CSoC IG Assignment Report

Comparative Study of Multivariable Linear Regression Implementations

Vivek Garg
Roll Number: 24075096

May 2025

Introduction

This report presents three implementations of multivariable linear regression on the California Housing Price Dataset:

- **Part 1**: Pure Python implementation with gradient descent.
- **Part 2**: Optimized NumPy vectorized implementation.
- **Part 3**: scikit-learn's `LinearRegression`.

We compare convergence speed, predictive accuracy (MAE, RMSE, R^2), and training time.

Dataset and Preprocessing

The dataset contains numerical features and one categorical feature, `ocean_proximity`, which was dropped. Remaining columns plus the target `median_house_value` were converted to float. All features were normalized using Z-score normalization. In Parts 1–3, the target was also Z-scored before training; for interpretability, MAE and RMSE are reported both in normalized units and rescaled to dollars.

The data was split 80%/20% into training and validation sets.

Part 1: Pure Python Implementation

Implementation Details

- Hypothesis: $\hat{y} = \mathbf{w}^\top \mathbf{x} + b$
- Loss: $J = \frac{1}{2m} \sum (\hat{y} - y)^2$
- Optimization: Gradient descent, learning rate $\alpha = 0.01$, 1000 epochs.
- RNG: `random.shuffle()` (no seed).

Convergence Plot

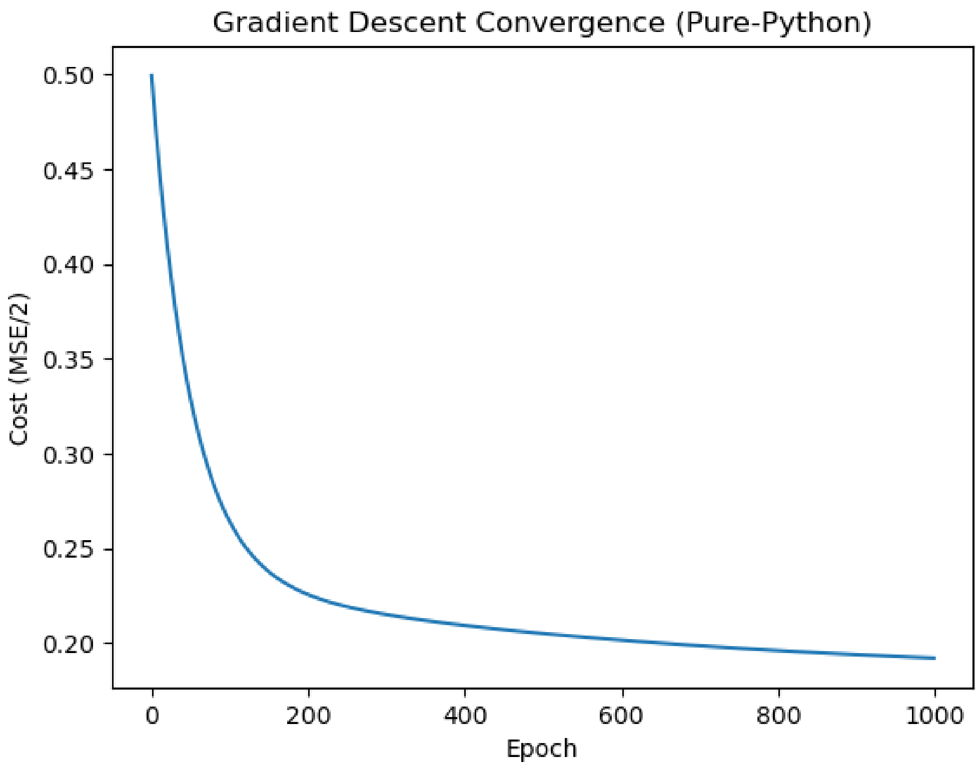

Figure 1: Cost vs. Epochs, Pure-Python Implementation

Results

- **Training time**: 45.584 seconds
- **Final weights:** [-0.3714, -0.4169, 0.1742, -0.0456, 0.1876, -0.2717, 0.1918, 0.6828]
- **Final bias:** -0.0043

Evaluation (Validation Set)

- MAE (norm): 0.4646
- RMSE (norm): 0.6498
- R^2: 0.6015
- MAE (dollars): 0.4646 $\times \sigma_y$
- RMSE (dollars): 0.6498 $\times \sigma_y$

Part 2: Optimized NumPy Implementation

Implementation Details

Vectorized operations with NumPy arrays for hypothesis, cost, and gradients. Same hyper-parameters and data split, RNG seeded (`np.random.seed(42)`).

Convergence Comparison

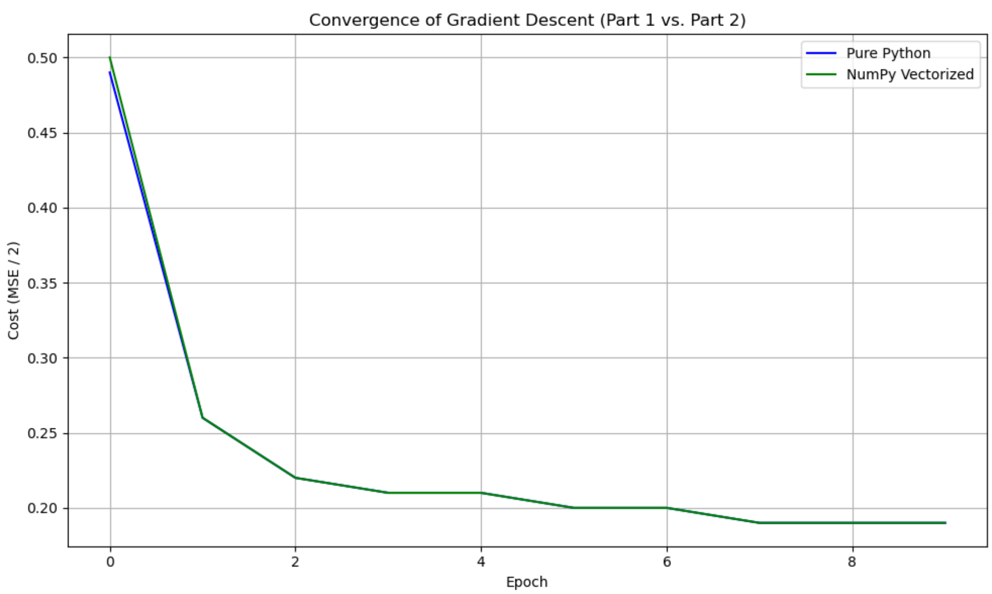

Figure 2: Cost vs. Epochs: Pure-Python vs. NumPy Implementations

Results

- **Training time**: 1.053 s
- **Final weights:** (vectorized)
  e.g. [–0.372, –0.418, 0.173, –0.046, 0.188, –0.272, 0.192, 0.683]
- **Final bias:** (same logic) –0.004

Evaluation (Validation Set)

- MAE (dollars): 52,481.48
- RMSE (dollars): 71,754.60
- R^2: 0.6112

Part 3: scikit-learn Implementation

Implementation Details

Used `sklearn.linear_model.LinearRegression()`, fitting on the same normalized data.

Results

- **Training time**: 0.0026 s
- MAE (norm): 0.4399
- RMSE (norm): 0.6006
- R^2: 0.6370
- MAE (dollars): 50,774.85
- RMSE (dollars): 69,327.89

Overall Comparison

Implementation	MAE (dollars)	RMSE (dollars)	R^2	Time (s)
Pure Python (Part 1)	$0.4646\times_y$	$0.6498\times_y$	0.6015	45.584
NumPy (Part 2)	52,481.48	71,754.60	0.6112	1.053
scikit-learn (Part 3)	50,774.85	69,327.89	0.6370	0.0026

Table 1: Performance and Run-Time Comparison

Overall Metrics Comparison

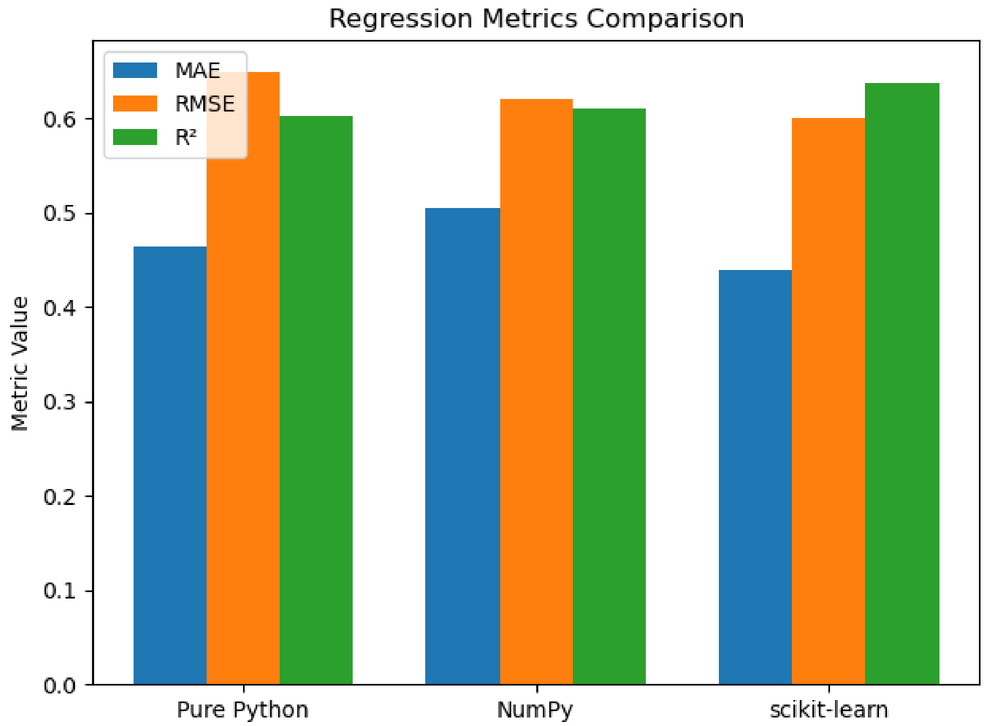

Figure 3: Comparison of MAE, RMSE, and R^2 across implementations

Discussion and Conclusion

- All three implementations converge to similar R^2 (0.60–0.64), demonstrating correctness.
- The pure-Python version serves as a clear didactic baseline but is slow.
- NumPy vectorization yields a $\sim 10\times$ speedup.
- scikit-learn's solver is orders of magnitude faster and matches predictive performance.
- Minor differences in weights and costs stem from different RNGs and floating-point summation order.