Comparative Study of Multivariable Linear Regression Implementations

CSOC IG

Abhishek Kumar Chaubey
IIT (BHU) VARANASI
Roll Number: 24144001

May 19, 2025

Contents

1 Introduction

This report compares three implementations of multivariable linear regression on the California Housing dataset:

- Part 1: Pure Python implementation
- Part 2: NumPy vectorized implementation
- Part 3: Scikit-learn implementation

The goal is to evaluate convergence behavior, prediction accuracy, and computational efficiency.

2 Data Preprocessing

- Missing values were dropped.
- Categorical feature `ocean_proximity` was mapped to integers.
- Features and target variable extracted.
- The dataset was split into training and test sets once using a fixed random seed to ensure reproducibility. This split was saved to a separate file and reused across all three implementations to maintain consistency in evaluation.
- Feature normalization was performed by computing the mean and standard deviation on the training set only. These statistics were then applied to normalize both training and test data, ensuring no information leakage.
- Normalization was also performed for the Scikit-learn implementation, although it uses Ordinary Least Squares (OLS), which is theoretically unaffected by feature scaling. This was done to maintain consistency across all methods.

3 Evaluation Criteria

3.1 1. Convergence Time

The models were timed during training/fitting to compare convergence speed.

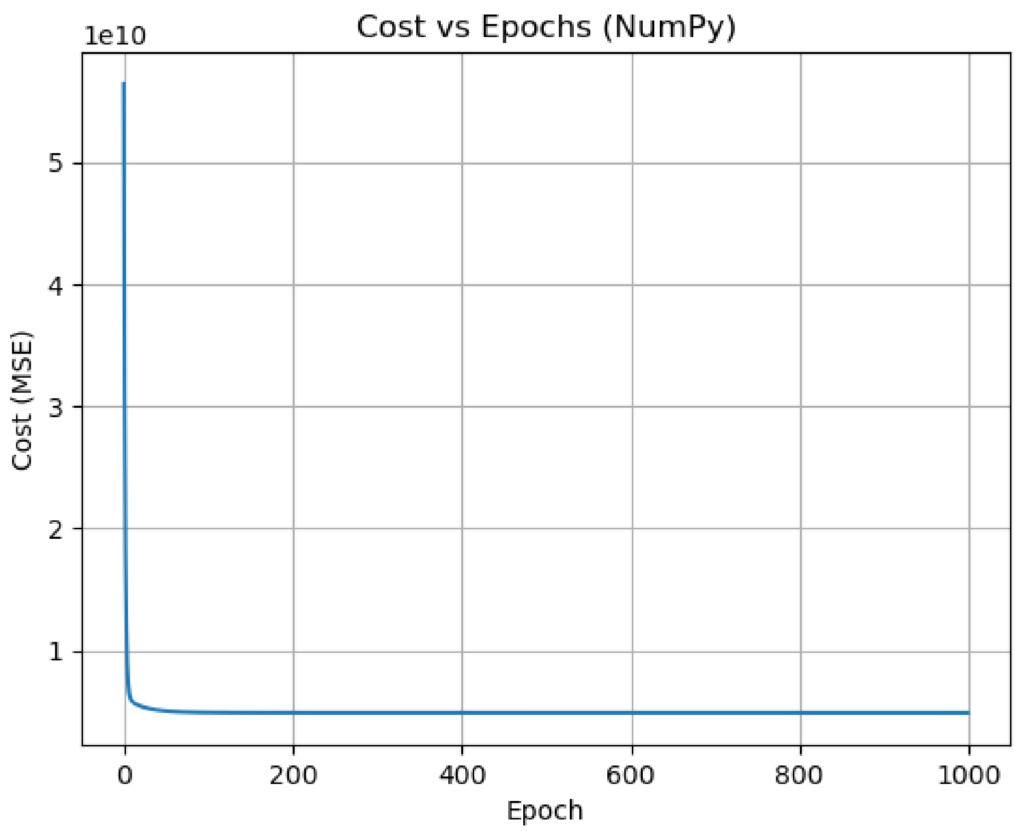
- Pure Python and NumPy models were trained for 1000 epochs.
- Scikit-learn's LinearRegression fitting time recorded.
- Identical initial weights and biases were used for Pure Python and NumPy implementations for fair comparison.

3.1.1 1.1 Time Comparison Results

Model	Training/Fitting Time (seconds)
Pure Python	31.4066
NumPy Vectorized	0.6124
Scikit-learn	0.0044

Table 1: Model convergence/fitting time comparison

Inference: Pure Python took significantly more time compared to NumPy and Scikit-learn because the pure Python implementation uses explicit loops and Python lists, which are slower. NumPy and Scikit-learn utilize vectorized calculations and optimized low-level implementations that greatly improve computational efficiency.


Cost vs Epochs (NumPy)
1e10
5
4
3
2
1
Cost (MSE)
0
200
400
600
800
1000
Epoch

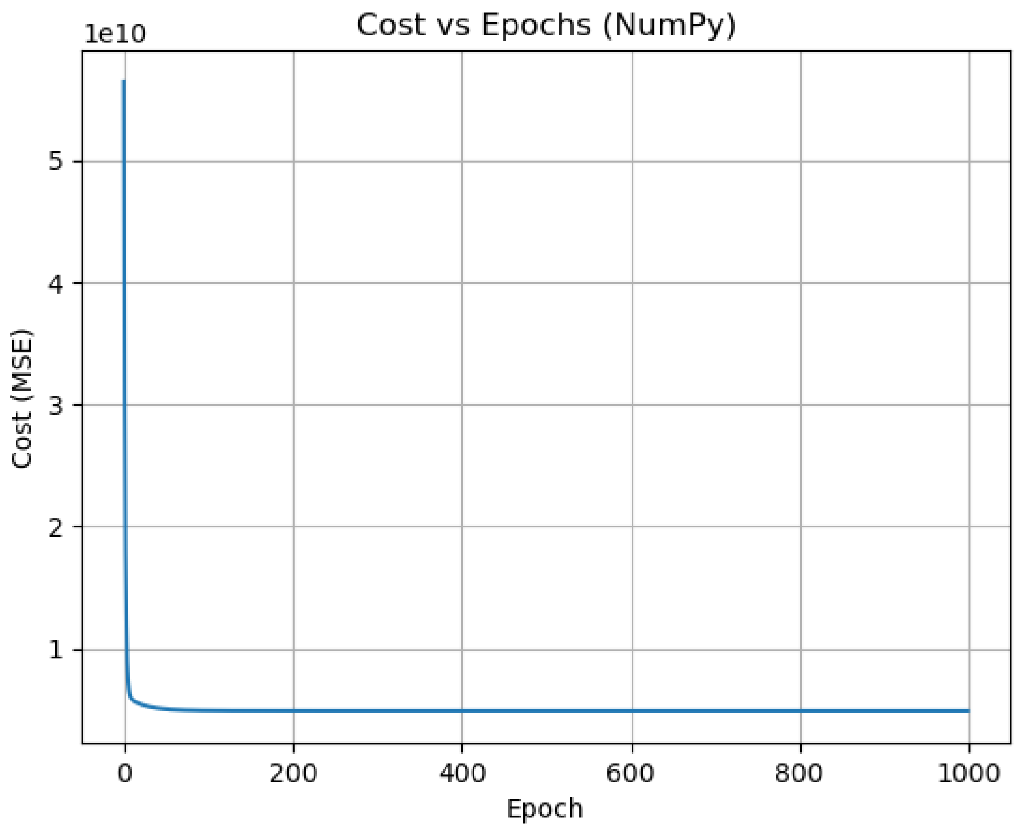


Figure 1: Cost function convergence over epochs for Pure Python and NumPy implementations.

3.1.2 1.2 Prediction Scatterplots

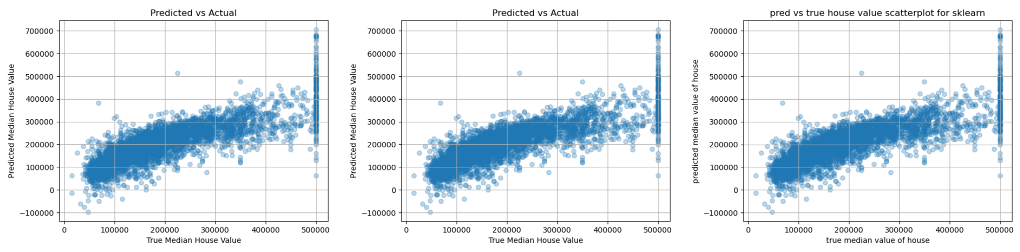


Figure 1: Predicted vs Actual Values for Pure Python (left), NumPy (middle), and Scikit-learn (right) Models

INFERENCES:

- Most data points cluster around the $y = x$ line, indicating good model accuracy.

- **Note:** Some predicted house median prices are negative, which is unrealistic in a real-world scenario. This could be mitigated by:
  - Using constrained optimization techniques.
  - Applying non-negative transformations such as ReLU.
  - Post-processing model outputs (e.g., clipping to zero).

3.2 2. Performance Metrics

Metrics were computed on both training and test datasets for all three models:

Model	Dataset	MAE	RMSE	R^2
Pure Python	Train	51014.6069	69915.1806	0.6328571
	Test	50341.6113	67983.9890	0.6540349
NumPy	Train	51014.6067	69915.1806	0.6328571
	Test	50341.6112	67983.9893	0.6540349
Scikit-learn	Train	51014.0738	69915.1667	0.6328573
	Test	50341.3370	67985.2141	0.6540225

Table 2: Performance metrics comparison

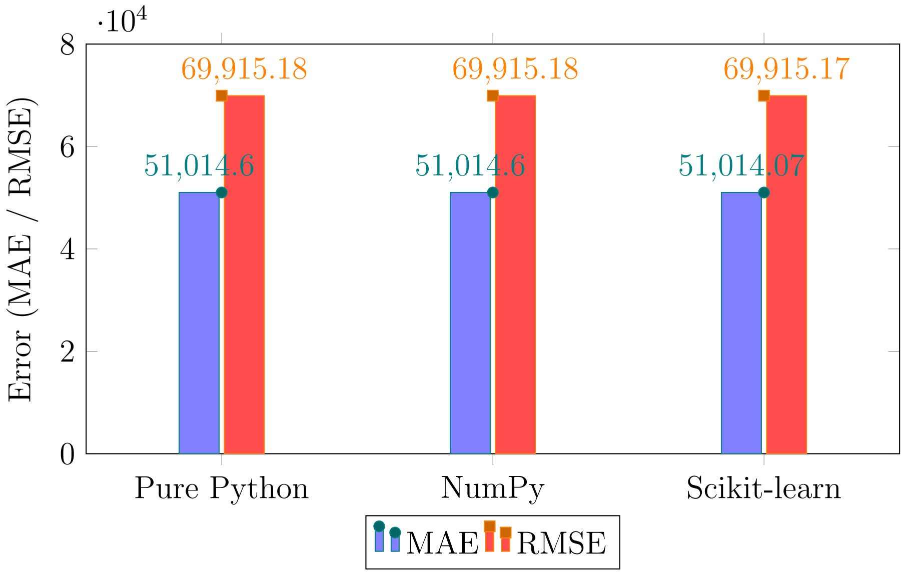


Figure 2: Bar chart comparing MAE and RMSE across models on training data.

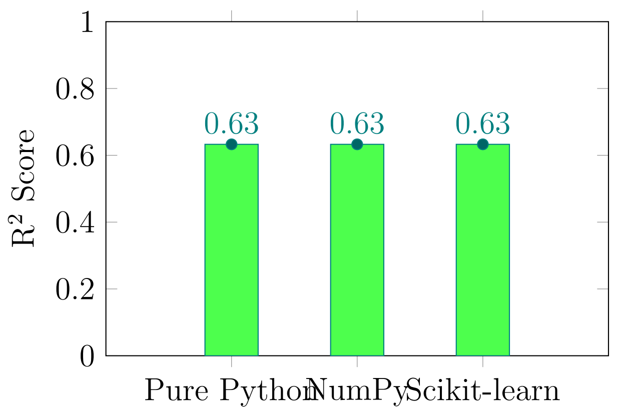


Figure 3: Bar chart comparing R^2 score across models on training data.

2.1 Inferences from Performance Metrics

- All three models achieve nearly identical MAE and RMSE values, indicating consistent predictive error across implementations.
- RMSE values are higher than MAE, reflecting RMSE's sensitivity to larger errors or outliers.
- R^2 scores around 0.63 indicate that the models explain approximately 63% of the variance in the target variable.
- The negligible differences between models confirm that pure Python and NumPy implementations achieve comparable accuracy to Scikit-learn.
- This validates that vectorization and optimization techniques improve efficiency without sacrificing predictive performance.

4 Analysis and Discussion

4.1 Convergence Time

The NumPy vectorized implementation converged significantly faster than the pure Python version due to efficient array operations. Scikit-learn's optimized solvers provided the fastest fitting time overall.

4.2 Performance Metrics

All models achieved comparable accuracy on training and test data, with slight differences attributable to numerical precision and optimization techniques.

4.3 Influence of Initialization and Learning Rate

Identical initialization ensured fairness. The learning rate choice (0.3) was experimentally verified to prevent divergence and enable steady convergence.

4.4 Scalability and Efficiency

Pure Python is not suitable for large datasets due to slow iteration. NumPy offers a balance between performance and code clarity. Scikit-learn is recommended for production due to optimized algorithms and ease of use.

4.5 Visualization Observations

Scatterplots of predicted vs actual values showed good alignment along the $y = x$ line, with some clustering near boundary values explained by dataset distribution.

5 Conclusion

The study aimed to compare three implementations for predicting median house values using the California Housing dataset. We found that while the pure Python implementation achieves comparable prediction accuracy to both the NumPy vectorized version and the Scikit-learn model, it is significantly slower. This performance gap arises because pure Python relies on high-level list operations, which are less efficient than the vectorized computations used by NumPy and the highly optimized algorithms in Scikit-learn. Therefore, for practical applications, leveraging vectorized libraries or dedicated frameworks is recommended to ensure efficient model training without sacrificing accuracy.

Additionally, I utilized a large language model (LLM) to assist with data normalization techniques when the model experienced exploding behavior during training, which helped stabilize and improve convergence. Also for generating latex skeleton for my report and refining grammatical errors.

6 References

- Scikit-learn Documentation: `https://scikit-learn.org`
- NumPy Documentation: `https://numpy.org`
- California Housing Dataset: `https://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html`
- Keith Galli, *Introduction to Pandas*, YouTube, `https://youtu.be/vmEHCJofslg?si=3jZ4v9lV1xMu1IPX`
- Kaggle, *Pandas Tutorial*, `https://www.kaggle.com/learn/pandas`
- NumPy Tutorial, YouTube, `https://youtu.be/4c_mwnYdbhQ?si=NKS3lHhbSftEHSMl`
- Matplotlib Tutorial, YouTube, `https://www.youtube.com/watch?v=0P7QnIQDBJY`
- Kaggle, *Data Visualization*, `https://www.kaggle.com/learn/data-visualization`
- FreeCodeCamp, *Data Preparation Tutorial*, YouTube, `https://www.youtube.com/watch?v=ZV07JD4J-sY`