Linear Regression From Scratch: Mathematics and Implementation

(Your Name)

October 3, 2025

1 Feature Matrix (Stick to Notes)

We define the feature matrix $X \in \mathbb{R}^{m \times n}$ as

$$X = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \dots & x_{mn} \end{bmatrix}, \qquad \vec{x}^{(i)} = (x_{i1}, x_{i2}, \dots, x_{in}).$$

Each row of X is a feature vector $\vec{x}^{(i)}$.

2 Prediction Rule

For a sample $\vec{x}^{(i)}$, the prediction is

$$\hat{y}^{(i)} = f(\vec{x}^{(i)}) = \vec{w} \cdot \vec{x}^{(i)} + b,$$

so vectorized:

$$\hat{\vec{y}} = X\vec{w} + b\,\mathbf{1}.$$

3 Error Vector

For each sample,

$$e^{(i)} = \hat{y}^{(i)} - y^{(i)},$$

and the stacked error vector is

$$\vec{e} = \begin{bmatrix} \hat{y}^{(1)} - y^{(1)} \\ \hat{y}^{(2)} - y^{(2)} \\ \vdots \\ \hat{y}^{(m)} - y^{(m)} \end{bmatrix} \in \mathbb{R}^m.$$

4 Gradient of the Multivariable Cost

We use the mean squared error with the $\frac{1}{2}$ convention:

$$J(\vec{w}, b) = \frac{1}{2m} \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)}\right)^2 = \frac{1}{2m} \|\vec{e}\|_2^2.$$

Its gradients are

$$\nabla_{\vec{w}} J = \frac{1}{m} X^\top \vec{e}, \qquad \frac{\partial J}{\partial b} = \frac{1}{m} \sum_{i=1}^{m} e^{(i)}.$$

5 Parameter Updates (Stick to Notes)

With learning rate α:

$$\vec{w} \leftarrow \vec{w} - \alpha \nabla_{\vec{w}} J = \vec{w} - \alpha \cdot \frac{1}{m} X^\top \vec{e}, \qquad b \leftarrow b - \alpha \frac{\partial J}{\partial b} = b - \frac{\alpha}{m} \sum_{i=1}^{m} e^{(i)}.$$

6 Normalization (z-Score)

Before regression we normalize each feature using training statistics:

$$x_j'^{(i)} = \frac{x_j^{(i)} - \mu_j}{\sigma_j}, \quad \mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)}, \quad \sigma_j = \sqrt{\frac{1}{m} \sum_{i=1}^{m} (x_j^{(i)} - \mu_j)^2},$$

and if $\sigma_j = 0$ we set $\sigma_j \leftarrow 1$.

7 Dataset Split (Stick to Notes)

We split the dataset as:

$$\text{Train: } 80\%, \quad \text{Validation: } 10\%, \quad \text{Test: } 10\%.$$

8 Implementation Snippets (Exact Code)

Loading Data and Building $X, \vec{y}$

```
import numpy as np
import pandas as pd

df = pd.read_csv("housing.csv", delim_whitespace=True, header=
    None)
X = df.iloc[:, :-1].to_numpy(dtype=float)
y = df.iloc[:, -1].to_numpy(dtype=float)
```

z-Score Normalization (with zero-std guard)

```
m, n = X.shape
mu = X.mean(axis=0)
sigma = X.std(axis=0, ddof=0)
sigma[sigma == 0] = 1.0
X = (X - mu) / sigma
```

Parameters and Prediction Function

```
w = np.zeros(n, dtype=float)
b = 0.0

def f(X, w, b):
    # Vectorized: \hat{y} = X w + b 1
    return X @ w + b
```

Gradient Step (Implements $X^\top e$ and bias average)

```
def step(X, y, w, b, alpha):
    m = len(y)
    e = f(X, w, b) - y              # e = \hat{y} - y
    grad_w = (X.T @ e) / m          # \nabla_w J = (1/m) X^T e
    grad_b = e.mean()               # dJ/db = (1/m) sum e_i
    return w - alpha * grad_w, b - alpha * grad_b
```

Cost Function $J(\vec{w}, b) = \frac{1}{2m}\|e\|^2$

```
def cost(X, y, w, b):
    e = f(X, w, b) - y
    return 0.5 * np.mean(e ** 2)
```

Batch Gradient Descent with Tolerance

```
alpha = 0.05
max_epochs = 5000
tol = 1e-8
prev_J = None

for epoch in range(1, max_epochs + 1):
    w, b = step(X, y, w, b, alpha)
    J = cost(X, y, w, b)
    if prev_J is not None and abs(prev_J - J) < tol:
        break
    prev_J = J
```

Predictions and Metrics

```python
y_pred = f(X, w, b)
mse = np.mean((y - y_pred) ** 2)

print(f"final_cost J(w,b): {J:.6f}")
print(f"MSE: {mse:.6f}")
print("bias b:", round(b, 6))
print("weights w:", np.round(w, 6))
```