Comparative Study of Multivariable Linear Regression Implementations

U.Ravi Kiran , 23074031

May 19,2025

1 Abstract

This report presents a comparative study of multivariable linear regression using three different implementations : Pure Python, Optimized NumPy, and Scikit learn. The objective is to evaluate performance in terms of computational efficiency and predictive accuracy on the California Housing Prices dataset. All approaches use the same preprocessed data and are evaluated on metrics including training time, Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and R^2 Score.

2 Introduction

Linear regression is a fundamental supervised learning algorithm used to predict continuous values. This project aims to compare three different implementations of multivariable linear regression on a realworld dataset like California Housing Prices. By implementing the algorithm in Pure Python, NumPy, and Scikit learn, we got differences in training time and prediction quality.

3 Dataset and Preprocessing

The California Housing Prices dataset includes features such as median income, house age, average rooms, and more. All data preprocessing steps such as minmax scaler for normalization, train test split and selecting all features means all columns were used to train models and all these methods are applied identically across all to ensure a fairness.

4 Methodology

4.1 Pure Python

Implemented using basic for loops and list comprehensions, using gradient descent for optimization. This version is the slowest due to lack of vectorization.Too more time than any other

4.2 Optimized NumPy

Utilized NumPy arrays for vectorized operations, significantly improving computational performance while maintaining identical logic to the pure Python version.So got same MAE RMSE and R^2 score but taken so less time compared to pure python

4.3 Scikit learn

Used Scikit-learn's `LinearRegression` model which is highly optimized and has linear algebra libraries internally. This model achieves high accuracy and fast computation.But used MinMax() scaler instead of Standard Scaler()

5 Results

The following table summarizes the performance of each approach:

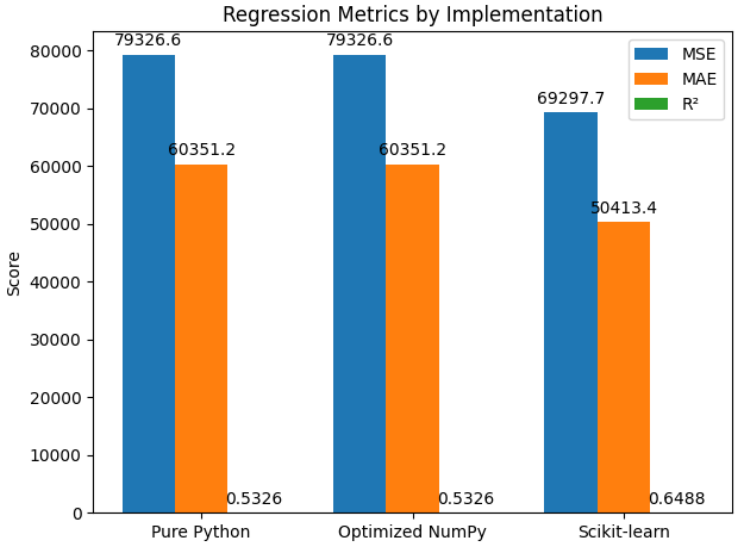

Table 1: Performance Comparison of Regression Implementations

Implementation	Training Time (s)	MAE	RMSE	R^2 Score
Pure Python	59.82	60351.17	79326.55	0.5326
NumPy	0.1526	60351.17	79326.55	0.5326
Scikit-learn	0.2376	50413.43	69297.72	0.6488

6 Analysis and Discussion

The Pure Python implementation while functionally correct but has long execution time due to lack of vectorization. The NumPy implementation drastically reduces training time with identical accuracy, showcasing the benefit of optimized numerical libraries. Scikit learn outperforms both in terms of predicting accuracy and efficiency, making it the most suitable for practical applications.

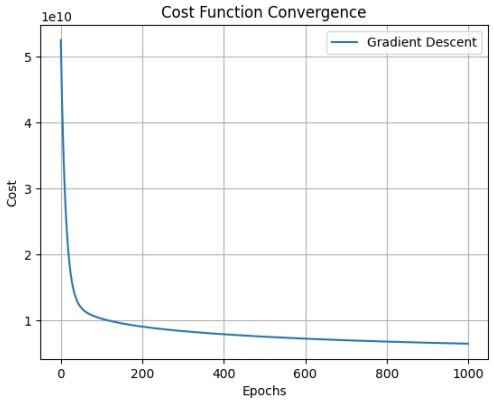

Figure 1: for pure python

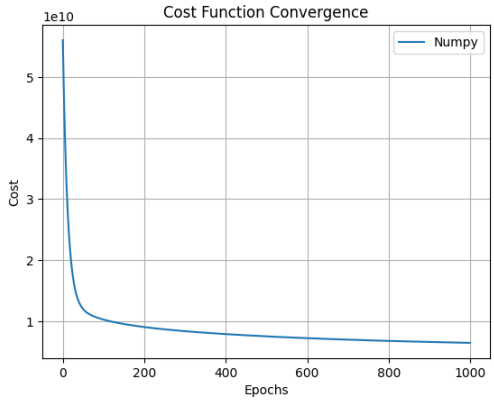

Figure 2: For Numpy

The above are the cost convergence graph for both pure python and numpy and their final cost is same as we can clearly see from the graph but Numpy converges more fastly than the pure python as we can clearly see the time at which training completes.
And if i change the weight initialization from all zeroes to ones then also i got same accuracy metrics which means my code is converging to the optimal value every time.And i initialized randomly with digits and checked it is also given same accuracy metrics.
And as for the learning rates for 1000 epoches I tried changing the learning rates started with 0.01 I got RMSE aprrox 80K but changing values in 0.2,0.5,0.7,0.9 and I found upto 0.93 the RMSE goes to 69K and for 0.94 started increasing again and for 1 got almost infinity.I got minimum Accuracy metrics at in between 0.93-0.94.Means up to this final cost decreases and then increases in between.

7 Conclusion

This study demonstrates the between implementation complexity and performance. While building models from scratch aids in learning, libraries like NumPy and Scikit learn is crucial for scalability and real world problems.