Logistic Regression from Scratch

Bhanu Prasanna Koppolu

Table of contents

> **Note**
>
> Full implementation available at GitHub - ML from Scratch

Logistic Regression

Why No Closed-Form Solution?

There is no closed-form solution for Logistic regression. The MLE doesn't exist like it does for Linear Regression.

Unlike Linear regression, which has a linear solution, Logistic regression makes use of the **sigmoid function** which makes it non-linear, and can't be directly derived in the closed form solution.

Gradient Descent for Logistic Regression

So, it is the same as Linear Regression Gradient Descent. The important thing is that now we have an activation function **sigmoid**. In linear regression we did not have an activation function (Identity Activation Function), but it is not considered as an activation function.

Now, let's see, the equation remains the same:

$$y = Xw + b$$

But now for $\hat{y}$ we have to do the following:

$$\hat{y} = \sigma(z)$$

$$z = Xw + b$$

Where σ is the sigmoid function:

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

Loss Function

The loss function has changed to **Binary Cross Entropy** loss taken from the negative log likelihood of Bernoulli:

$$\ell(\hat{y}, y) = -(y \cdot \log(\hat{y}) + (1 - y) \cdot \log(1 - \hat{y}))$$

Gradient Computation

So, the optimization looks like this. We are again trying to find $\frac{\partial \ell}{\partial w}$ and $\frac{\partial \ell}{\partial b}$:

$$\frac{\partial \ell}{\partial w} = \frac{\partial \ell}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w} = (\hat{y} - y) \cdot X$$

$$\frac{\partial \ell}{\partial b} = \frac{\partial \ell}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial b} = (\hat{y} - y)$$

$$\frac{\partial J}{\partial w} = \frac{1}{m} X^T (\hat{y} - y)$$

$$\frac{\partial J}{\partial b} = \frac{1}{m} (\hat{y} - y)$$

```python
def logistic_regression(X, y, learning_rate=0.01, iterations=1000):
    w = np.random.rand(X.shape[1])
    b = np.random.rand(1)
    m = X.shape[0]

    cost_list = []

    for i in range(iterations):
        z = (X @ w + b)
        y_hat = 1 / (1 + np.power(np.e, -z))

        dj_dw = (1/m) * (X.T @ (y_hat - y))
        dj_db = (1/m) * np.sum(y_hat - y)

        w = w - (learning_rate * dj_dw)
        b = b - (learning_rate * dj_db)

        cost = -1 * ((y * np.log(y_hat + 1e-15)) + ((1 - y) * np.log(1 - y_hat + 1e-15)))
```

```
        cost = (1 / m) * np.sum(cost)
        cost_list.append(cost)

    return w, b, cost_list
```