Logistic Regression using Gradient Descent - Labwork 3

Do Thanh Dat - 2440059

May 7, 2025

1 Introduction

Logistic Regression is a fundamental algorithm in Machine Learning used for binary classification tasks. Unlike Linear Regression, its goal is not to predict a continuous value but to estimate the probability that a given input belongs to class 1. In this Lab-work, I implemented Logistic Regression from scratch using the gradient descent method, applied to a dataset consisting of two features (x_1, x_2) and a binary label $y \in \{0, 1\}$.

2 Method

The model estimates the probability as follows:

$$\hat{y}_i = \sigma(w_1 x_1^{(i)} + w_2 x_2^{(i)} + w_0) = \frac{1}{1 + e^{-(w_1 x_1 + w_2 x_2 + w_0)}}$$

where $\sigma(z)$ is the sigmoid function.

The binary cross-entropy loss function is used:

$$L_i = -\left[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)\right]$$

To minimize this loss, gradient descent is applied by updating the weights based on the gradients (these formulas are the shorten version):

$$\frac{\partial J}{\partial w_0} = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i), \quad \frac{\partial J}{\partial w_1} = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i) x_1^{(i)}, \quad \frac{\partial J}{\partial w_2} = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i) x_2^{(i)}$$

3 Code Implementation

```
def sigmoid_func(u):
    return 1/(1 + math.exp(-u))

def load_data_from_csv(file_path):
```

```
    x1, x2, y = [], [], []
    with open(file_path, 'r') as f:
        reader = csv.reader(f)
        next(reader)
        for row in reader:
            x1.append(float(row[0]))
            x2.append(float(row[1]))
            y.append(int(row[2]))
    return x1, x2, y

def logistic_regression(x1, x2, y, r: float, iters: int):
  N = len(y)
  w0 = 0
  w1 = 1
  w2 = 2
  for i in range(iters):
    total_loss = 0
    for j in range(N):
      y_hat = w1*x1[j] + w2*x2[j]+ w0
      sig_y_hat = sigmoid_func(y_hat)
      neg_sig_y_hat = sigmoid_func(-y_hat)

      dw0 = 1 - y[j] - neg_sig_y_hat
      dw1 = -y[j] * x1[j] + x1[j] * (1 - neg_sig_y_hat)
      dw2 = -y[j] * x2[j] + x2[j] * (1 - neg_sig_y_hat)

      loss_j = -y[j]*y_hat + math.log(1 + math.exp(y_hat))
      total_loss += loss_j

      w0 -= r * dw0
      w1 -= r * dw1
      w2 -= r * dw2

      avg_loss = total_loss / N

    print(f"{i} {w0} {w1} {w2} {avg_loss}")

  return w0, w1,w2

x1, x2, y = load_data_from_csv(file_path)

for lr in [0.1,0.001,0.0001]:
  w0, w1,w2 = logistic_regression(x1,x2, y, lr, iters=10)
  print(f"LR model {lr}: y = {w1} * x1  {w2} * x2  {w0}")
```

4 Results

The model was tested with different learning rates. Below is an example with $r = 0.0001$:

Step	w_0	w_1	w_2	**Loss**
0	0.0059	1.2365	1.9712	0.6852
1	0.0111	1.4642	1.9581	0.6751
2	0.0159	1.6472	1.9458	0.6673
3	0.0203	1.7964	1.9343	0.6611
4	0.0243	1.9194	1.9235	0.6558

Table 1: Training process with learning rate $r = 0.0001$ (sample steps)

5 Comment

- Logistic regression performed well on the binary classification task, correctly minimizing the binary cross-entropy loss.
- As in linear regression, the learning rate r is crucial. A large r (e.g., 0.01) caused divergence, while a smaller rate such as $r = 0.0001$ led to stable convergence.
- We observed that the gradient update rule using the sigmoid derivative was effective, and the decision boundary between the classes can be visualized using matplotlib.

6 Conclusion

This Lab-work showed how to implement logistic regression using gradient descent from scratch. I learned how to calculate the sigmoid-based prediction, apply cross-entropy loss, and update weights iteratively. The results confirmed that the method works correctly with the right learning rate and number of iterations.