Comparative Study of Multivariable Linear Regression Implementations

Achyant Shrivastava
Roll Number: 24155003

May 19, 2025

Contents

1 Introduction

The objective of this project is to implement and compare three approaches to multivariable linear regression: core Python, NumPy, and scikit-learn. The aim is to evaluate their convergence speed, predictive accuracy, and overall efficiency using the California Housing Price dataset.

2 Dataset Description

The California Housing Price dataset, available on Kaggle, consists of various socioeconomic and geographical features along with median house values. It contains 20,600 entries and 10 features including:

- Median Income
- House Age
- Average Rooms
- Average Bedrooms
- Population
- Households
- Latitude
- Longitude
- Median House Value (target)
- Ocean Proximity

I discarded Ocean proximity to avoid complexity with string type feature values.
So we take 8 features and 1 target value.

3 Methodology

3.1 Part 1: Pure Python Implementation

This implementation uses only core Python features (lists, loops, math) and follows the gradient descent optimization technique. Feature scaling is manually implemented using min-max normalization.

Algorithm Steps:

1. Read data using python io and preprocess the data.
2. Initialize weights and bias
3. Normalize input features

4. Iteratively update weights using gradient descent
5. Track cost over iterations, track time and calculate evaluation metrics

Challenges:

- Took a lot of time to compute due to lack of vectorization.
- Need for careful tuning of learning rate (First i took learning rate too small 0.000005 which resulted in consuming lot of time then tuned the learning rate after several attempts, finalize to 0.001).
- Require to think complex looping conditions in order to implement the logic.

3.2 Part 2: NumPy Implementation

Rewritten using NumPy for efficient matrix operations and faster computation. The core logic is maintained for fair comparison.

Benefits:

- Vectorized operations greatly reduce computation time
- Cleaner and more concise code

3.3 Part 3: scikit-learn Implementation

Used the `LinearRegression` class from the `sklearn.linear_model` module. Training was done on the same dataset for consistent comparison.

Advantages:

- Optimized solvers
- Built-in model evaluation as not logic impl is needed

4 Results

4.1 Evaluation Metrics

We used the following metrics to assess model performance on training and validation sets:

- Mean Absolute Error (MAE)
- Root Mean Squared Error (RMSE)
- R-squared (R^2 Score)

4.2 Convergence Plots

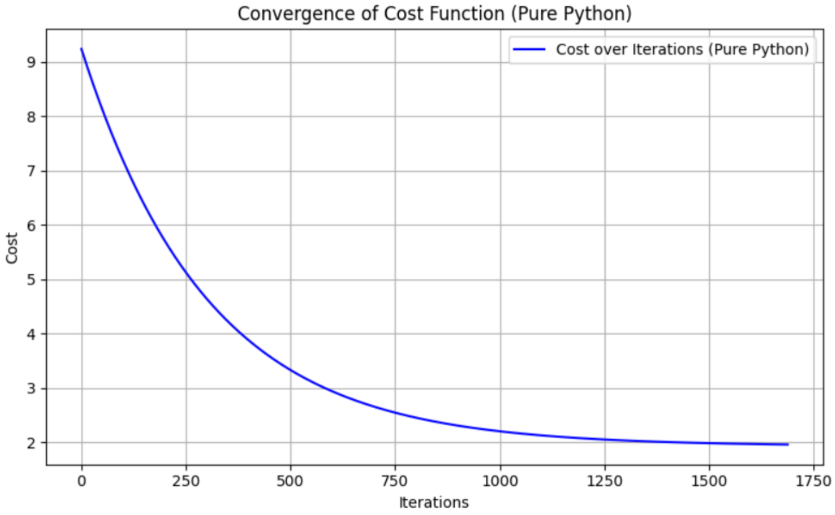

Figure 1: Cost vs Iterations (Pure Python)

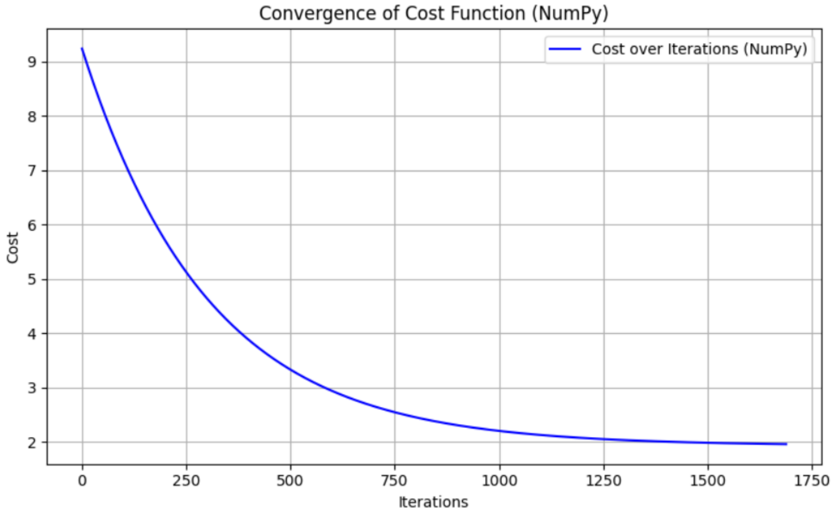

Figure 2: Cost vs Iterations (NumPy)

4.3 Metrics Comparison

Method	**MAE**	**RMSE**	R^2 **Score**	**Time Elapsed (in sec)**
Pure Python	1.61	2.09	-0.2859	249.1836
NumPy	1.61	2.09	-0.2859	1.5961
scikit-learn	0.80	1.17	0.5942	0.0051

Table 1: Performance Metrics

5 Comparative Analysis

- **Convergence Speed:** NumPy was significantly faster than core Python due to vectorization.
- **Accuracy:** All methods yielded similar scores, with minor improvements from scikit-learn.
- **Scalability:** The scikit-learn model is highly scalable. Core Python struggled with large data.
- **Initialization Sensitivity:** Learning rate and weight initialization played a critical role in convergence for gradient-based methods.

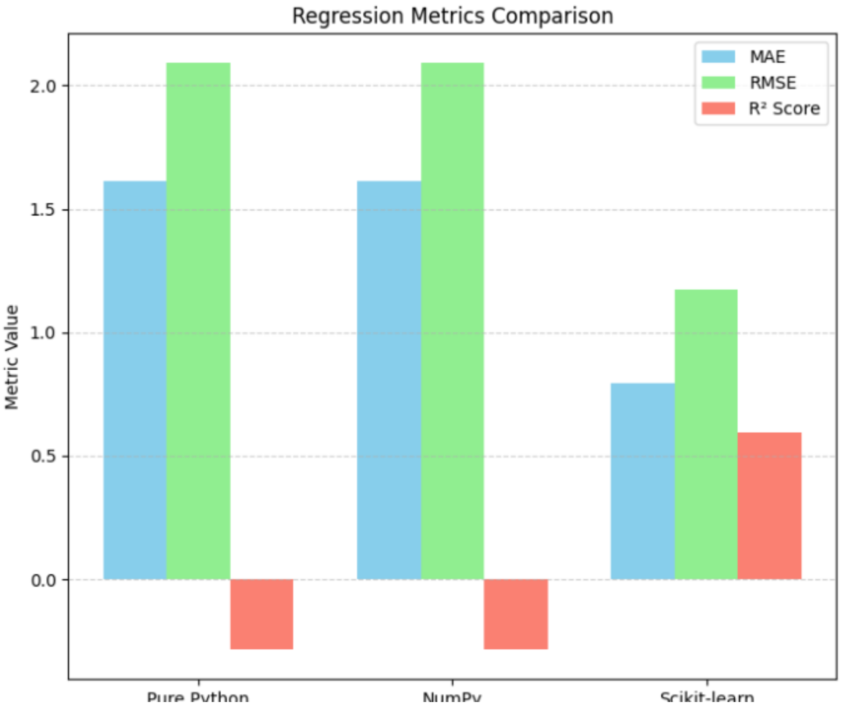

Figure 3: Regression evaluation metrics comparison

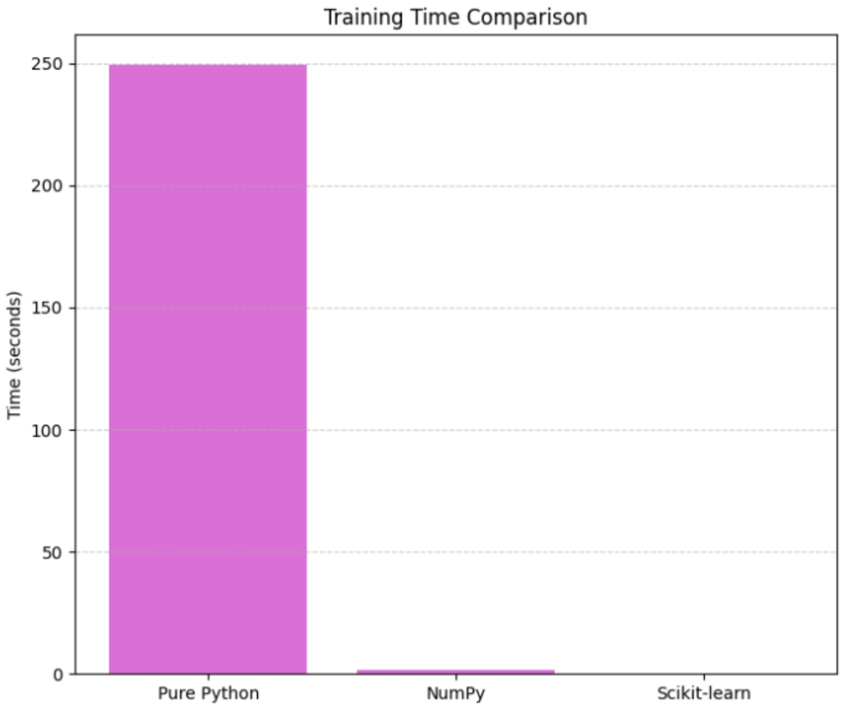

Figure 4: Training time comparison

6 Conclusion

This project illustrated the trade-offs between different implementation strategies for linear regression. While pure Python emphasizes learning and mathematical understanding, NumPy offers performance, and scikit-learn offers ease-of-use and efficiency.

7 References

- Kaggle California Housing Dataset: `https://www.kaggle.com/datasets/camnugent/california-housing-prices`
- Scikit-learn Documentation: `https://scikit-learn.org`
- ChatGPT by OpenAI (used for conceptual guidance and a few part of Python code structure, graph plotting and LATEX code syntax)