Logistic Regression - Loss Function - Maximum Likelihood | Binary Cross Entropy

1. Introduction

At this point, the focus is on why we need a *loss function* in **Logistic Regression**.

- In **Linear Regression**, we typically use **Mean Squared Error (MSE)** to measure how far predictions are from actual values.
- But Logistic Regression deals with **classification problems** (e.g., "yes" or "no", "spam" or "not spam"), so predictions are probabilities between **0 and 1**.
- This means we need a loss function that measures **how well our probability predictions match the actual labels** in a classification setting.

2. The Problem

The issue with using something like MSE here is:

- Probabilities are **non-linear** with respect to the model parameters when using the **sigmoid function**.
- MSE doesn't handle probability predictions effectively in classification, and it can lead to **slow learning** or getting stuck in local minima during optimization.

We need:

- A method to find the *best set of parameters* (weights) that maximize how well the model explains the observed data.
- A way to quantify "how likely" the observed labels are, given the predicted probabilities from the model.

This is where **Maximum Likelihood Estimation (MLE)** comes into play.

3. Maximum Likelihood

What is Maximum Likelihood?

- It’s a statistical method used to estimate model parameters (weights) by **maximizing the probability of observing the actual training data**.
- In Logistic Regression, it means choosing weights such that the predicted probabilities are as close as possible to the actual binary labels.

Step-by-step reasoning:

1. **Predicted probability for a single example:**
   - For label `y = 1`, probability is:

     $$P(y = 1|x) = \sigma(w \cdot x)$$

     where σ is the sigmoid function.
   - For label `y = 0`, probability is:

     $$P(y = 0|x) = 1 - \sigma(w \cdot x)$$

2. **Combine into one formula:**

   We can write both cases together:

   $$P(y|x) = [\sigma(w \cdot x)]^y \cdot [1 - \sigma(w \cdot x)]^{(1-y)}$$

   - If $y = 1$, the second term becomes 1.
   - If $y = 0$, the first term becomes 1.

3. **Likelihood for the whole dataset:**

   Assuming data points are independent:

   $$L(w) = \prod_{i=1}^{n} [\sigma(w \cdot x_i)]^{y_i} \cdot [1 - \sigma(w \cdot x_i)]^{(1-y_i)}$$

   - This is called the **likelihood function**.

4. **Log-Likelihood:**
   - Multiplying many probabilities can lead to very small numbers, so we take the **log** (logarithm) to avoid numerical underflow and make calculations easier:

     $$\ell(w) = \sum_{i=1}^{n} \left[y_i \log(\sigma(w \cdot x_i)) + (1 - y_i) \log(1 - \sigma(w \cdot x_i)) \right]$$

   - This is called the **log-likelihood function**.

5. **Goal:**
   - **Maximize** this log-likelihood to find the best weights.
   - Maximizing log-likelihood = finding weights that make our predicted probabilities match the actual outcomes as closely as possible.

➢ Binary Cross Entropy Connection

When we **maximize** the log-likelihood, mathematically, it is equivalent to **minimizing** something called **Binary Cross Entropy Loss**:

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

- Here, $\hat{y}_i = \sigma(w \cdot x_i)$ is the predicted probability.
- If $y_i = 1$, the first term dominates (we penalize low predicted probability for class 1).
- If $y_i = 0$, the second term dominates (we penalize high predicted probability for class 0).

Why it works well:

- It naturally fits classification where outputs are probabilities.
- It heavily penalizes confident but wrong predictions.
- It's smooth and convex for logistic regression, so gradient descent can find the optimal weights effectively.

➢ Summary:

We want our model to output probabilities close to the true labels.

- **Maximum Likelihood** says: choose weights that make the actual observed labels most probable.
- This leads to the **Binary Cross Entropy Loss**, which punishes wrong predictions more if the model is overconfident.
- Instead of guessing weights randomly, we use math to pick the ones that make our data look as likely as possible under the model.
