



Supervised Learning - Linear Regression

Simple and Multiple Regression, Cost Functions, Gradient Descent

Sarwan Ali

Department of Computer Science
Georgia State University

 Linear Regression Fundamentals 

Today's Learning Journey

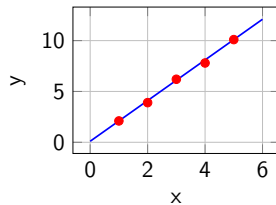
- 1 Introduction to Linear Regression
- 2 Simple Linear Regression
- 3 Multiple Linear Regression
- 4 Cost Functions
- 5 Gradient Descent
- 6 Analytical Solution
- 7 Feature Scaling and Normalization
- 8 Model Evaluation and Performance Metrics
- 9 Assumptions and Limitations
- 10 Practical Implementation
- 11 Summary and Next Steps

What is Linear Regression?

Definition: Linear regression is a fundamental supervised learning algorithm that models the relationship between a dependent variable and one or more independent variables using a linear equation.

Key Characteristics:

- **Supervised Learning** - Uses labeled training data
- **Regression** - Predicts continuous values
- **Linear** - Assumes linear relationship
- **Parametric** - Has fixed number of parameters



Applications of Linear Regression

Real Estate

- House price prediction
- Property valuation

Finance

- Stock price analysis
- Risk assessment

Healthcare

- Drug dosage optimization
- Disease progression

Manufacturing

- Quality control
- Production optimization

Simple Linear Regression

Mathematical Model:

$$y = \beta_0 + \beta_1 x + \epsilon$$

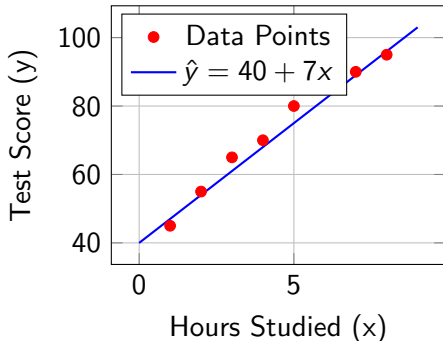
$$\hat{y} = \beta_0 + \beta_1 x$$

Where:

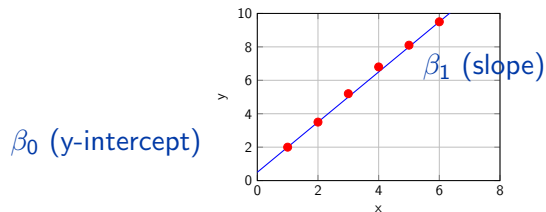
- y = dependent variable (target)
- x = independent variable (feature)
- β_0 = y-intercept (bias)
- β_1 = slope (weight)
- ϵ = error term
- \hat{y} = predicted value

(1)

(2)



Geometric Interpretation



The goal is to find the line that **best fits** the data by minimizing the distance between predicted and actual values.

Multiple Linear Regression

Extension to Multiple Features:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n + \epsilon \quad (3)$$

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n \quad (4)$$

Matrix Form:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon} \quad (5)$$

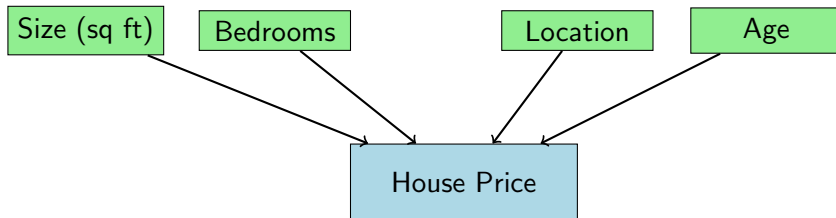
$$\hat{\mathbf{y}} = \mathbf{X}\boldsymbol{\beta} \quad (6)$$

Where:

- \mathbf{y} = target vector ($m \times 1$)
- \mathbf{X} = feature matrix ($m \times (n + 1)$) with bias column
- $\boldsymbol{\beta}$ = parameter vector ($(n + 1) \times 1$)
- m = number of samples, n = number of features

Multiple Regression Example

House Price Prediction:



Model:

$$\text{Price} = \beta_0 + \beta_1 \cdot \text{Size} + \beta_2 \cdot \text{Bedrooms} + \beta_3 \cdot \text{Location} + \beta_4 \cdot \text{Age}$$

Matrix Representation

Training Data Structure:

$$\mathbf{X} = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} & \cdots & x_n^{(m)} \end{bmatrix}, \mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}, \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_n \end{bmatrix}$$

Key Points:

- First column of \mathbf{X} is all 1's (bias term)
- Each row represents one training example
- Each column (except first) represents one feature
- $\boldsymbol{\beta}$ contains all parameters to be learned

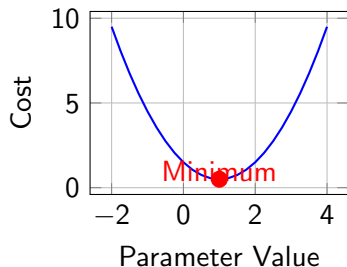
Why Do We Need Cost Functions?

Purpose: Measure how well our model fits the training data

Goal: Find parameters β that minimize the cost

Intuition:

- Good fit = Low cost
- Poor fit = High cost
- Cost function guides learning



Mean Squared Error (MSE)

Most Common Cost Function for Linear Regression:

$$J(\beta) = \frac{1}{2m} \sum_{i=1}^m (h_{\beta}(x^{(i)}) - y^{(i)})^2 \quad (7)$$

$$= \frac{1}{2m} \sum_{i=1}^m (\hat{y}^{(i)} - y^{(i)})^2 \quad (8)$$

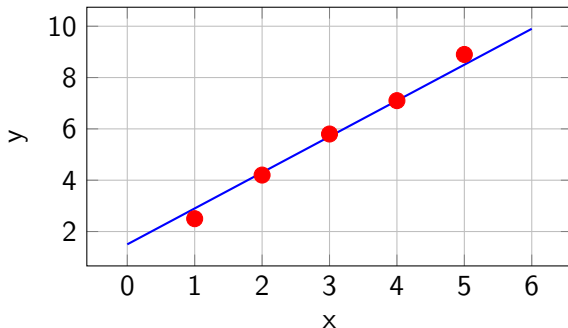
Matrix Form:

$$J(\beta) = \frac{1}{2m} (\mathbf{X}\beta - \mathbf{y})^T (\mathbf{X}\beta - \mathbf{y})$$

Properties:

- Always non-negative
- Convex function (single global minimum)
- Differentiable everywhere
- Penalizes large errors more than small ones

Understanding MSE Geometrically



MSE = $\frac{1}{2m} \times (\text{Sum of squared vertical distances})$
The factor $\frac{1}{2}$ simplifies derivative calculations.

Other Cost Functions

Cost Function	Formula	Properties
Mean Absolute Error	$\frac{1}{m} \sum_{i=1}^m y^{(i)} - \hat{y}^{(i)} $	Robust to outliers
Root Mean Squared Error	$\sqrt{\frac{1}{m} \sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)})^2}$	Same units as target
Huber Loss	Mixed L1/L2	$\begin{cases} \frac{1}{2}e^2 & e \leq \delta \\ \delta e - \frac{1}{2}\delta^2 & e > \delta \end{cases}$

Why MSE is Popular:

- Mathematically convenient (differentiable)
- Convex optimization landscape
- Statistical interpretation (MLE under Gaussian noise)

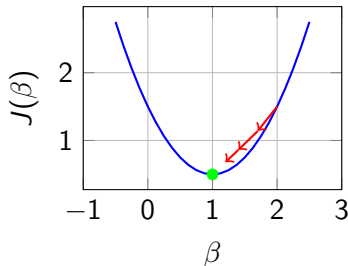
What is Gradient Descent?

Definition: An iterative optimization algorithm to find the minimum of a cost function.

Key Idea:

- Start with random parameters
- Compute gradient (slope)
- Move in opposite direction
- Repeat until convergence

Analogy: Rolling ball down a hill



Gradient Descent Algorithm

Update Rule:

$$\beta_j := \beta_j - \alpha \frac{\partial}{\partial \beta_j} J(\beta)$$

For Linear Regression:

$$\frac{\partial}{\partial \beta_j} J(\beta) = \frac{1}{m} \sum_{i=1}^m (h_{\beta}(x^{(i)}) - y^{(i)}) x_j^{(i)} \quad (9)$$

Simultaneous Update:

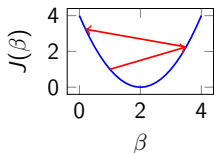
$$\beta_0 := \beta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\beta}(x^{(i)}) - y^{(i)}) \quad (10)$$

$$\beta_j := \beta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\beta}(x^{(i)}) - y^{(i)}) x_j^{(i)} \quad (11)$$

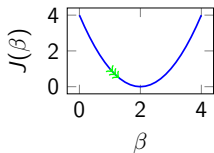
Where α is the [learning rate](#).

Learning Rate α

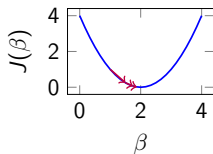
Too Large: May overshoot minimum



Too Small: Slow convergence



Just Right: Efficient convergence



Choosing α :

- Start with 0.01
- Try: 0.001, 0.01, 0.1, 1
- Plot $J(\beta)$ vs iterations
- Use adaptive methods

Gradient Descent Variants

Type	Description	Pros/Cons
Batch GD	Uses entire dataset per update	Stable convergence Slow for large data
Stochastic GD	Uses one sample per update	Fast updates Noisy convergence
Mini-batch GD	Uses small batches (e.g., 32-256 samples)	Good balance Memory efficient

Mini-batch Update:

$$\beta_j := \beta_j - \alpha \frac{1}{|B|} \sum_{i \in B} (h_{\beta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

where B is a mini-batch of training examples.

Convergence Criteria

When to Stop Gradient Descent?

① Cost Function Convergence:

$$|J(\beta)^{(t)} - J(\beta)^{(t-1)}| < \epsilon$$

② Parameter Convergence:

$$\|\beta^{(t)} - \beta^{(t-1)}\| < \epsilon$$

③ Gradient Magnitude:

$$\|\nabla J(\beta)\| < \epsilon$$

④ Maximum Iterations: Fixed number of iterations (e.g., 1000)

Typical threshold: $\epsilon = 10^{-6}$ or 10^{-8}

Normal Equation

Closed-form Solution: For linear regression, we can solve analytically!

Derivation:

$$J(\beta) = \frac{1}{2m}(\mathbf{X}\beta - \mathbf{y})^T(\mathbf{X}\beta - \mathbf{y}) \quad (12)$$

$$\frac{\partial J}{\partial \beta} = \frac{1}{m}\mathbf{X}^T(\mathbf{X}\beta - \mathbf{y}) = 0 \quad (13)$$

$$\mathbf{X}^T\mathbf{X}\beta = \mathbf{X}^T\mathbf{y} \quad (14)$$

$$\beta = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \quad (15)$$

Normal Equation:

$$\boxed{\beta = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}}$$

Gradient Descent vs Normal Equation

Aspect	Gradient Descent	Normal Equation
Speed	Iterative, may be slow	Direct computation
Scalability	Works with large n	Slow for large n ($O(n^3)$)
Complexity	$O(kn^2m)$	$O(n^3)$
Applicability	Works for all algorithms	Linear regression only
Matrix Inverse	Not needed	Requires $(\mathbf{X}^T \mathbf{X})^{-1}$
Feature Scaling	Recommended	Not necessary

Rule of Thumb:

- $n \leq 10^4$: Use Normal Equation
- $n > 10^4$: Use Gradient Descent
- Non-invertible $\mathbf{X}^T \mathbf{X}$: Use Gradient Descent

Why Feature Scaling Matters

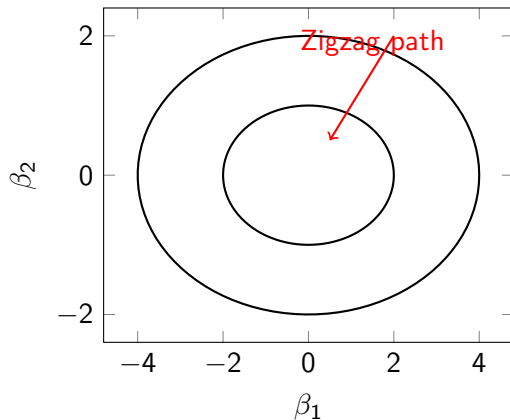
Problem: Different scales can hurt convergence

Example:

- House size: 500-5000 sq ft
- Number of bedrooms: 1-5
- Age: 0-100 years

Without Scaling:

- Gradient descent oscillates
- Slow convergence
- Numerical instability



Feature Scaling Techniques

1. Min-Max Normalization (Scaling to [0,1]):

$$X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

2. Standardization (Z-score normalization):

$$X_{scaled} = \frac{X - \mu}{\sigma}$$

3. Robust Scaling:

$$X_{scaled} = \frac{X - \text{median}}{Q_{75} - Q_{25}}$$

Method	When to Use	Properties
Min-Max	Known bounds	Preserves relationships
Standardization	Gaussian distribution	Mean=0, Std=1
Robust	Outliers present	Less sensitive to outliers

Evaluation Metrics for Regression

1. Mean Squared Error (MSE):

$$MSE = \frac{1}{m} \sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)})^2$$

2. Root Mean Squared Error (RMSE):

$$RMSE = \sqrt{\frac{1}{m} \sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)})^2}$$

3. Mean Absolute Error (MAE):

$$MAE = \frac{1}{m} \sum_{i=1}^m |y^{(i)} - \hat{y}^{(i)}|$$

4. R-squared (Coefficient of Determination):

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} = 1 - \frac{\sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)})^2}{\sum_{i=1}^m (y^{(i)} - \bar{y})^2}$$

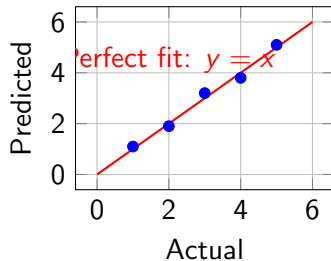
Understanding R-squared

Interpretation:

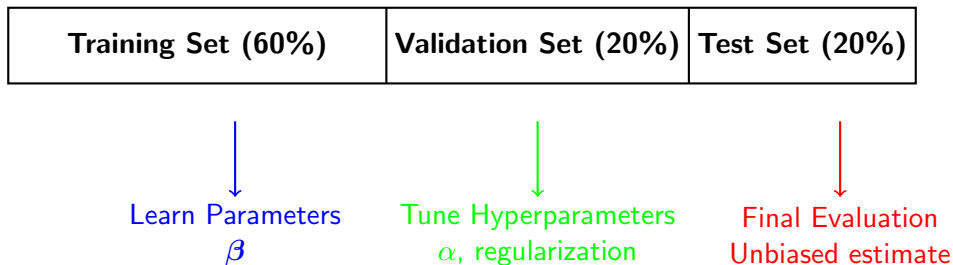
- $R^2 = 1$: Perfect fit
- $R^2 = 0$: No better than mean
- $R^2 < 0$: Worse than mean
- $0 < R^2 < 1$: Partial explanation

Example:

- $R^2 = 0.8$: Model explains 80% of variance
- $R^2 = 0.3$: Model explains 30% of variance



Train-Validation-Test Split

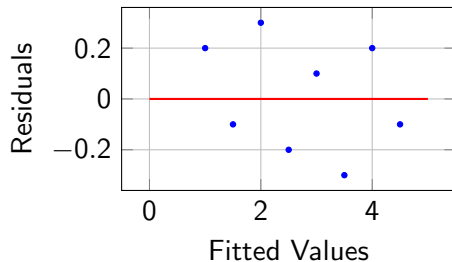


Why Three Sets?

- **Training:** Learn model parameters
- **Validation:** Select best hyperparameters
- **Test:** Evaluate final model performance

Linear Regression Assumptions

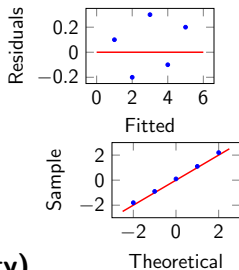
1. **Linearity:** Relationship between X and y is linear
2. **Independence:** Observations are independent
3. **Homoscedasticity:** Constant variance of residuals
4. **Normality:** Residuals are normally distributed
5. **No Multicollinearity:** Features are not highly correlated



Checking Assumptions

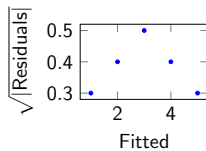
Diagnostic Plots:

1. Residuals vs Fitted

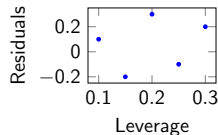


2. Q-Q Plot (Normality)

3. Scale-Location Plot



4. Leverage Plot



Common Problems and Solutions

Problem	Symptoms	Solutions
Non-linearity	Curved residual patterns	Polynomial features, transforms
Heteroscedasticity	Fan-shaped residuals	Log transform, weighted regression
Multicollinearity	High VIF values	Remove features, PCA, regularization
Outliers	High leverage points	Robust regression, outlier removal
Non-normality	Skewed Q-Q plot	Transform target variable

Variance Inflation Factor (VIF):

$$VIF_j = \frac{1}{1 - R_j^2}$$

Rule of thumb: $VIF > 10$ indicates multicollinearity

Using Scikit-learn:

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Prepare data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)

# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train model
model = LinearRegression()
model.fit(X_train_scaled, y_train)

# Make predictions
y_pred = model.predict(X_test_scaled)

# Evaluate
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"MSE: -{mse:.2 f}")
print(f"R^2: -{r2:.2 f}")
print(f"Coefficients: -{model.coef_}")
print(f"Intercept: -{model.intercept_:.2 f}")
```

Manual Implementation

Gradient Descent from Scratch:

```
import numpy as np

def gradient_descent(X, y, learning_rate=0.01, num_iterations=1000):
    m, n = X.shape

    # Initialize parameters
    theta = np.zeros(n)
    cost_history = []

    for i in range(num_iterations):
        # Forward pass
        h = X @ theta

        # Compute cost
        cost = (1/(2*m)) * np.sum((h - y)**2)
        cost_history.append(cost)

        # Compute gradients
        gradients = (1/m) * X.T @ (h - y)

        # Update parameters
        theta = theta - learning_rate * gradients

        # Check convergence
        if i > 0 and abs(cost_history[i-1] - cost) < 1e-8:
            break

    return theta, cost_history

# Usage
theta, costs = gradient_descent(X_train, y_train)
```

Key Takeaways

What We Learned:

- 1 **Linear Regression** models linear relationships between features and targets
- 2 **Cost Functions** (MSE) measure model performance
- 3 **Gradient Descent** iteratively optimizes parameters
- 4 **Normal Equation** provides analytical solution
- 5 **Feature Scaling** improves convergence
- 6 **Evaluation Metrics** assess model quality
- 7 **Assumptions** must be checked for validity

When to Use Linear Regression:

- Linear relationship between features and target
- Interpretability is important
- Baseline model for comparison
- Small to medium-sized datasets

Limitations and Extensions

Limitations:

- Assumes linear relationships
- Sensitive to outliers
- May overfit with many features
- Requires assumption validation

Extensions and Improvements:

- **Polynomial Regression:** Non-linear relationships
- **Regularization:** Ridge, Lasso, Elastic Net
- **Robust Regression:** Handle outliers
- **Generalized Linear Models:** Different distributions

Next Topics:

- Regularization techniques
- Classification algorithms
- Model selection and validation

Practice Problems

Try These Exercises:

- 1 Implement gradient descent for simple linear regression
- 2 Compare Normal Equation vs Gradient Descent on different dataset sizes
- 3 Analyze the effect of learning rate on convergence
- 4 Create diagnostic plots for assumption checking
- 5 Build a house price prediction model using multiple features

Datasets to Practice:

- Boston Housing Dataset
- California Housing Dataset
- Auto MPG Dataset

Thank You!

? Questions?