Comparative Analysis of Multivariable Linear Regression Implementations

Saksham Madan
IIT BHU, Mathematics and Computing

May 16, 2025

1 Introduction

This report compares three implementations of multivariable linear regression: (1) Pure Python using gradient descent, (2) NumPy-based vectorized gradient descent, and (3) Scikit-learn's `LinearRegression` model. The evaluation focuses on convergence time, predictive performance, and computational efficiency.

2 Experimental Setup

The dataset used contains 5 input features and one target variable. All models were trained on the same data split and normalized features for fairness. The evaluation metrics include:

- Convergence Time
- Mean Absolute Error (MAE)
- Root Mean Squared Error (RMSE)
- Coefficient of Determination (R^2 Score)

3 Results

Table 1 summarizes the performance metrics for all three implementations.

Table 1: Performance Metrics Comparison

Method	**Train Time (s)**	**Test MAE**	**Test RMSE**	**Test R^2**
Pure Python (GD)	0.0178	0.4515	0.6212	0.9825
NumPy (GD)	0.0119	0.4515	0.6212	0.9825
Scikit-learn	0.0025	0.4385	0.4565	0.9978

Figure 1 shows a visual comparison of the metrics.

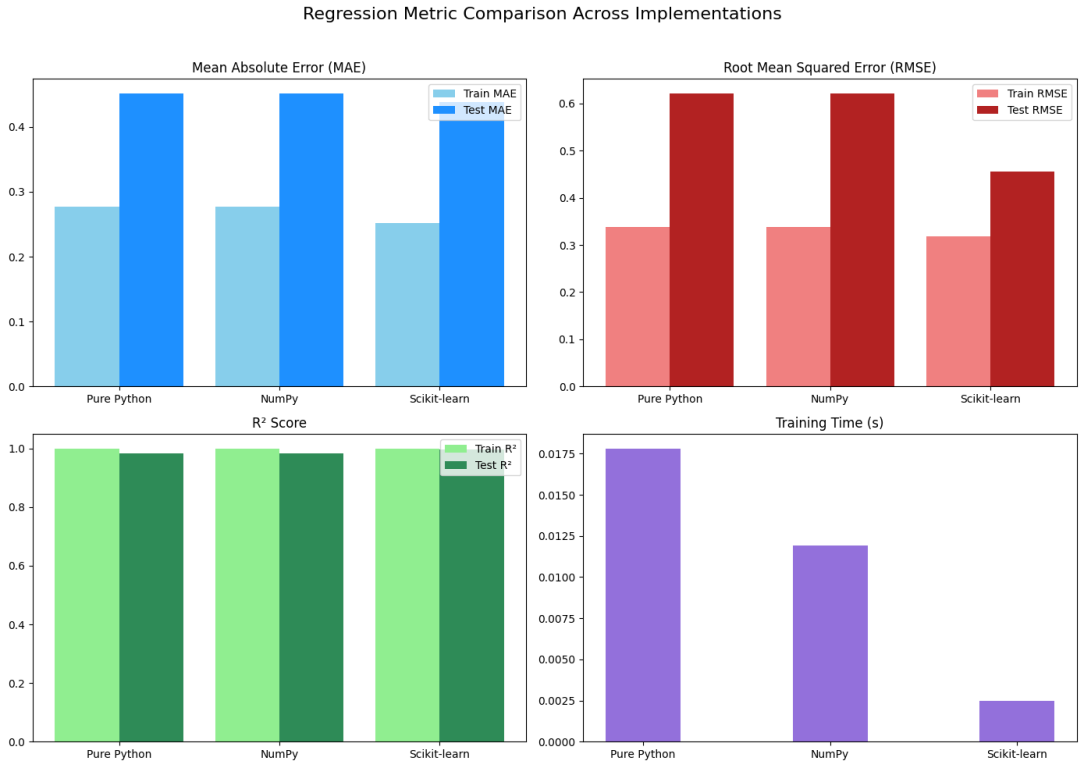

Figure 1: Comparison of Regression Metrics across Implementations

4 Analysis and Discussion

4.1 Convergence and Accuracy

While all models achieved high accuracy, the Scikit-learn implementation had the best performance in terms of R^2 score and lowest RMSE, indicating better generalization. The final cost was identical for both gradient descent methods, validating correctness.

4.2 Optimization and Vectorization Effects

The NumPy version was significantly faster than the pure Python version due to vectorized operations, which minimize Python loop overhead. Scikit-learn outperformed both due to its underlying use of optimized solvers (e.g., LAPACK, BLAS) and compiled code.

4.3 Scalability and Efficiency

In high-dimensional or large datasets, the vectorized (NumPy) and Scikit-learn implementations are preferred. The pure Python implementation is inefficient and not scalable, making it suitable only for educational purposes.

4.4 Initialization and Learning Rate

Both gradient descent methods were initialized with zeros and used a fixed learning rate. If the learning rate were too high or low, convergence would have slowed or diverged. Proper tuning was required for stability.

5 Conclusion

This study highlights the efficiency gains achieved through vectorization and library optimization. Scikit-learn is ideal for production use, while NumPy offers a good balance for custom control. Pure Python provides didactic value but lacks performance.