Multivariate Linear Regression via Gradient Descent

From geometry to vectorized implementation with feature scaling, learning-rate tuning, and convergence diagnostics

2019-01-01

Table of contents

This walkthrough builds multivariate linear regression from the ground up, linking the geometric interpretation to a fully vectorized gradient-descent implementation with practical tuning advice.

1 Hypothesis and problem setup

The model assumes that the target value is a linear combination of the input features plus an intercept. Training consists of finding coefficients that minimise prediction error over the dataset.

Mathematics. For features $x_1, \dots, x_n$ and parameters θ, the hypothesis is

$$\hat{y} = \theta_0 + \theta_1 x_1 + \cdots + \theta_n x_n.$$

Training means picking θ that minimise the total squared error between predictions and targets.

---

2 Notation that scales

Representing the dataset with matrices removes explicit loops and exposes efficient linear algebra operations that scale to many features and observations.

Mathematics. With m examples and n features: - Design matrix $X \in \mathbb{R}^{m \times (n+1)}$ with a leading column of ones for the intercept. - Parameter vector $\theta \in \mathbb{R}^{(n+1)\times 1}$. - Targets $y \in \mathbb{R}^{m \times 1}$.

Vectorised prediction is just

$$\hat{y} = X\theta.$$

---

3 Cost function

The mean squared error objective penalises squared residuals. The factor of $\frac{1}{2}$ ensures clean derivatives while leaving the optimum unchanged.

Mathematics.

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 = \frac{1}{2m} \|X\theta - y\|^2.$$

The $\frac{1}{2}$ factor keeps the gradient tidy.

---

4 Gradient of the cost

The gradient points in the direction of steepest increase of the cost. Taking a step opposite to this vector decreases the objective for sufficiently small learning rates.

Mathematics. Taking derivatives of J with respect to θ gives

$$\nabla_\theta J(\theta) = \frac{1}{m} X^\top (X\theta - y).$$

---

5 Gradient descent loop

At each iteration, the algorithm computes residuals, forms the gradient, and updates the parameters. Vectorisation keeps this routine compact and efficient.

Algorithm. Repeat until convergence (or for a fixed budget): 1. $r = X\theta - y$ 2. $g = \frac{1}{m} X^\top r$ 3. $\theta \leftarrow \theta - \alpha g$

---

6 Feature scaling

Normalising features to comparable ranges produces a level cost surface so gradient descent converges faster and avoids numerical issues.

Practices. - Standardisation (z-score): $x_j \leftarrow \frac{x_j - \mu_j}{\sigma_j}$ - Min–max scaling: $x_j \leftarrow \frac{x_j - \min_j}{\max_j - \min_j}$

Statistics are estimated on the training set and reused everywhere else.

---

7 Learning rate selection

Choose a learning rate that yields a monotonic decrease in the cost without oscillations. Monitoring gradient norms and parameter updates helps confirm that the optimisation is settling.

Diagnostics. Cost curves should drop smoothly and flatten near convergence, while gradient norms and parameter deltas shrink toward zero.

---

8 Reference implementation

Putting the pieces together reinforces the mechanics.

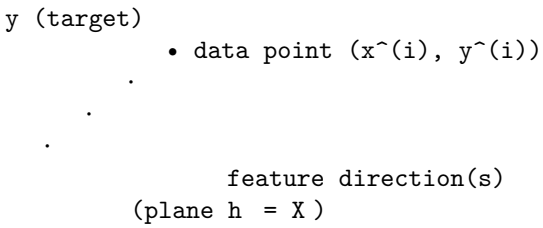
```
import numpy as np

# Synthetic data: y  3 + 2*x1 - 0.5*x2 + noise
rng = np.random.default_rng(42)
m = 400
x1 = rng.normal(50, 10, size=m)
x2 = rng.normal(5, 2, size=m)
y = 3 + 2*x1 - 0.5*x2 + rng.normal(0, 3, size=m)

X = np.column_stack([np.ones(m), x1, x2])

# Feature scaling (skip bias column)
mu = X[:, 1:].mean(axis=0)
std = X[:, 1:].std(axis=0)
X[:, 1:] = (X[:, 1:] - mu) / std
```

```
theta = np.zeros(X.shape[1])
alpha = 0.1
epochs = 5000


def cost(X, y, th):
    r = X @ th - y
    return 0.5 / len(y) * (r @ r)

hist = []
for it in range(epochs):
    r = X @ theta - y
    grad = (X.T @ r) / m
    theta -= alpha * grad
    if it % 100 == 0:
        hist.append(cost(X, y, theta))

print("theta (scaled):", theta)
print("final cost:", cost(X, y, theta))
```

---

9 Geometry of the model

In two dimensions the hypothesis is a line; in higher dimensions it becomes a hyperplane. The squared-error objective measures vertical deviations of data points from this hyperplane.



The hyperplane $h_\theta = X\theta$ cuts through the cloud of points; vectorisation simply evaluates that plane for all rows simultaneously.

---

10 Common pitfalls

- Missing intercept column → biased predictions.
- Unscaled features → slow or unstable convergence.
- Oversized learning rate → cost oscillation or divergence.
- Scaling statistics computed on full data → leakage.
- Stopping too early → cost still trending down.

---

11 Normal equation

The closed form is like solving a circuit by hand: for small, well-behaved systems it is instant, but the method becomes brittle as complexity rises.

Mathematics.

$$\theta^\star = (X^\top X)^{-1} X^\top y.$$

---

12 L_2 regularisation

Ridge adds a penalty proportional to the squared magnitude of the coefficients, shrinking them toward zero and reducing sensitivity to correlated features.

Mathematics.

$$J_\lambda(\theta) = \frac{1}{2m}\|X\theta - y\|^2 + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2,$$

$$\nabla J_\lambda(\theta) = \frac{1}{m}X^\top(X\theta - y) + \frac{\lambda}{m}\begin{bmatrix} 0 \\ \theta_{1:n} \end{bmatrix}.$$

---

13 Deployment checklist

- ☐ Build X with a bias column of ones.
- ☐ Split data into train/validation/test.
- ☐ Standardise features using train statistics only.
- ☐ Initialise θ (zeros or small random values).
- ☐ Tune the learning rate via a quick sweep and monitor cost.
- ☐ Stop when cost plateaus or gradients are tiny.
- ☐ Evaluate on validation, then test.
- ☐ Persist scaling parameters and θ for inference.

---

14 Where to go next

- Implement stochastic or mini-batch gradient descent with momentum/Adam and compare convergence speed.
- Explore feature engineering (polynomials, interactions) and apply regularisation to manage complexity.
- Extend the notebook to include diagnostics such as residual plots and R^2 on validation data.