Comparative Study of Multivariable Linear Regression Implementations

By
Tanishq Singh

COPS Summer of Code 2025

Intelligence Guild

Club of Programmers, IIT (BHU) Varanasi

Official IG Website: https://cops-iitbhu.github.io/IG-website

Introduction

Multivariable linear regression is a supervised learning algorithm used to predict a continuous outcome based on multiple input features.

Hypothesis and Error Function

Let the number of features be n, and the input vector be $\mathbf{x} = [x_1, x_2, \ldots, x_n]$, so the hypothesis becomes:

$$\hat{y} = \beta_0 + \beta_1 x_1 + \ldots + \beta_n x_n$$

The error (cost) function we aim to minimize is the Mean Squared Error (MSE):

$$E = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

where:

1. n is the number of training examples,
2. y_i is the actual output for the i^{th} example.

Gradient Computation

To perform gradient descent, we need the partial derivatives of the cost function with respect to each parameter β_i. The derivative is:

$$\frac{\partial E}{\partial \beta_0} = \frac{-2}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)$$

$$\frac{\partial E}{\partial \beta_i} = \frac{-2}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)\, x_i$$

Gradient Descent Algorithm

We update each parameter β_i using the rule:

$$\beta_2 := \beta_1 - \alpha \cdot \frac{\partial E}{\partial \beta_i}$$

where:

1. α is the learning rate (a small positive value).
2. β_2 is new beta
3. β_1 is old beta

Steps:

1. Initialize all β_i to 1 or small random values. 2. Repeat until convergence:

- Compute the prediction $\hat{y}_i$ for all m examples.
- Calculate the gradient for each β_i.
- Update each parameter using the update rule.

3. Use the learned parameters for prediction.

1. Pure Python Implementation

```python
class MLRGD_Core:
    def __init__(self, learning_rate=0.01, epochs=100):
        self.coef = None
        self.intercept = None

        self.lr = learning_rate
        self.epochs = epochs

        self.t2c = None         # Time to Converge

        self.error = []         # To store error at each epoch
        self.iterations = []    # To store iteration numbers

    def fit(self, x, y):
        self.intercept = 0 # Assuming zero
        self.coef = [1 for i in range(len(x[1]))] # Assuming with all ones

        self.t2c = time.time()

        for i in range(self.epochs):
            y_hat = [(self.intercept + sum([self.coef[k] * x[j][k] for k in range(len(x[j]))])) for j in range(len(x))]
            der_intercept = -2 * (sum([y[j] - y_hat[j] for j in range(len(x))]) / len(x))

            self.error.append(np.mean((y_train - y_hat) ** 2))
            self.iterations.append(i)

            for j in range(len(x[0])):
                der_coef_j = -2 * (sum([(y[k] - y_hat[k]) * x[k][j] for k in range(len(x))]) / len(x))
                self.coef[j] = self.coef[j] - (self.lr * der_coef_j)

            self.intercept = self.intercept - (self.lr * der_intercept)

        self.t2c = time.time() - self.t2c

    def predict(self, x):
        return [(self.intercept + sum([self.coef[k] * x[j][k] for k in range(len(x[j]))])) for j in range(len(x))]
```

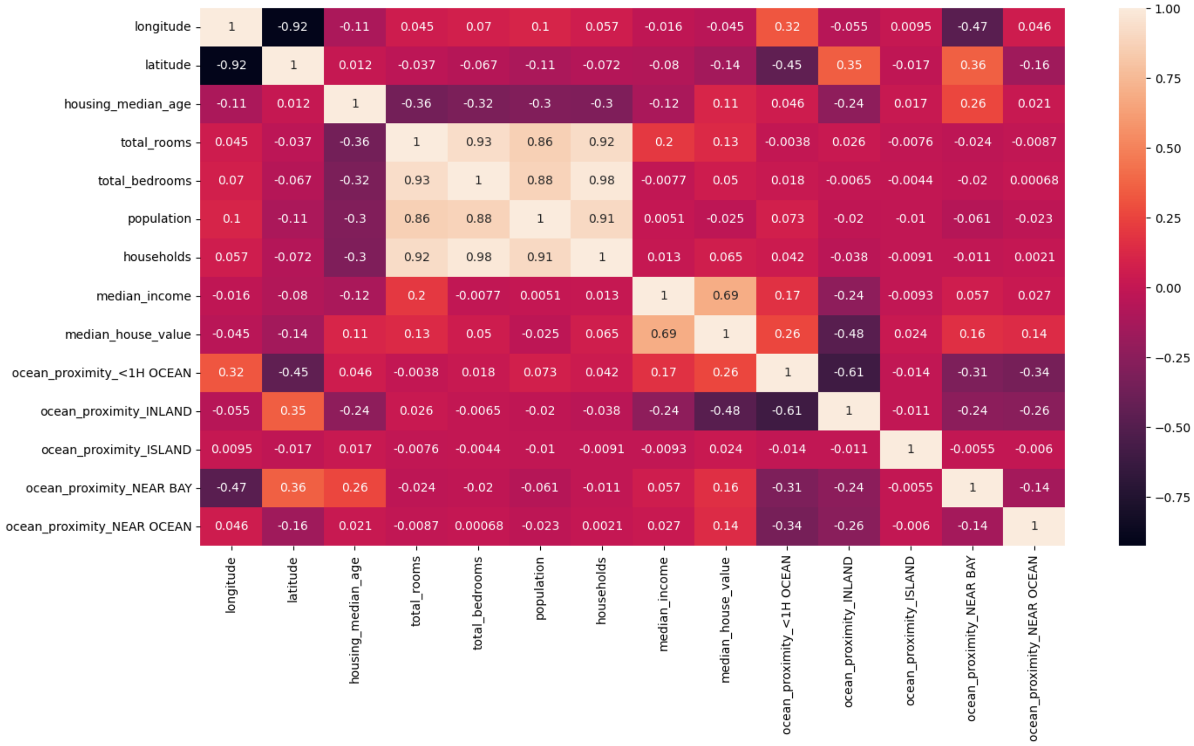
2. Numpy Implementation

```
class MLRGD:
    def __init__(self, learning_rate=0.01, epochs=100):
        self.coef = None
        self.intercept = None

        self.lr = learning_rate
        self.epochs = epochs

        self.t2c = None         # Time to Converge

        self.error = []         # To store error at each epoch
        self.iterations = []    # To store iteration numbers

    def fit(self, x_train, y_train):
        n_rows, n_features = x_train.shape

        self.intercept = 0 # Assuming zero
        self.coef = np.ones(n_features) # Assuming all ones

        self.t2c = time.time()

        for i in range(self.epochs):
            y_hat = np.dot(x_train, self.coef) + self.intercept

            self.error.append(np.mean((y_train - y_hat) ** 2))
            self.iterations.append(i)

            der_intercept = -2 * np.mean(y_train - y_hat)
            der_coef = -2 * (np.dot((y_train - y_hat), x_train) / n_rows)

            self.coef = self.coef - (self.lr * der_coef)
            self.intercept = self.intercept - (self.lr * der_intercept)

        self.t2c = time.time() - self.t2c

    def predict(self, x_test):
        return np.dot(x_test, self.coef) + self.intercept
```

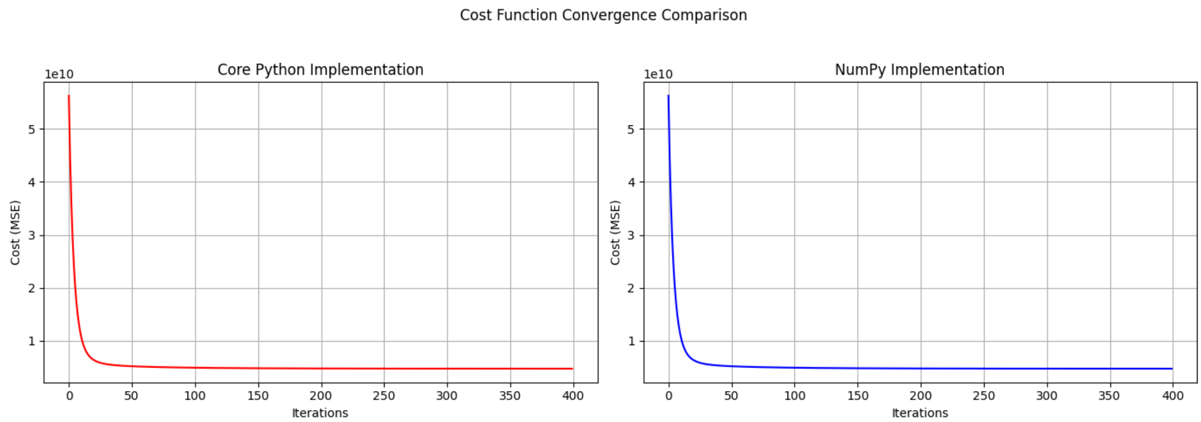
3. Scikit-Learn Implementation

```
from sklearn.linear_model import LinearRegression
```

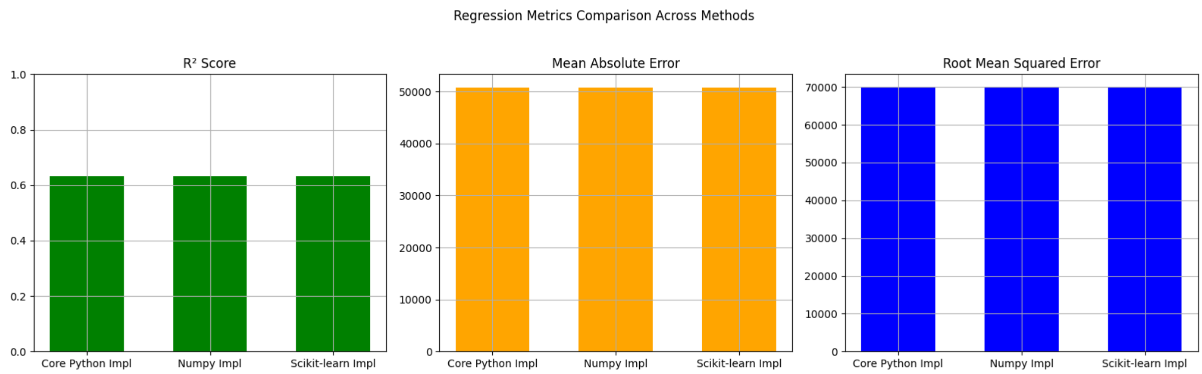
Metrics



longitude 1 -0.92 -0.11 0.045 0.07 0.1 0.057 -0.016 -0.045 0.32 -0.055 0.0095 -0.47 0.046
latitude -0.92 1 0.012 -0.037 -0.067 -0.11 -0.072 -0.08 -0.14 -0.45 0.35 -0.017 0.36 -0.16
housing_median_age -0.11 0.012 1 -0.36 -0.32 -0.3 -0.3 -0.12 0.11 0.046 -0.24 0.017 0.26 0.021
total_rooms 0.045 -0.037 -0.36 1 0.93 0.86 0.92 0.2 0.13 -0.0038 0.026 -0.0076 -0.024 -0.0087
total_bedrooms 0.07 -0.067 -0.32 0.93 1 0.88 0.98 -0.0077 0.05 0.018 -0.0065 -0.0044 -0.02 0.00068
population 0.1 -0.11 -0.3 0.86 0.88 1 0.91 0.0051 -0.025 0.073 -0.02 -0.01 -0.061 -0.023
households 0.057 -0.072 -0.3 0.92 0.98 0.91 1 0.013 0.065 0.042 -0.038 -0.0091 -0.011 0.0021
median_income -0.016 -0.08 -0.12 0.2 -0.0077 0.0051 0.013 1 0.69 0.17 -0.24 -0.0093 0.057 0.027
median_house_value -0.045 -0.14 0.11 0.13 0.05 -0.025 0.065 0.69 1 0.26 -0.48 0.024 0.16 0.14
ocean_proximity_<1H OCEAN 0.32 -0.45 0.046 -0.0038 0.018 0.073 0.042 0.17 0.26 1 -0.61 -0.014 -0.31 -0.34
ocean_proximity_INLAND -0.055 0.35 -0.24 0.026 -0.0065 -0.02 -0.038 -0.24 -0.48 -0.61 1 -0.011 -0.24 -0.26
ocean_proximity_ISLAND 0.0095 -0.017 0.017 -0.0076 -0.0044 -0.01 -0.0091 -0.0093 0.024 -0.014 -0.011 1 -0.0055 -0.006
ocean_proximity_NEAR BAY -0.47 0.36 0.26 -0.024 -0.02 -0.061 -0.011 0.057 0.16 -0.31 -0.24 -0.0055 1 -0.14
ocean_proximity_NEAR OCEAN 0.046 -0.16 0.021 -0.0087 0.00068 -0.023 0.0021 0.027 0.14 -0.34 -0.26 -0.006 -0.14 1
longitude
latitude
housing_median_age
total_rooms
total_bedrooms
population
households
median_income
median_house_value
ocean_proximity_<1H OCEAN
ocean_proximity_INLAND
ocean_proximity_ISLAND
ocean_proximity_NEAR BAY
ocean_proximity_NEAR OCEAN
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75



Cost Function Convergence Comparison
Core Python Implementation
NumPy Implementation
1e10
Cost (MSE)
Iterations
0 50 100 150 200 250 300 350 400
1 2 3 4 5



Regression Metrics Comparison Across Methods
R² Score
Mean Absolute Error
Root Mean Squared Error
Core Python Impl
Numpy Impl
Scikit-learn Impl