# Logistic Regression: Mathematical Notes

Logistic Regression is a fundamental classification algorithm used when the dependent variable is categorical, especially binary (e.g., 0 or 1, True or False, Yes or No). Despite its name, it's a classification model rather than a regression model in the traditional sense, as it outputs probabilities that are then mapped to classes.

We'll use a small, simplified dataset to illustrate the core concepts and calculations involved in training a Logistic Regression model using **Gradient Descent**. Unlike Linear Regression, Logistic Regression typically doesn't have a simple closed-form solution for its coefficients, requiring iterative optimization methods.

**Dataset: Student Exam Pass/Fail Prediction**

Suppose we want to predict whether a student passes an exam based on the number of hours they studied.

| Student ID | Hours Studied ($x_1$) | Passed Exam ($y$, 1=Yes, 0=No) |
|---|---|---|
| 1 | 0.5 | 0 |
| 2 | 1.5 | 0 |
| 3 | 2.5 | 1 |
| 4 | 3.0 | 1 |
| 5 | 1.0 | 0 |

**Goal:** Train a Logistic Regression model to predict the probability of a student passing the exam based on hours studied.

## I. Logistic Regression Model

**Hypothesis Function (Sigmoid Function):**

Logistic Regression uses the sigmoid (or logistic) function to map any real-valued number into a probability between 0 and 1. The linear combination of features and weights is first calculated:

$$z = \beta_0 + \beta_1 x_1$$

Then, this $z$ value is passed through the sigmoid function to get the predicted probability $\hat{p}$:

$$\hat{p} = h_{\boldsymbol{\beta}}(\mathbf{x}) = \frac{1}{1 + e^{-z}} = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1)}}$$

Here, $\mathbf{x}$ is the feature vector including the bias term ($[1, x_1]^T$) and $\boldsymbol{\beta}$ is the weight vector ($[\beta_0, \beta_1]^T$). So, $z = \boldsymbol{\beta}^T \mathbf{x}$.

The prediction for the class is typically made by comparing $\hat{p}$ to a threshold (e.g., 0.5). If $\hat{p} \geq 0.5$, predict class 1; otherwise, predict class 0.

**Cost Function (Log-Loss / Binary Cross-Entropy):**

Instead of Mean Squared Error, Logistic Regression uses a cost function suitable for probabilities. For a single training example $(x_i, y_i)$, the cost is:

$$\text{Cost}(h_{\boldsymbol{\beta}}(\mathbf{x}_i), y_i) = \begin{cases} -\log(h_{\boldsymbol{\beta}}(\mathbf{x}_i)) & \text{if } y_i = 1 \\ -\log(1 - h_{\boldsymbol{\beta}}(\mathbf{x}_i)) & \text{if } y_i = 0 \end{cases}$$

This can be written in a compact form for a single training example:

$$\text{Cost}(h_{\boldsymbol{\beta}}(\mathbf{x}_i), y_i) = -y_i \log(h_{\boldsymbol{\beta}}(\mathbf{x}_i)) - (1 - y_i) \log(1 - h_{\boldsymbol{\beta}}(\mathbf{x}_i))$$

The total cost function (average log-loss over all $n$ training examples) to be minimized is:

$$J(\boldsymbol{\beta}) = -\frac{1}{n} \sum_{i=1}^{n} [y_i \log(h_{\boldsymbol{\beta}}(\mathbf{x}_i)) + (1 - y_i) \log(1 - h_{\boldsymbol{\beta}}(\mathbf{x}_i))]$$

**Optimization Algorithm: Gradient Descent**

Since there's no simple closed-form solution for $\boldsymbol{\beta}$, we use an iterative optimization algorithm like Gradient Descent. The weights are updated iteratively by moving in the direction opposite to the gradient of the cost function.

The update rule for each parameter $\beta_j$ is:

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

where $\alpha$ is the learning rate.

The partial derivative of the cost function with respect to each $\beta_j$ is:

$$\frac{\partial}{\partial \beta_j} J(\boldsymbol{\beta}) = \frac{1}{n} \sum_{i=1}^{n} (h_{\boldsymbol{\beta}}(\mathbf{x}_i) - y_i) x_{ij}$$

For $\beta_0$ (intercept), $x_{i0}$ is always 1.

**Calculations for Logistic Regression using Gradient Descent:**

Let's apply Gradient Descent for a few iterations. **Dataset:**

| Student ID | $\mathbf{x_1}$ | y |
|---|---|---|
| 1 | 0.5 | 0 |
| 2 | 1.5 | 0 |
| 3 | 2.5 | 1 |
| 4 | 3.0 | 1 |
| 5 | 1.0 | 0 |

Number of training examples ($n$) = 5. We'll include a bias term $x_0 = 1$ for each student. Initial parameters (often initialized to 0 or small random values): Let's start with $\beta_0 = 0.0$ and $\beta_1 = 0.0$. Learning rate ($\alpha$) = 0.1 (a small value for demonstration).

**Iteration 1:**

**Step 1: Calculate $z_i$ and $\hat{p}_i$ for each example with current $\boldsymbol{\beta}$.** Current $\boldsymbol{\beta} = [0.0, 0.0]^T$.

$$\hat{p}_i = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_{i1})}}$$

- **Student 1** ($x_1 = 0.5, y = 0$)**:** $z_1 = 0.0 + 0.0(0.5) = 0.0$ $\hat{p}_1 = \frac{1}{1+e^{-0.0}} = \frac{1}{1+1} = 0.5$
- **Student 2** ($x_1 = 1.5, y = 0$)**:** $z_2 = 0.0 + 0.0(1.5) = 0.0$ $\hat{p}_2 = 0.5$
- **Student 3** ($x_1 = 2.5, y = 1$)**:** $z_3 = 0.0 + 0.0(2.5) = 0.0$ $\hat{p}_3 = 0.5$
- **Student 4** ($x_1 = 3.0, y = 1$)**:** $z_4 = 0.0 + 0.0(3.0) = 0.0$ $\hat{p}_4 = 0.5$
- **Student 5** ($x_1 = 1.0, y = 0$)**:** $z_5 = 0.0 + 0.0(1.0) = 0.0$ $\hat{p}_5 = 0.5$

**Step 2: Calculate gradients.**

$$\frac{\partial J}{\partial \beta_0} = \frac{1}{n} \sum_{i=1}^{n} (\hat{p}_i - y_i) x_{i0}$$

$$\frac{\partial J}{\partial \beta_1} = \frac{1}{n} \sum_{i=1}^{n} (\hat{p}_i - y_i) x_{i1}$$

- $(\hat{p}_1 - y_1)x_{10} = (0.5 - 0) \cdot 1 = 0.5$
- $(\hat{p}_2 - y_2)x_{20} = (0.5 - 0) \cdot 1 = 0.5$
- $(\hat{p}_3 - y_3)x_{30} = (0.5 - 1) \cdot 1 = -0.5$
- $(\hat{p}_4 - y_4)x_{40} = (0.5 - 1) \cdot 1 = -0.5$

- $(\hat{p}_5 - y_5)x_{50} = (0.5 - 0) \cdot 1 = 0.5$

$$\frac{\partial J}{\partial \beta_0} = \frac{1}{5}(0.5 + 0.5 - 0.5 - 0.5 + 0.5) = \frac{1}{5}(0.5) = 0.1$$

- $(\hat{p}_1 - y_1)x_{11} = (0.5 - 0) \cdot 0.5 = 0.25$
- $(\hat{p}_2 - y_2)x_{21} = (0.5 - 0) \cdot 1.5 = 0.75$
- $(\hat{p}_3 - y_3)x_{31} = (0.5 - 1) \cdot 2.5 = -1.25$
- $(\hat{p}_4 - y_4)x_{41} = (0.5 - 1) \cdot 3.0 = -1.50$
- $(\hat{p}_5 - y_5)x_{51} = (0.5 - 0) \cdot 1.0 = 0.50$

$$\frac{\partial J}{\partial \beta_1} = \frac{1}{5}(0.25 + 0.75 - 1.25 - 1.50 + 0.50) = \frac{1}{5}(-1.25) = -0.25$$

**Step 3: Update parameters.**

$$\beta_0 := 0.0 - 0.1 \cdot (0.1) = -0.01$$

$$\beta_1 := 0.0 - 0.1 \cdot (-0.25) = 0.025$$

New $\boldsymbol{\beta} = [-0.01, 0.025]^T$.

**Iteration 2:**

**Step 1: Calculate $z_i$ and $\hat{p}_i$ with new $\boldsymbol{\beta} = [-0.01, 0.025]^T$.**

$$\hat{p}_i = \frac{1}{1 + e^{-(-0.01 + 0.025 x_{i1})}}$$

- **Student 1** $(x_1 = 0.5, y = 0)$**:** $z_1 = -0.01 + 0.025(0.5) = 0.0025 \implies \hat{p}_1 = \frac{1}{1+e^{-0.0025}} \approx 0.5006$
- **Student 2** $(x_1 = 1.5, y = 0)$**:** $z_2 = -0.01 + 0.025(1.5) = 0.0275 \implies \hat{p}_2 = \frac{1}{1+e^{-0.0275}} \approx 0.5069$
- **Student 3** $(x_1 = 2.5, y = 1)$**:** $z_3 = -0.01 + 0.025(2.5) = 0.0525 \implies \hat{p}_3 = \frac{1}{1+e^{-0.0525}} \approx 0.5131$
- **Student 4** $(x_1 = 3.0, y = 1)$**:** $z_4 = -0.01 + 0.025(3.0) = 0.0650 \implies \hat{p}_4 = \frac{1}{1+e^{-0.0650}} \approx 0.5162$
- **Student 5** $(x_1 = 1.0, y = 0)$**:** $z_5 = -0.01 + 0.025(1.0) = 0.0150 \implies \hat{p}_5 = \frac{1}{1+e^{-0.0150}} \approx 0.5037$

**Step 2: Calculate gradients.**

- $(\hat{p}_1 - y_1)x_{10} = (0.5006 - 0) \cdot 1 = 0.5006$
- $(\hat{p}_2 - y_2)x_{20} = (0.5069 - 0) \cdot 1 = 0.5069$
- $(\hat{p}_3 - y_3)x_{30} = (0.5131 - 1) \cdot 1 = -0.4869$
- $(\hat{p}_4 - y_4)x_{40} = (0.5162 - 1) \cdot 1 = -0.4838$
- $(\hat{p}_5 - y_5)x_{50} = (0.5037 - 0) \cdot 1 = 0.5037$

$$\frac{\partial J}{\partial \beta_0} = \frac{1}{5}(0.5006 + 0.5069 - 0.4869 - 0.4838 + 0.5037) = \frac{1}{5}(0.5405) = 0.1081$$

- $(\hat{p}_1 - y_1)x_{11} = (0.5006 - 0) \cdot 0.5 = 0.2503$
- $(\hat{p}_2 - y_2)x_{21} = (0.5069 - 0) \cdot 1.5 = 0.7604$
- $(\hat{p}_3 - y_3)x_{31} = (0.5131 - 1) \cdot 2.5 = -1.2173$
- $(\hat{p}_4 - y_4)x_{41} = (0.5162 - 1) \cdot 3.0 = -1.4514$
- $(\hat{p}_5 - y_5)x_{51} = (0.5037 - 0) \cdot 1.0 = 0.5037$

$$\frac{\partial J}{\partial \beta_1} = \frac{1}{5}(0.2503 + 0.7604 - 1.2173 - 1.4514 + 0.5037) = \frac{1}{5}(-1.1543) = -0.2309$$

**Step 3: Update parameters.**

$$\beta_0 := -0.01 - 0.1 \cdot (0.1081) = -0.01 - 0.01081 = -0.02081$$

$$\beta_1 := 0.025 - 0.1 \cdot (-0.2309) = 0.025 + 0.02309 = 0.04809$$

New $\boldsymbol{\beta} = [-0.02081, 0.04809]^T$.

This iterative process continues until the parameters converge (i.e., the changes in $\beta$ values become very small) or a maximum number of iterations is reached.

## III. Multiple Logistic Regression (Vectorized Form)

For multiple features, the equations naturally extend using matrix/vector notation.

**Hypothesis Function (Vectorized):**

For a single training example $\mathbf{x} = [x_0, x_1, \ldots, x_p]^T$ (where $x_0 = 1$ for the bias term):

$$z = \boldsymbol{\beta}^T \mathbf{x}$$

$$\hat{p} = h_{\boldsymbol{\beta}}(\mathbf{x}) = \frac{1}{1 + e^{-\boldsymbol{\beta}^T \mathbf{x}}}$$

For all $n$ training examples, if $\mathbf{X}$ is the $n \times (p+1)$ design matrix and $\boldsymbol{\beta}$ is the $(p+1) \times 1$ vector of coefficients:

$$\mathbf{z} = \mathbf{X}\boldsymbol{\beta}$$

$$\hat{\mathbf{p}} = \frac{1}{1 + e^{-\mathbf{X}\boldsymbol{\beta}}}$$

where the division by 1 and exponential are element-wise.

**Cost Function (Vectorized):**

$$J(\boldsymbol{\beta}) = -\frac{1}{n}[\mathbf{y}^T \log(\hat{\mathbf{p}}) + (\mathbf{1} - \mathbf{y})^T \log(\mathbf{1} - \hat{\mathbf{p}})]$$

where $\mathbf{1}$ is a vector of ones, and log is element-wise.

**Gradient (Vectorized):**

The gradient of the cost function with respect to $\boldsymbol{\beta}$ is:

$$\nabla J(\boldsymbol{\beta}) = \frac{1}{n}\mathbf{X}^T(\hat{\mathbf{p}} - \mathbf{y})$$

**Update Rule (Vectorized):**

$$\boldsymbol{\beta} := \boldsymbol{\beta} - \alpha\frac{1}{n}\mathbf{X}^T(\hat{\mathbf{p}} - \mathbf{y})$$

## Discussion for Graduate Students:

- **Probabilistic Interpretation:** Emphasize that Logistic Regression directly models $P(Y = 1|\mathbf{X})$, providing a probabilistic output which is highly interpretable. This is valuable for tasks requiring confidence scores, not just binary classifications.
- **Decision Boundary:** Explain how the linear combination $z = \boldsymbol{\beta}^T \mathbf{x}$ defines a linear decision boundary in the feature space ($z = 0 \implies \hat{p} = 0.5$). For our example, $-0.02081 + 0.04809x_1 = 0$, implies $x_1 \approx 0.02081/0.04809 \approx 0.43$. This means students studying more than approximately 0.43 hours are predicted to pass (once the model converges).
- **Regularization (L1/L2):** Just like Linear Regression, Logistic Regression can suffer from overfitting, especially with many features. Discuss L1 (Lasso) and L2 (Ridge) regularization by adding penalty terms to the cost function. This leads to penalized maximum likelihood estimation.
  - **L2 (Ridge):** $J(\boldsymbol{\beta}) + \frac{\lambda}{2n}\sum_{j=1}^{p}\beta_j^2$
  - **L1 (Lasso):** $J(\boldsymbol{\beta}) + \frac{\lambda}{n}\sum_{j=1}^{p}|\beta_j|$

  (Note: $\beta_0$ is typically not regularized).
- **Maximum Likelihood Estimation (MLE):** Connect the log-loss cost function to the principle of Maximum Likelihood Estimation. Minimizing the negative log-likelihood is equivalent to maximizing the likelihood of observing the training data given the model parameters.

- **Multiclass Logistic Regression (Softmax Regression):** Briefly introduce how Logistic Regression extends to multiclass classification problems using the softmax function (also known as Multinomial Logistic Regression).
- **Evaluation Metrics:** Discuss appropriate evaluation metrics for classification tasks: Accuracy, Precision, Recall, F1-score, ROC Curve, AUC (Area Under the Curve), Log-Loss (which is also the cost function).
- **Relationship to Neural Networks:** Position Logistic Regression as a single-neuron neural network without a hidden layer, with a sigmoid activation function. This provides a natural bridge to deeper learning concepts.
- **Optimization Challenges:** Discuss challenges with Gradient Descent, such as choosing an appropriate learning rate, getting stuck in local minima (though not an issue for convex Logistic Regression loss), and computational cost for very large datasets (leading to Stochastic Gradient Descent or Mini-Batch Gradient Descent).
- **Interpreting Coefficients (Log-Odds):** A one-unit increase in $x_j$ increases the log-odds of the positive class by $\beta_j$. So, $e^{\beta_j}$ is the odds ratio. This is a crucial interpretive point for graduate students.

Prepared By:

**Md. Atikuzzaman**

Lecturer

Department of Computer Science and Engineering

Green University of Bangladesh

Email: atik@cse.green.edu.bd