Multivariable Linear Regression Implementations

Jadhav Rudranil Sandiip
Roll Number: 24095043
CSoC 2025 – Intelligence Guild

May 19, 2025

1 Introduction

This report presents a comparative study of Multivariable Linear Regression using three different approaches: (1) Pure Python, (2) NumPy, and (3) Scikit-learn. We evaluate these approaches in terms of convergence time, predictive accuracy, and overall implementation experience. The dataset used is the California Housing Price dataset. This report includes detailed discussions on methodology, performance metrics, visualization, and critical insights from the results.

2 Dataset: About and Pre-processing

2.1 About

The dataset contains information on houses located in a district of California, based on the 1990 census data. It includes 10 features, with the primary objective being to predict the *Median House Value*.

2.2 Preprocessing

To prepare the dataset for modeling, several preprocessing steps were applied to clean the data, encode categorical features, scale numerical values, and engineer new features. These steps are described in detail below:

- **Data Loading and Cleaning:** The dataset was loaded using `pandas.read_csv()`. A total of 207 rows containing missing values were removed using `dropna()` to ensure model inputs were complete and consistent.
- **Categorical Encoding:** The `ocean_proximity` feature was one-hot encoded using `pd.get_dummies()`, and the resulting boolean values were converted to integers (0 and 1).
- **Train-Test Split:** The dataset was split into training and testing sets using `train_test_split()` with 75% of data used in training and the remaining 25% in testing.

- **Feature Scaling:** Min-max normalization was applied to scale all features to a range between 0 and 1.
- **Feature Engineering:** To evaluate feature relevance, the `.corr()` method was applied to the target variable. Although `households` and `population` individually showed weak correlations with the target, their ratio (`households / population`) exhibited a significantly stronger correlation. A similar observation was made for the ratio of `total_rooms` to `total_bedrooms`. Consequently, these engineered features were included in the model input.

  Another feature, defined as (`latitude`)^2/`longitude`, had a positive correlation with the target (approximately 0.32). However, it was excluded from the final feature set due to the absence of a clear physical interpretation.

  Finally, features with weak correlation to the target variable—such as `population`, `households`, and `total_bedrooms`—were removed from the dataset.
- **Saving Processed Data:** The final training and testing sets were exported to CSV files named `train.csv` and `test.csv`.

These preprocessing steps ensured that the data was clean, normalized, and rich in informative features, which are crucial for effective machine learning model training and evaluation.

3 Methodology

3.1 Part 1: Pure Python Implementation

- Implemented gradient descent algorithm manually to minimize the cost function.
- No external libraries used for improving performance.

3.2 Part 2: NumPy Implementation

- Utilized NumPy for efficient matrix operations.
- Vectorized version of the pure Python implementation.
- Enhanced performance and convergence stability.
- Due to efficient handling of calculations, it was able to perform 10 times more iterations with 500 times less training time.

3.3 Part 3: Scikit-learn Implementation

- Used `LinearRegression()` from `sklearn.linear_model`.
- Automatically optimized using Ordinary Least Squares.
- Simplified implementation with high performance.

4 Evaluation Criteria

4.1 Metrics Used

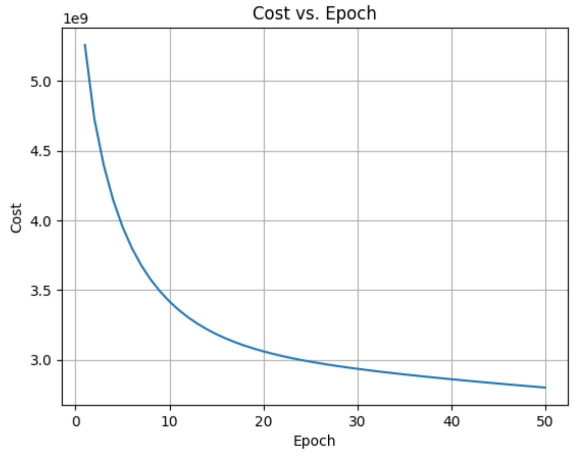
- Mean Absolute Error (MAE)
- Root Mean Squared Error (RMSE)
- R-squared (R^2) Score

4.2 Results Table

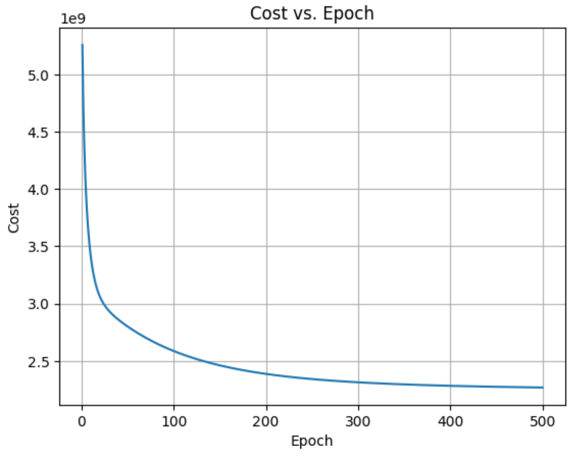
Metric	**Pure Python**	**NumPy**	**Scikit-learn**
MAE	55356.67	48863.18	48720.11
RMSE	74937.60	68384.79	67883.15
R^2 Score	0.5893	0.6579	0.6629
Convergence Time	276.36s	0.49s	0.0058s

5 Visualizations

- Cost function vs. Iterations for Part 1 and Part 2.

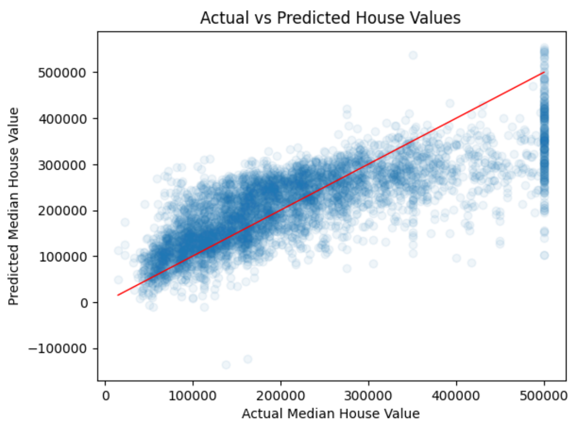


(a) Pure Python (50 epochs)

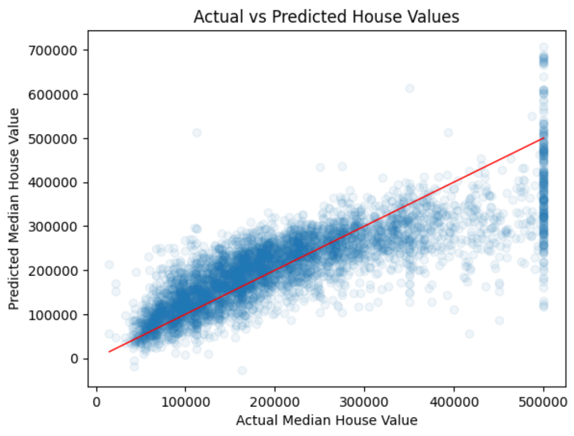


(b) NumPy (500 epochs)

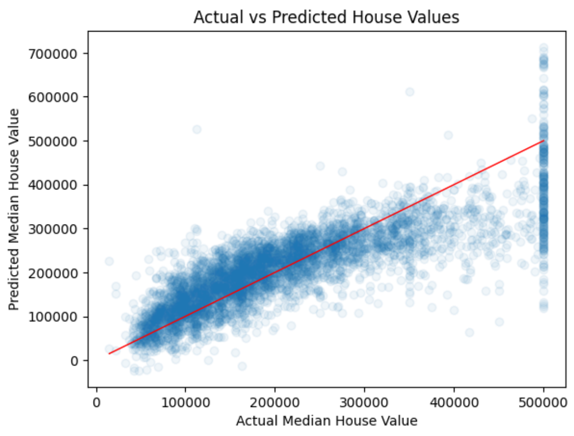
Figure 1: Cost vs. Iterations



(a) Pure Python



(b) NumPy



(c) Scikit-learn

Figure 2: Actual vs. Predicted Value using three different methods of Linear Regression

6 Analysis and Discussion

- Pure Python offered clarity but was computationally expensive.
- NumPy was significantly faster due to vectorization optimized using C.
- Scikit-learn was extremely fast and accurate, best for production use.
- All 3 methods perform well at low and medium valued house but our model failed to predict well and mostly undervalued houses with high actual mean value
- When initialized with the same parameters and learning rate, the final model parameters should ideally be similar but due to computational limitations they were diffrent ,PS : If we ran Part 2 with same number of iterations as Part 1 we would get same result.
-

7 Conclusion

This assignment provided a thorough understanding of Multivariable Linear Regression. Implementing the model from scratch and comparing it with library-based implementations highlighted key performance considerations.