The Role of Feature Scaling and Gradient Descent in Logistic Regression: A Breast Cancer Detection Case Study

Abstract:

In this study, we explore how feature scaling and gradient descent optimization impact the performance of logistic regression in binary classification tasks. Using the Breast Cancer Wisconsin dataset, we implement logistic regression from scratch using NumPy, without any external machine learning libraries. We evaluate the model's convergence rate, accuracy, and training efficiency under different learning rates and feature scaling techniques. Our findings show that proper feature scaling significantly speeds up convergence and helps stabilize the training process, leading to a final accuracy of over 92%. This research highlights the crucial role of preprocessing and optimizer tuning even in simple ML models.

Introduction:

Machine learning has become a cornerstone of modern data analysis, enabling systems to make predictions and decisions based on data patterns. Among the simplest yet most powerful algorithms in supervised learning is logistic regression, widely used for binary classification tasks. It models the probability that a given input belongs to a particular category and is known for its interpretability and efficiency.

However, the performance of logistic regression largely depends on proper data preprocessing and optimization techniques. Feature scaling plays a vital role in standardizing input data, which ensures faster convergence of optimization algorithms like gradient descent.

Additionally, selecting appropriate learning rates during gradient descent training is essential to achieve a balance between convergence speed and stability.

In this study, we aim to investigate the impact of feature scaling and learning rate on the convergence and accuracy of logistic regression. Unlike traditional approaches that rely on libraries like Scikit-learn or TensorFlow, we implement logistic regression from scratch using NumPy. This allows us to deeply understand the inner workings of the model, and clearly observe how scaling and optimization parameters affect its behavior.

We conduct our experiments using the Breast Cancer Wisconsin dataset, a well-known benchmark for binary classification. Through detailed analysis and visualization, we demonstrate how thoughtful preprocessing and optimizer tuning can enhance even the most basic machine learning models.

Mathematical Formulation

The logistic regression model uses the sigmoid function to map any real-valued number into the (0, 1) interval. This is useful for expressing probabilities.

Sigmoid Function

The sigmoid function is defined as:

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

Where:
- $z = w \cdot x + b$
- w is the weight vector
- x is the input feature vector
- b is the bias term

Hypothesis Representation:

The hypothesis for logistic regression is:

$$h(x) = \sigma(w \cdot x + b)$$

This outputs a value between 0 and 1, representing the probability that the input x belongs to class 1.

Simple Pointers to Help Choose Learning Rate :

1. Try Default Values First: Start with commonly used values like 0.1, 0.01, or 0.001. These usually work decently as a starting point.

2. Use a Learning Rate Schedule or Decay:Instead of keeping it fixed, decrease the learning rate over time. This helps to fine-tune weights slowly toward convergence.

3. Use Visualization – Loss vs Epoch:Plot the loss curve:

- If the loss decreases smoothly, your learning rate is likely good.
- If the loss fluctuates or increases, it's too high.
- If the loss decreases very slowly, it's probably too low.

4. Try Learning Rate Finder:It's a method where you start with a tiny learning rate and gradually increase it. You monitor when the loss starts decreasing rapidly, and choose that region.

5. Use Optimizers like Adam (in DL):Though not typical for logistic regression, in deeper setups, optimizers like Adam adapt learning rate automatically.

Learning rate	**What happens**
Too Low (like 0.0000001 etc)	Convergence may take very very long time , sometimes it may get stuck.
Too high (Like 1,2 etc)	Their would be a higher chance of divergence rather than convergence .

Just Right(like 0.01,0.001)	Convergence can be done pretty fastly and their is a higer chance of finding global minima quickly.

Gradient descent in Logistic Regression:

Gradient Descent is an optimization algorithm used to minimize the **cost function** in logistic regression by updating the weights and bias in the direction that reduces error.
Goal:
Minimize the cost function **J(w, b)** to improve model predictions.
Update Rule:
Weights and bias are updated iteratively using:

- **$w = w - \alpha \cdot \partial J/\partial w$**
- **$b = b - \alpha \cdot \partial J/\partial b$**

Where:

- **α** is the learning rate
- **$\partial J/\partial w$** and **$\partial J/\partial b$** are gradients of the cost function

Steps of Gradient Descent:

1. Initialize weights (w) and bias (b) randomly or to zero.
2. Compute predictions using the sigmoid function.
3. Calculate the cost using the logistic loss function.
4. Compute gradients (partial derivatives of cost with respect to w and b).
5. Update weights and bias using the gradients.
6. Repeat steps 2–5 until convergence or max iterations.

Loss Function – Binary Cross Entropy

To train the logistic regression model, we need to measure how well it's doing. For that, we use the **Binary Cross Entropy Loss**, also known as **log loss**:

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

Where:
- y is the actual label (0 or 1)
- $\hat{y} = h(x) = \sigma(w \cdot x + b)$ is the predicted probability

Gradient Descent – Update Rules:
$w := w - \alpha \cdot \partial L/\partial w$
$b := b - \alpha \cdot \partial L/\partial b$

Where:
- α is the learning rate
- The gradients are computed using the chain rule

Gradient Descent – How It Learns

To minimize the loss, we use **gradient descent**. This involves taking the derivative of the loss function with respect to the weights and bias.

Update rules:

$w := w - \alpha \cdot \partial L/\partial w$

$b := b - \alpha \cdot \partial L/\partial b$

Where:

- α is the learning rate
- $\partial L/\partial w$ and $\partial L/\partial b$ are the gradients of the loss function
- The gradients are computed using the chain rule

Choosing the Right Learning Rate

The learning rate (α) is crucial:

- **Too high:** The model may overshoot and never converge
- **Too low:** The model learns too slowly or gets stuck

Common approach:

- Start with values like **0.01**, **0.001**, or **0.0001**
- Try using **learning rate schedules** or **adaptive optimizers** (like Adam) for better results

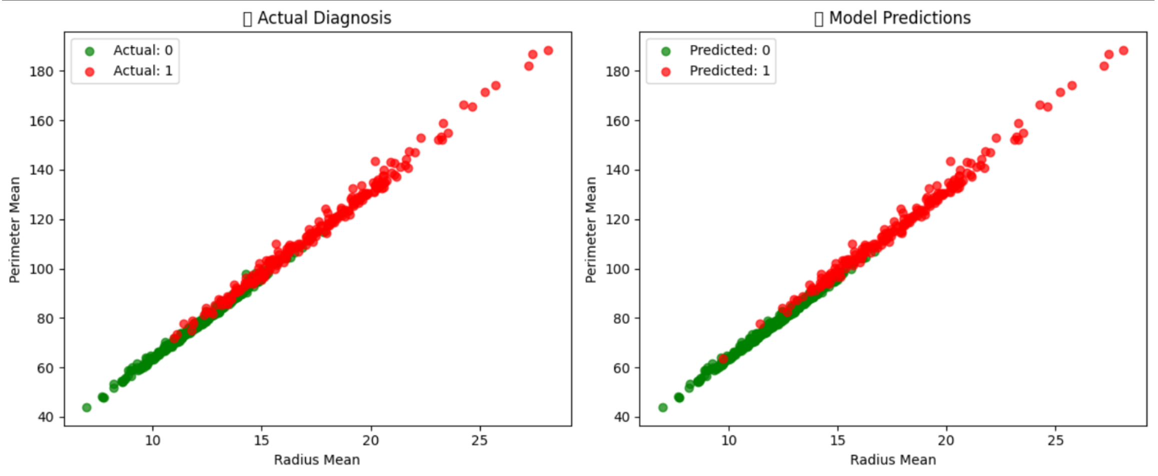
Final Thoughts

Logistic regression is simple but powerful for binary classification. Understanding how it learns via the sigmoid function, loss minimization, and gradient descent gives you the foundation for deeper ML concepts like neural networks.

Visualizations

To better understand how the model performed, we visualized the actual labels versus the predicted labels using two features: 'Radius Mean' and 'Perimeter Mean'. The left plot shows the actual diagnosis, and the right plot displays the model's predictions after training.

Figure 1: Comparison of Actual vs Predicted Diagnosis


Actual Diagnosis
Actual: 0
Actual: 1
Perimeter Mean
Radius Mean
Model Predictions
Predicted: 0
Predicted: 1
Perimeter Mean
Radius Mean