Detailed Report on Linear Regression: Boston Housing Dataset

Atharv Bhatt

October 3, 2025

1 Introduction

Linear regression is a foundational algorithm in supervised machine learning for predicting a continuous numerical outcome. It establishes a linear relationship between a dependent variable y and one or more independent variables x.

In this project, we implement linear regression from scratch to predict house prices (`medv`) using the average number of rooms (`rm`) in the Boston Housing dataset. The implementation emphasizes **loss function derivation, gradient descent optimization, and iterative model training**.

2 Dataset Description

The Boston Housing dataset contains information about residential homes in Boston, including:

- `rm` – average number of rooms per dwelling (feature)
- `medv` – median value of owner-occupied homes in $1000's (target)

Other features exist but for simplicity, we use `rm` as the predictor.

3 Data Preprocessing

We extract the relevant columns and convert them into NumPy arrays for computation:

```
X = data['rm'].values
y = data['medv'].values
```

Here, X contains the feature values (average rooms) and y contains the target house prices. This conversion allows efficient vectorized computations in gradient descent.

4 Linear Regression Model

The model predicts the target using a simple linear equation:

$$\hat{y} = mx + b$$

where:

- m is the slope of the line (weight)
- b is the y-intercept (bias)
- $\hat{y}$ is the predicted house price

4.1 Loss Function

We use **Mean Squared Error (MSE)** to measure prediction error:

$$L(m,b) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - (mx_i + b))^2$$

The goal is to minimize $L(m,b)$ with respect to m and b.

5 Gradient Descent Optimization

To minimize the loss function, we compute its partial derivatives with respect to the model parameters:

$$\frac{\partial L}{\partial m} = -\frac{2}{n}\sum_{i=1}^{n} x_i(y_i - (mx_i + b))$$

$$\frac{\partial L}{\partial b} = -\frac{2}{n}\sum_{i=1}^{n}(y_i - (mx_i + b))$$

We iteratively update the parameters using:

$$m := m - \alpha\frac{\partial L}{\partial m}, \quad b := b - \alpha\frac{\partial L}{\partial b}$$

where α is the learning rate, controlling the step size.

5.1 Python Implementation

```
def gradient_descend(m_now, b_now, points, learning_rate):
    m_gradient = 0
    b_gradient = 0
    n = len(points)
    for i in range(n):
```

```
        X = points.iloc[i].rm
        y = points.iloc[i].medv
        m_gradient += -2/n * X * (y - (m_now * X + b_now))
        b_gradient += -2/n * (y - (m_now * X + b_now))
    m = m_now - m_gradient * learning_rate
    b = b_now - b_gradient * learning_rate
    return m, b
```

6 Training Procedure

We initialize $m = 0$ and $b = 0$, set the learning rate $\alpha = 0.0001$, and run gradient descent for 1000 epochs:

```
m = 0
b = 0
learning_rate = 0.0001
epochs = 1000

for i in range(epochs):
    m, b = gradient_descend(m, b, data, learning_rate)

print(m, b)
```

This iterative process gradually reduces the loss and converges to optimal parameters.

7 Interpretation of Parameters

- m indicates how much the house price increases for each additional room.
- b represents the estimated price when the number of rooms is zero.

8 Conclusion

This report demonstrates:

- Implementing linear regression from scratch
- Defining the mean squared error loss function
- Computing gradients and updating parameters via gradient descent
- Training a model to predict Boston house prices based on room count

The model captures the positive correlation between the number of rooms and house prices effectively.