COPS Summer of Code Report

M Gokulan,
Roll no: 24155043
Department: Mining Engineering

Project Title:
Comparative Study of Multiple Linear Regression Implementations

Abstract

This report discusses the implementation of a Multiple Linear Regression Model with three distinct approaches and compares the three models in terms of convergence speed and regression metrics. All the three approaches are trained on the same dataset with same scaling and same training dataset.

Introduction

Multiple Linear Regression is a supervised learning technique for a dataset with one dependent variable and two or more independent variables. In this project I implemented a Multiple Linear Regression model with three different strategies:

1. **Pure Python Code Implementation**
   Developing a multiple linear regression algorithm only using the core Python features with gradient descent as the optimization technique
2. **Optimized NumPy Implementation**
   Developing the same algorithm with the help of NumPy Python library to leverage vectorized operations with gradient descent optimization technique
3. **Scikit-learn Implementation**
   Using the LinearRegression class from the scikit-learn Python library to train the Multiple Linear Regression model on the same training dataset used in the previous two strategies.

Data Preprocessing

Dataset: California Housing Prize Dataset Kaggle
Features: Latitude, Longitude, Housing Median Age, Total Rooms, Total Bedrooms, Population, Median Income, Ocean Proximity
Target: Median House Value
Label Encoding: The feature Ocean Proximity contains object data type and it is encoded by label encoding method.
Scaling: StandardScalar is used as a scaling technique for all the features and target

Evaluation Metrics

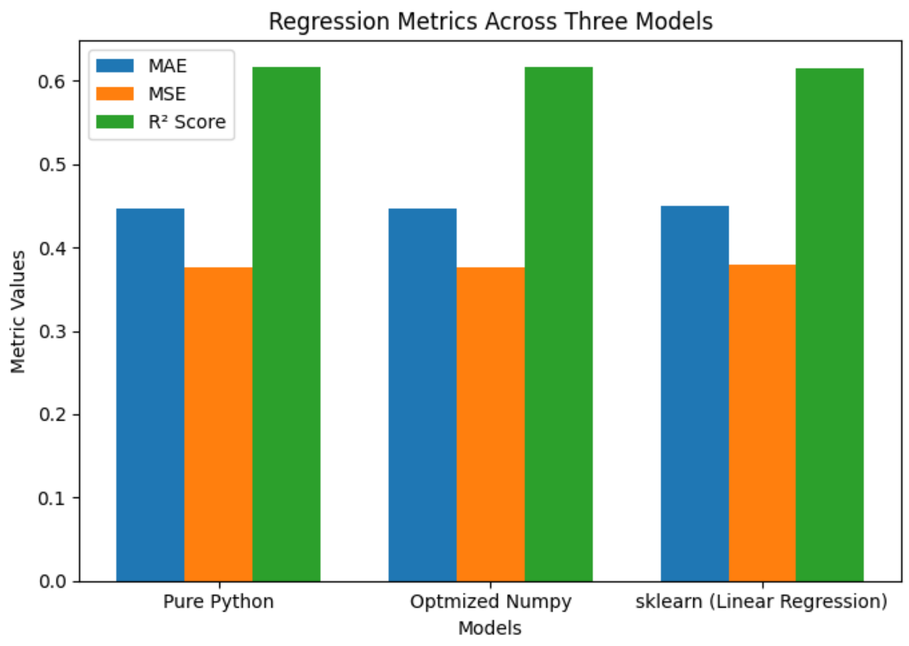

Figure 1: Regression Metrics

- The Pure Python model and Optimized NumPy model uses the gradient descent as the optimization technique but in the case of scikit learn LinearRegression model which uses the OLS (Ordinary Least Squares) to find the suitable weights and bias for the model.
- The Model1 and Model2 are stopped training once they reached convergence (where further training does not lead to substantial improvements).

- The similarity in the regression metrics across all the three models indicates that they have reached the global minimum of cost function despite the implementation methods are different among the three models but the mathematical goal(to reach the minimum cost) of optimization is same.
- In the Model1 and model2 I have used the gradient descent and for the Model3 I have used the OLS method but still the regression metrics of Model3 is almost same as Model1 and Model2 this is because the method of finding the optimized weights and bias may be different but at the end the goal of the optimization technique is to get the global minimum of cost function and that is why they all have similar regression metrics.

Cost Functions Convergence

The Cost Functions for the both the models (Pure Python, Optimized NumPy) are converging exactly in same manner that's because the mathematical approach of finding the optimized weights and bias is same (i.e. Gradient Descent) for the both models.

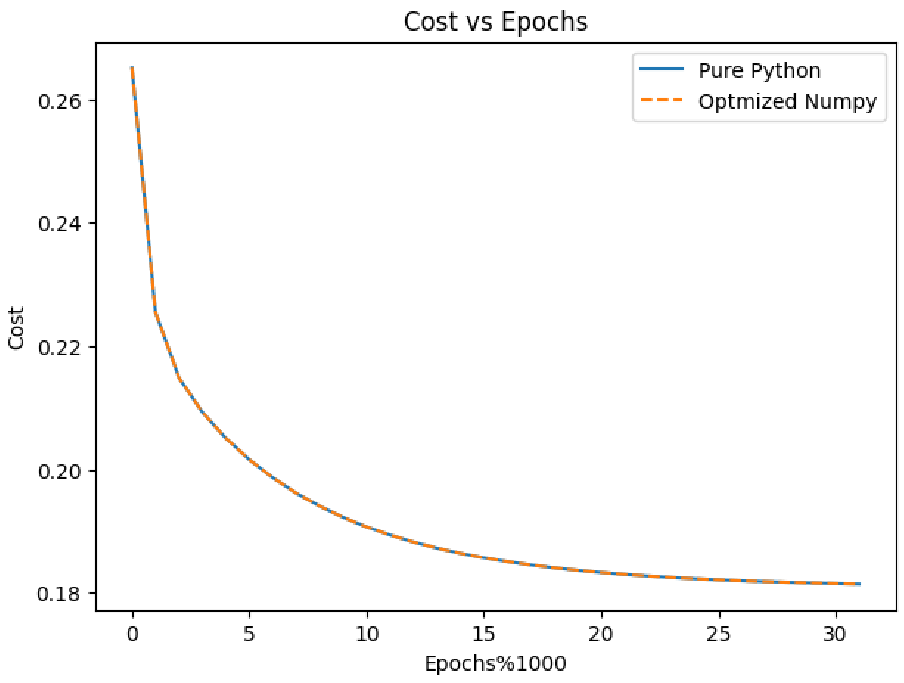

Figure 2: Cost vs Epochs

Time of Convergence

The time taken from the starting of the training process up to the point where the cost function approximately reaches its absolute minima is called the time of convergence.

Model	Time of Convergence
Model1 (Pure Python)	539.45 sec
Model2 (Optimized NumPy)	0.86 sec

- Model 3 (Scikit-Learn Linear Regression) 5.94 milli sec (fitting time)

Convergence Speed

- The Model1 and Model2 both converged at the same epoch 3200 but the time taken for both the model has a huge difference.
- Model 2 (Optimized NumPy) reached the convergence state very fastly as compared to Model 1.
- This is because the Model 1 (Pure Python) obtain the optimized weights and bias through manual looping each time and it takes more time whereas in the model 2 the vectorized form of calculating the optimized weights and bias takes less time by using NumPy library.

Scalability and Efficiency

1. The Pure Python Implementation has a poor scalability because as the dataset size increases the loops get exponentially slower for calculating the optimized weights and bias each time.
2. The Optimized NumPy has a good Scalability much faster but still limited by single threaded execution and memory constraints.
3. The Scikit-Learn LinearRegression model has an excellent scalability and it has a very high-level efficiency on large datasets and it uses optimized linear algebra, multi-threading and an optimized memory constraint.

Influence of Initial parameters on Convergence

- The convergence will be reached when we reach almost close to absolute minima.
- I have initialized all the weights and bias as zero initially for both the model 1 and model 2 to compare the convergence of two models and as I expect they both converge at the same epoch because to achieve the absolute minima.
- I have implemented the same method of optimization and that's why the both converge at the same point but not at same time because the vectorized form makes it faster to calculate than by using manual loops.
- But if we initialize with some random values of weights and bias for both the models there may differ in the point of convergence because if model1's randomly initialized weights and bias lies closer to the convergence then it will reach the convergence state faster in terms of epochs this is how the initial parameters affect the convergence.

Influence of Learning Rate on Convergence

The right learning rate for a model makes the cost function to reach the absolute minima quickly and smoothly and I have used 0.01 Learning rate for both the models. The figure 2 clearly shows that cost functions converge smoothly.

- Too High Learning Rate causes the cost functions to fluctuate.
- Too Low Learning Rate leads to very slow convergence and may takes more iterations (epochs).