Multiple Variable Linear Regression

Prerna Tripathi

May 19, 2025

1. Initial Procedures

1.1. Approach

The common process across all the three parts of the assignment is to preprocess the data by removing all the missing values in the dataset which can lead to improper training of the model and also convert the feature 'ocean proximity'into a form the algorithm can understand and process using one-hot encoding. The dataset is then divided into a 80:20 ratio into training and validation sets. The features are then also scaled using Z-score normalisation across all the three parts. Feature scaling is highly necessary for implementing gradient descent. Few things as in the formation of the feature 'ocean proximity' was done with the help of gpt.

2. Pure Python Implementation

The pure python implementation implements multiple variable linear regression using gradient descent without using any libraries,using only loops and lists.The maximum no. of iterations is 1000 and the learning rate is 0.01. The graph of convergence vs iteration shows that our gradient descent has converged well enough to a point of global minimum Use of AI was done in areas using scikit-library to calculate regression metrics and in splitting the data.Otheriwse the implementation of the main gradient descent algorithm was done avoiding the use of AI.

2.1. Analysis

Convergence time: 713.8439 seconds MAE= Training:50081.0779, Validation:50081.3546 RMSE= Training:69272.7465, Validation:68649.0109 R2-score= Training:0.6394, Validation:0.6480

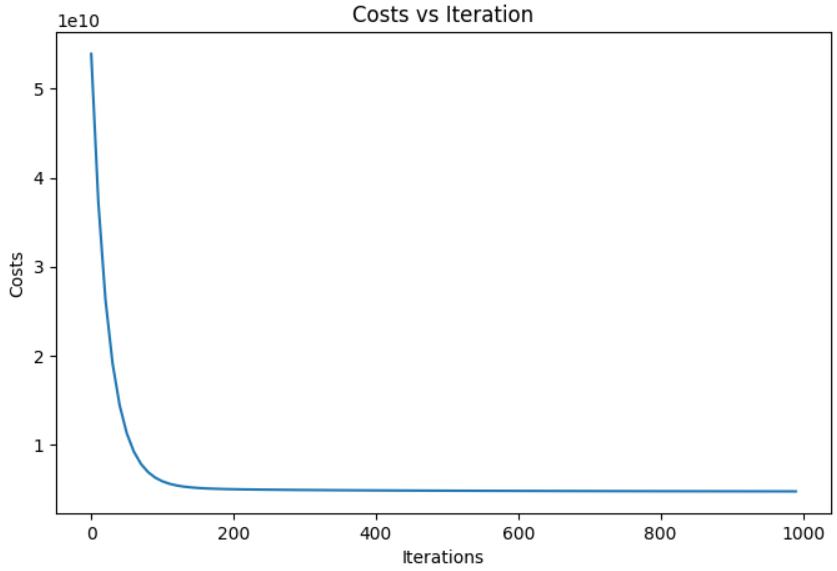

Figure 1: Graph of Convergence vs Iteration:Pure Python

3. Numpy Implementation

In this approach, the algorithm remains the same as the first part including the initial parameters, but instead of loops and lists Numpy library is used which through vectorized algebra and parallel operations improves the execution speed and shortens the code also. Here also the learning rate is 0.01 and the no. of iterations is 1000 and the graph of convergence vs iterations shows that our gradient decsent has converged well enough to a minimum. Use of AI was again restricted to the use of scikit-library functions and also to a few numpy operations which have been used in the code.

3.1. Analysis

Convergence time: 0.2084 seconds MAE= Training:49964.6857,Validation:49973.7986 RMSE= Training:69200.6814, Validation:68569.5633 R2-score= Training:0.6402, Validation:0.6489

4. Scikit-learn Implementation

4.1. Approach

The scikit-learn implementation uses the `LinearRegression` class, which abstracts away the algorithmic details. It provides built-in methods for model fitting, prediction, and performance evaluation. Again the use of AI has been done to incorporate the scikit's operations.

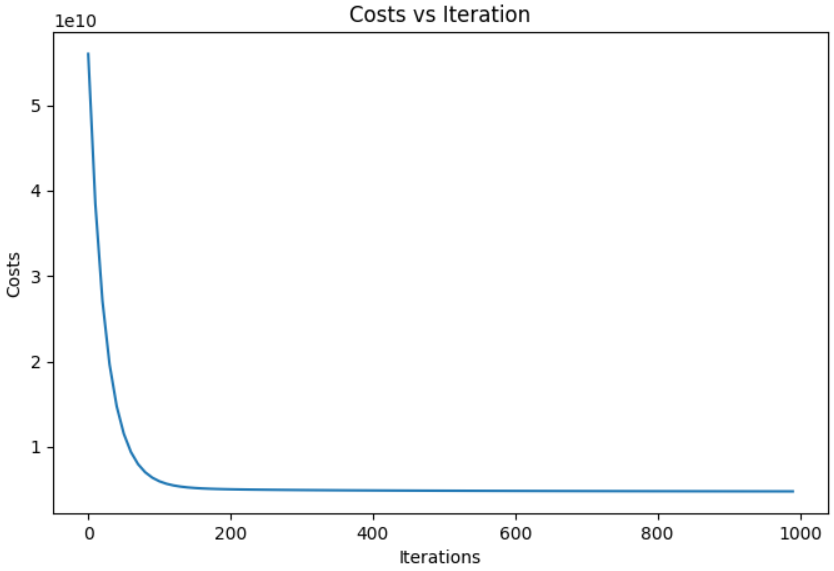

Figure 2: Graph of Convergence vs Iteration: Numpy Optimised

4.2. Analysis

Fitting Duration:0.0132 seconds MAE= Training:49752.9160, Validation:49964.0128 RMSE= Training:68758.3663, Validation:68201.5212 R2-score= Training: 0.6448, Validation=0.6526

5. Discussion

In the first two parts the initial parameter values decide from which point are we starting the gradient descent,i.e from which point the minimising is beginning and hence it was really important to keep those initials same for both the implementations in order for a good comparison. The learning rate also is very crucial as a very small value can make our convergence very slow and a large value can cause overshooting or divergence and hence a proper value was necessary. Again the value had to be kept same for both numpy and python so as to compare their convergence times. We observe that the convergence time for the pure python implmentation was way too much compared to the numpy one. This is because the numpy approach optimises the updations through parallel operations unlike the pure python one which updates through loops iterating one by one. Hence the numpy approach shortens the code and is way much faster and is similar in accuracy to the pure python implementation hence it is a much more efficient method. Out of the above 3 implementations the most efficient method was surely using scikit library as it was very user friendly and had built in

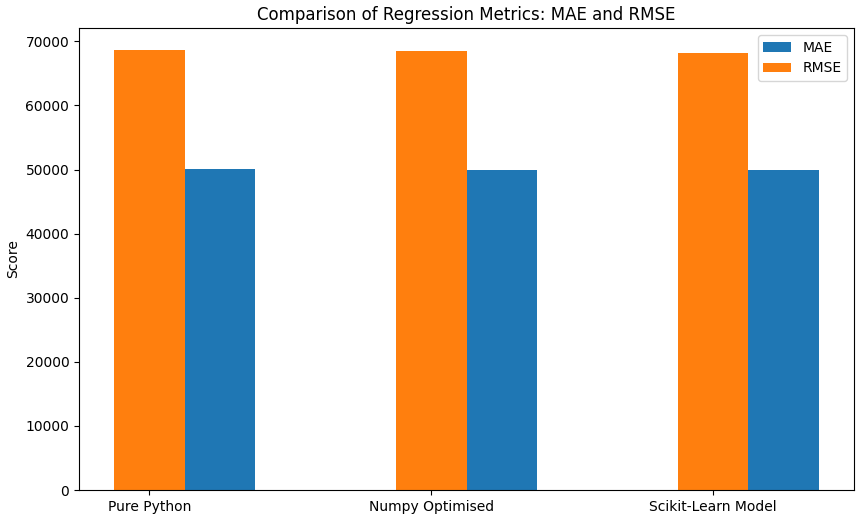

Figure 3: Comparsion Of MAE and RMSE over the three parts

models to fit and train the model.

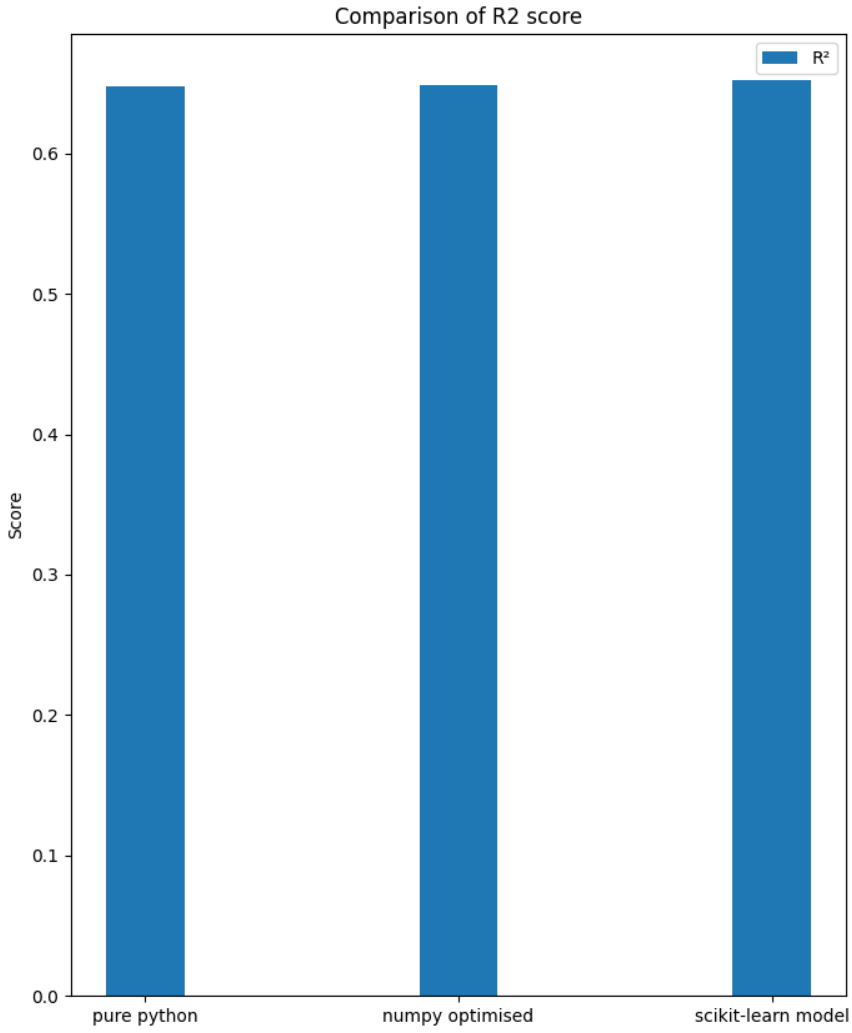

Figure 4: Comparsion Of R2 score over the three parts