \times (-0.4625078105473689) \approx -0.09625078105473689$$

$$b_1 \leftarrow 0 + 0.1 \times 0.07498437890526222 \approx 0.007498437890526222$$

**After iteration 2:** $b_0 \approx -0.0962507811,\ b_1 \approx 0.0074984379$

# Logistic Regression: Find the bias $b_0$ and weight $b_1$

**Iteration 3 (start $b_0 \approx -0.09625078,\ b_1 \approx 0.00749844$)**

1. Linear scores:

$$z_1 \approx -0.09625078 + 0.00749844 \cdot 1 \approx -0.08875234316421067$$

$$z_2 \approx -0.09625078 + 0.00749844 \cdot 2 \approx -0.08125390527368445$$

$$z_3 \approx -0.09625078 + 0.00749844 \cdot 3 \approx -0.07375546738315822$$

2. Sigmoids:

$$\hat{y} \approx [0.4778264673349967,\ 0.479697692439044,\ 0.481569487361292]$$

3. Errors:

$$e \approx [-0.4778264673,\ -0.4796976924,\ 0.5184305126]$$

4. Gradients:

$$g_{b_0} = \sum e_i \approx -0.4390936471353327$$

$$g_{b_1} = \sum e_i x_i \approx (-0.4778264673) \cdot 1 + (-0.4796976924) \cdot 2 + (0.5184305126) \cdot 3 \approx 0.118069685703039 44$$

5. Update:

$$b_0 \leftarrow -0.09625078105473689 + 0.1 \times (-0.4390936471353327) \approx -0.14016014576827016$$

$$b_1 \leftarrow 0.007498437890526222 + 0.1 \times 0.11806968570303944 \approx 0.019305406460830166$$

**After iteration 3:** $\boxed{b_0 \approx -0.1401601458,\ b_1 \approx 0.01930540646}$

# EXTRA

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

# Logistic Regression

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

## Quick Summary Formula Sheet

$$\hat{y}_i = \frac{1}{1+e^{-(b_0+b_1x_i)}}$$

$$\ell(b_0, b_1) = \sum_i [y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)]$$

$$b_j := b_j + \alpha \sum_i (y_i - \hat{y}_i)x_{ij}$$

# How does logistic regression works? (p2)

## 🎯 4. **Cost Function (Loss Function)**

To train the model, we minimize the **Binary Cross-Entropy Loss** (also called **Log Loss**):

Given $m$ training examples $\{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$, where $y^{(i)} \in \{0, 1\}$:

- **Cost for one sample:**

$$\mathcal{L}(h_w(x^{(i)}), y^{(i)}) = -\left[y^{(i)} \log(h_w(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_w(x^{(i)}))\right]$$

- **Cost over all samples:**

$$J(w, b) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(h_w(x^{(i)}), y^{(i)})$$

Cost function explained on next slide

# How does logistic regression works? (p2.1)

## 🎯 What is the **Cost Function** in Logistic Regression?

The cost function tells us **how well** our model's predictions match the actual labels. In logistic regression, we use a cost function called:

### ➤ Binary Cross-Entropy Loss

(Also known as **Log Loss**)

This is specifically designed for **binary classification problems**, where each output is either `0` or `1`.

## 🧪 Let's Start With Probabilities

**Suppose:**

- Your model predicts:
  $\hat{y} = P(y = 1 \mid x)$ — i.e., the probability that the output is 1 given input $x$
- The true label is $y \in \{0, 1\}$

We want our prediction $\hat{y}$ to be **close to the actual label** $y$.

## 📉 The Idea Behind Cross-Entropy

Cross-entropy comes from **information theory**, where it measures the distance between two probability distributions:

- One is the **true label** $y$
- The other is the **predicted probability** $\hat{y}$

## Binary Cross-Entropy Formula:

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

### 🔸 Case 1: True label $y = 1$

Then:

$$\mathcal{L} = -\left[1 \cdot \log(\hat{y}) + 0 \cdot \log(1 - \hat{y})\right] = -\log(\hat{y})$$

> If $\hat{y} = 0.9$, then loss = $-\log(0.9) = 0.105$ (small loss 👍)
> If $\hat{y} = 0.1$, then loss = $-\log(0.1) = 2.302$ (large loss 👎)

### 🔸 Case 2: True label $y = 0$

Then:

$$\mathcal{L} = -\left[0 \cdot \log(\hat{y}) + 1 \cdot \log(1 - \hat{y})\right] = -\log(1 - \hat{y})$$

> If $\hat{y} = 0.1$, then loss = $-\log(0.9) = 0.105$ (good)
> If $\hat{y} = 0.9$, then loss = $-\log(0.1) = 2.302$ (bad)

## 💡 Key Insight:

- The loss is **small** when the model is confident and **correct**.
- The loss is **large** when the model is confident and **wrong**.

## 🔄 Cost Over All Training Samples

If you have $m$ training examples:

$$J(w, b) = \frac{1}{m} \sum_{i=1}^{m} \left[-y^{(i)} \log(\hat{y}^{(i)}) - (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})\right]$$

Where:

- $\hat{y}^{(i)} = \sigma(w^T x^{(i)} + b)$ — prediction for example $i$
- $y^{(i)} \in \{0, 1\}$ — actual label

# How does logistic regression works? (p2.2)

## Why Logarithms in the Cost Function?

We use **logarithms** in the binary cross-entropy loss for two key reasons:

### 1. Penalize Wrong Predictions More

Logarithms grow rapidly negative as the predicted probability gets close to **0**, which helps to strongly **punish confident wrong predictions**.

**Example:**

If the true label is 1:

- Predicting $\hat{y} = 0.9$:

$$\text{Loss} = -\log(0.9) \approx 0.105 \quad \text{(small penalty)}$$

- Predicting $\hat{y} = 0.1$:

$$\text{Loss} = -\log(0.1) \approx 2.302 \quad \text{(huge penalty)}$$

This ensures that the model learns to assign **high probability to the correct class**.

### 2. Natural Result of Maximum Likelihood Estimation (MLE)

If we assume that the target $y \in \{0, 1\}$ is a Bernoulli random variable, then the **likelihood** of a single training sample is:

$$P(y \mid x) = \hat{y}^y \cdot (1 - \hat{y})^{1-y}$$

To simplify optimization, we take the **log-likelihood**:

$$\log P(y \mid x) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})$$

We **negate** this because we want to **minimize** (instead of maximizing) the function:

$$\text{Loss} = -\left[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})\right]$$

This becomes the **binary cross-entropy loss function** used in logistic regression.

| Reason for log function | Explanation |
|---|---|
| Punish wrong predictions | Log function penalizes low probabilities strongly |
| Derived from probability theory | It comes from the log of the Bernoulli likelihood in MLE |
| Smooth & differentiable | Great for optimization using gradient descent |

$$h_\theta(x) = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n)}}$$

This is the **general logistic regression hypothesis** for multivariate data.

## Decision Rule

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

Logistic regression uses **Binary Cross Entropy Loss:**

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

And optimized using **gradient descent.**