Linear Regression Model Analysis and Comparison

Your Name

May 19, 2025

1 Introduction

This report presents a comparative analysis of three different implementations of linear regression on the California housing dataset:

1. Loop-based Gradient Descent
2. Vectorized Gradient Descent using NumPy
3. Scikit-learn's `LinearRegression`

The models are assessed based on convergence behavior, training time, and regression metrics like MAE, RMSE, and R^2.

2 Data Preprocessing

1.

3 Mathematical Foundations

3.1 Linear Regression Model

Given training data with m samples and n features, the linear model predicts the output $\hat{y}$ as:

$$\hat{y}^{(i)} = \theta_0 + \theta_1 x_1^{(i)} + \cdots + \theta_n x_n^{(i)} = \boldsymbol{x}^{(i)} \cdot \boldsymbol{\theta}$$

where:

- $\boldsymbol{x}^{(i)}$ is the feature vector of the i-th sample.
- $\boldsymbol{\theta}$ is the vector of parameters (weights).

3.2 Cost Function (Mean Squared Error)

The objective is to minimize the cost function:

$$J(\boldsymbol{\theta}) = \frac{1}{2m} \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)} \right)^2$$

where $y^{(i)}$ is the true output and $\hat{y}^{(i)}$ is the predicted output.

3.3 Gradient Descent

To minimize $J(\boldsymbol{\theta})$, we update weights using:

$$\theta_j := \theta_j - \alpha \cdot \frac{\partial J}{\partial \theta_j}$$

where α is the learning rate and the partial derivative is:

$$\frac{\partial J}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)} \right) x_j^{(i)}$$

In vectorized form:

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \alpha \cdot \frac{1}{m} \boldsymbol{X}^T (\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y})$$

3.4 Normal Equation (Used by Scikit-learn)

The closed-form solution is:

$$\boldsymbol{\theta} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{y}$$

This is numerically efficient when n (number of features) is small.

4 Regression Metrics

4.1 Mean Absolute Error (MAE)

$$\text{MAE} = \frac{1}{m} \sum_{i=1}^{m} \left| y^{(i)} - \hat{y}^{(i)} \right|$$

4.2 Root Mean Squared Error (RMSE)

$$\text{RMSE} = \sqrt{\frac{1}{m} \sum_{i=1}^{m} \left(y^{(i)} - \hat{y}^{(i)} \right)^2}$$

4.3 R-squared Score

$$R^2 = 1 - \frac{\sum_{i=1}^{m} (y^{(i)} - \hat{y}^{(i)})^2}{\sum_{i=1}^{m} (y^{(i)} - \bar{y})^2}$$

where $\bar{y}$ is the mean of all target values.

5 Performance Results

Method	**MAE**	**RMSE**	R^2 **Score**
Loop-Based GD	53192.05	72545.69	0.6152
Vectorized GD	53192.05	72545.69	0.6152
Scikit-learn LR	51372.67	70156.12	0.6401

Table 1: Performance Comparison Across Models

6 Observations and Discussion

6.1 Convergence Times and Accuracies

- **Loop-based GD** took longer due to explicit iteration and was computationally expensive.
- **Vectorized GD** achieved identical accuracy but was significantly faster due to matrix operations.
- **Scikit-learn** provided the best performance and accuracy using optimized internal solvers.

6.2 Causes for Differences

- **Vectorization** reduces overhead from Python loops, enabling faster convergence.
- **Solver Type:** Scikit-learn uses closed-form solutions or QR/SVD decompositions which are more stable.
- **Initialization and Learning Rate:** Poor choices in θ_0 or α can prevent convergence in gradient descent.

6.3 Scalability and Efficiency

- Loop-based GD is impractical for large datasets.
- Vectorized GD scales well for large m and n.
- Scikit-learn is efficient for most use cases but may not scale to massive datasets unless using online variants like `SGDRegressor`.

6.4 Effect of Learning Rate and Initialization

- Choosing a small learning rate can slow down convergence significantly.
- Large learning rates might cause oscillation or divergence.
- Initialization (e.g., all zeros or random) can influence the number of epochs needed.

7 Conclusion

This analysis highlights that while all three methods can produce reliable linear models, their usability depends on the context. Gradient descent teaches foundational concepts but is less practical. Scikit-learn offers the best combination of accuracy, efficiency, and ease of use for most applications.