Introduction to Linear Regression

A Comprehensive Guide to Theory and Practice

Table of Contents

---

1. Introduction

Linear regression is one of the most fundamental and widely-used algorithms in machine learning and statistics. It serves as the foundation for understanding more complex models and provides powerful predictive capabilities when data exhibits linear relationships.

Why Linear Regression Matters: - **Simplicity**: Easy to understand and interpret - **Efficiency**: Computationally efficient for large datasets - **Foundation**: Basis for many advanced algorithms - **Interpretability**: Coefficients have clear meaning - **Widespread Use**: Applied across finance, healthcare, marketing, and more

---

2. What is Linear Regression?

Linear regression is a **supervised machine-learning algorithm** that learns from labeled datasets to map input features to output predictions using a linear function.

Core Concept

The algorithm assumes a **linear relationship** between inputs and outputs: - As the input changes, the output changes at a constant rate - This relationship is represented by a straight line (or hyperplane in higher dimensions)

Example: Exam Score Prediction

Imagine predicting a student's exam score based on study hours: - **Independent variable** (input): Hours studied - **Dependent variable** (output): Exam score - **Observation**: More study hours → higher scores - **Goal**: Find the linear function that best captures this relationship

---

3. Real-World Examples

Regression Problems: Estimating Real Numbers

Fish Weight Estimation - Input: Fish dimensions [length, width] measured by camera - Output: Estimated weight - Application: Automated sorting in fish processing

Agricultural Production - Input: Rainfall, sunshine hours, pest levels - Output: Annual corn farm production - Application: Crop yield forecasting

Medical Applications - Input: Insulin dose, patient characteristics - Output: Blood glucose level - Application: Diabetes management

Market Pricing - Input: Product cost, quality metrics, transportation distance - Output: Market price - Application: Pricing optimization

---

4. The Supervised Learning Framework

Components

`Training Data: {(x₁, y₁), (x₂, y₂), ..., (xₙ, yₙ)} ↓ Learner ↓ Model h: X → Y ↓ Prediction for new input`

Terminology

Input Features (x_i) - Also called: attributes, variables, predictors - Types: - **Numeric**: Continuous values (length: 0.2, 1.3, ...) - **Categorical**: Discrete categories (blood type: A/B/AB/O) - **Ordinal**: Ranked categories (difficulty: easy/normal/hard)

Output Labels (y_i) - Also called: target variable, ground-truth label - For regression: Real-valued numbers ($\mathbb{R}$)

The I.I.D. Assumption

Critical Requirement: Training data must be **independently and identically distributed**

1. **Independent**: Each example (x_i, y_i) is drawn independently from probability distribution P(x, y)
2. **Identical**: All examples come from the same distribution P(x, y)

This assumption extends to test data, ensuring our model generalizes well.

5. Mathematical Formulation

Simple Linear Regression

For one independent variable:

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

Where: - **y**: Predicted value (dependent variable) - **x**: Input value (independent variable)
- **β_0**: Intercept (value of y when x = 0) - **β_1**: Slope (change in y per unit change in x)

Multiple Linear Regression

For multiple independent variables:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_k x_k$$

Vector Notation

For a d-dimensional input vector $\mathbf{x} \in \mathbb{R}^d$:

$$h_w(x) = x^\top w = w_1 x_1 + w_2 x_2 + \ldots + w_a x_a$$

Where $\mathbf{w} \in \mathbb{R}^d$ is the parameter vector to be learned.

Augmented Form

To include the intercept elegantly, we augment the input:

$$x = [1, x_1, x_2, \ldots, x_a]^\top \quad w = [w_0, w_1, w_2, \ldots, w_a]^\top$$

$$h(x) = x^\top w = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_a x_a$$

---

6. The Best-Fit Line

Goal

Find the line that **minimizes the difference** between: - Actual data points (observed values) - Predicted values from our model

Visual Intuition

For the fish weight example with features [length, width]:

```
Weight | ● Actual data points 1.0| ● ● | ● ● 0.8| ● ● |●● Fitted plane 0.6|_____________________ 60 70 80 Length Width →
```

The "plane" in 3D space represents our linear model fitting the data points.

Residuals

The **residual** for each data point is:

$$\text{Residual} = y_i - \hat{y}_i$$

Where: - y_i: Actual observed value - $\hat{y}_i$: Predicted value from our model

7. Loss Functions and Optimization

What is a Loss Function?

A loss function $L(\hat{y}, y)$ measures how "close" our prediction $\hat{y}$ is to the actual value y.

Common Loss Functions

1. Squared Loss (L_2 Loss)

$$L_2(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$$

Properties: - Smooth and differentiable everywhere - Heavily penalizes large errors (quadratic growth) - Most commonly used for linear regression

Objective Function: $\min_w \frac{1}{2n} \sum_{i=1}^{n} (x_i^T w - y_i)^2$

2. Absolute Loss (L_1 Loss)

$$L_1(\hat{y}, y) = |\hat{y} - y|$$

Properties: - More robust to outliers (linear growth) - Not differentiable at zero - Requires specialized solvers

Objective Function: $\min_w \frac{1}{n} \sum_{i=1}^{n} |x_i^T w - y_i|$

3. L_∞ Loss (Maximum Error)

$$L_\infty(\hat{y}, y) = \max_i |x_i^T w - y_i|$$

Properties: - Controls worst-case error - Minimizes the maximum deviation - Useful when all predictions must be accurate

Empirical Risk Minimization (ERM)

General framework:

$$\min_{w \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} L(x_i^T w, y_i)$$

We average the loss over all training examples and find parameters that minimize this average.

---

8. Matrix Formulation

Notation

Define the **data matrix** X and **output vector** y:

$$X = \begin{bmatrix} — & x_1^T & — \\ & \vdots & \\ — & x_n^T & — \end{bmatrix} \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

> $X \in \mathbb{R}^{n \times d}$ $\quad$ $y \in \mathbb{R}^n$

Predictions

The prediction vector:

``` ŷ = Xw = [— x₁⊤ —] [w₁] [— x₂⊤ —] [w₂] [ ⋮ ] [ ⋮] [— xₙ⊤ —] [wₐ]

> ŷ ∈ ℝⁿ

```

L₂ Loss in Matrix Form

Using the L₂ norm $\|a\|_2^2 = a^\top a$:

`J(w) = (1/2n) ‖Xw - y‖₂² = (1/2n) (Xw - y)T(Xw - y)`

Goal: Find w* that minimizes J(w).

---

9. Solving Linear Regression

Approach 1: Direct (Analytical) Solution

For L₂ loss, we can solve directly using calculus.

Step 1: Compute the gradient

`∇J(w) = (1/n) XT(Xw - y)`

Step 2: Set gradient to zero

`XT(Xw - y) = 0 XTXw = XTy`

Step 3: Solve for w (if X⊤X is invertible)

`w* = (XTX)⁻¹XTy`

This is called the **Normal Equation** or **Ordinary Least Squares (OLS)** solution.

Advantages: - Closed-form solution - No hyperparameters to tune - Exact answer (up to numerical precision)

Disadvantages: - Requires matrix inversion: O(d³) complexity - Fails if X⊤X is singular - Not suitable for very large d

Approach 2: Gradient Descent (Iterative)

When direct solution is impractical, use iterative optimization.

Algorithm:

`1. Initialize: w⁽⁰⁾ = 0ₐ (or small random values) 2. For t = 0, 1, 2, ... a. Compute gradient: ∇J(w⁽ᵗ⁾) = (1/n) XT(Xw⁽ᵗ⁾ - y) b. Update: w⁽ᵗ⁺¹⁾ = w⁽ᵗ⁾ - η∇J(w⁽ᵗ⁾) 3. Stop when ‖∇J(w⁽ᵗ⁾)‖ < ε or max iterations reached`

Where: - **η**: Step size (learning rate) - **ε**: Convergence tolerance

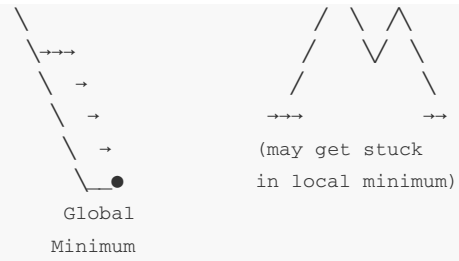
Visualization:

For convex functions (like L₂ loss), gradient descent always converges to the global minimum:

``` Convex Loss Surface Non-Convex Loss Surface

> \ ⋀

```




```

**Why Convexity Matters:** - Convex: Any local minimum is global → guaranteed convergence - Non-convex: May get stuck in local minima

---

# 10. Multiple Loss Functions Comparison

## Loss Function Families

The **$L_p$ loss**: $\|Xw - y\|_p^p$ where $0 \leq p \leq \infty$

## Comparison of Different p Values

| p value | Convex | Smooth | Characteristics | Use Case | |---------|--------|--------|-----------------|----------| | p = 0.1 | ✗ | ✗ | Very robust to outliers | Noisy data | | p = 1 ($L_1$) | ✓ | ✗ | Robust to outliers | Moderate outliers | | p = 2 ($L_2$) | ✓ | ✓ | Efficient optimization | Standard regression | | p = 5 | ✓ | ✓ | Sensitive to outliers | Clean data | | p = ∞ ($L_\infty$) | ✓ | ✗ | Minimizes worst error | Critical applications |

## Visual Comparison

Loss function shape for single data point (x-axis shows $\hat{y} - y$):

`p = 5: U-shaped, steep sides p = 2: Parabola (standard) p = 1: V-shaped p = 0.1: Nearly flat near zero, sharp elsewhere`

## Outlier Sensitivity Example

Dataset with one outlier:

``` y 30| ● ← Outlier | 10| |●●●●●●●●●●●●●●●●●●●● 0|___ x

$L_1$ loss: ———— (ignores outlier) $L_2$ loss: ———╱ (chases outlier) $L_\infty$ loss: ——╱ (heavily influenced) ```

## Trade-offs

**Convex Losses (p ∈ [1, ∞])** - ✓ Efficient optimization - ✓ Global minimum guaranteed - ✗ Sensitive to outliers (especially large p)

**Robust Losses (p ∈ [0, 1))** - ✓ Resistant to outliers - ✓ Slow growth in loss - ✗ Non-convex: difficult to optimize - ✗ May get stuck in local minima

## Solving Other Losses

### $L_1$ Loss as Linear Programming

Not differentiable at zero, but can reformulate:

``` $\min_{w,\delta} (1/n) \Sigma_i \delta_i$
```

subject to: $\delta_i \geq 0$ for all i $x_i^\top w - y_i \leq \delta_i$ for all i $y_i - x_i^\top w \leq \delta_i$ for all i ```

This is a **Linear Programming (LP)** problem → efficient solvers available.

L∞ Loss as Linear Programming

``` $\min_{w,\delta} \delta$

subject to: $\delta \geq 0$ $Xw - y \leq \delta \cdot 1_n$ $y - Xw \leq \delta \cdot 1_n$ ```

Also an LP problem.

---

# 11. Assumptions of Linear Regression

For reliable results, linear regression requires several assumptions:

## 1. Linearity

The relationship between X and Y is linear.

``` Valid: $y \approx \beta_0 + \beta_1 x$

Invalid: $y \approx \beta_0 + \beta_1 x^2$ (non-linear) ```

**Check**: Plot residuals vs. fitted values. Look for random scatter.

## 2. Independence of Errors

Errors in predictions should not affect each other.

**Violation Example**: Time series data where errors are correlated.

## 3. Homoscedasticity (Constant Variance)

Errors should have equal spread across all input values.

```
Homoscedastic (Good): Heteroscedastic (Bad): ε | ● ● ε | ●●● | ● ● ● | ●●●● |● ● ● ● | ●●●●● 0 |
──────── x 0 |────────── x
```

Check: Plot residuals vs. fitted values. Look for constant width.

4. Normality of Errors

Errors should follow a normal (bell-shaped) distribution.

Check: Q-Q plot or histogram of residuals.

5. No Multicollinearity

(For multiple regression) Independent variables shouldn't be highly correlated.

Problem: Makes coefficient estimates unstable.

Check: Variance Inflation Factor (VIF) < 10

6. No Autocorrelation

Errors shouldn't show repeating patterns, especially in time-based data.

Check: Durbin-Watson statistic ≈ 2

7. Additivity

The total effect on Y is the sum of individual effects from each X.

```
y = β₀ + β₁x₁ + β₂x₂

Not: y = β₀ + β₁x₁ · x₂ (interaction term)
```

---

12. Types of Linear Regression

Simple Linear Regression

One independent variable:

$$\hat{y} = \theta_0 + \theta_1 x$$

Example: Predicting salary (y) from years of experience (x)

When to Use: - Single predictor available - Exploring basic relationships - Baseline model

Multiple Linear Regression

Multiple independent variables:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_k x_k$$

Example: Predicting house price from: - x_1: Square footage - x_2: Number of bedrooms - x_3: Age of house - x_4: Distance to city center

When to Use: - Multiple factors influence outcome - Real-world problems (usually multivariate) - Better predictive accuracy

Polynomial Regression

Still "linear" in parameters, but non-linear in features:

$$\hat{y} = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$$

Transform: Create new features $[x, x^2, x^3] \rightarrow$ Apply linear regression

When to Use: Curved relationships between variables

---

13. Regularization Techniques

Why Regularize?

Problem: Complex models with many features can **overfit**: - Fit training data perfectly - Poor performance on new data

Solution: Add penalty term to discourage large coefficients

Ridge Regression (L_2 Regularization)

Objective: $J(\theta) = (1/2m)\ \Sigma_i(\hat{y}_i - y_i)^2 + \lambda\Sigma_j\theta_j^2$ (MSE | L_2 penalty)

Effect: - Shrinks all coefficients toward zero - Keeps all features - λ controls regularization strength

When to Use: - Multicollinearity present - Many correlated features - Want to keep all features

Lasso Regression (L₁ Regularization)

Objective: `J(θ) = (1/2m) Σᵢ(ŷᵢ - yᵢ)² + λΣⱼ|θⱼ| └──── MSE ────┘ └─ L₁ penalty ─┘`

Effect: - Shrinks some coefficients to exactly zero - Performs **feature selection** - Produces sparse models

When to Use: - Feature selection needed - Believe many features irrelevant - Want interpretable model

Elastic Net

Objective: `J(θ) = MSE + αλΣⱼ|θⱼ| + ½(1-α)λΣⱼθⱼ² └─ L₁ ─┘ └─── L₂ ────┘`

Combines both penalties: - α = 0: Pure Ridge - α = 1: Pure Lasso - 0 < α < 1: Mix of both

When to Use: - Grouped variable selection - Best of both worlds

Choosing λ (Regularization Strength)

Use **cross-validation**:

`Small λ → Less regularization → May overfit Large λ → More regularization → May underfit`

Optimal λ minimizes validation error.

---

14. Evaluation Metrics

Mean Squared Error (MSE)

`MSE = (1/n) Σᵢ(yᵢ - ŷᵢ)²`

Properties: - Always non-negative - Same units as y² - Heavily penalizes large errors

Root Mean Squared Error (RMSE)

`RMSE = √MSE = √[(1/n) Σᵢ(yᵢ - ŷᵢ)²]`

Properties: - Same units as y (more interpretable than MSE) - Standard deviation of prediction errors

Mean Absolute Error (MAE)

`MAE = (1/n) Σᵢ|yᵢ - ŷᵢ|`

Properties: - Same units as y - Less sensitive to outliers than MSE - All errors weighted equally

R² (Coefficient of Determination)

``` R² = 1 - (RSS/TSS)

where: RSS = Σᵢ(yᵢ - ŷᵢ)² Residual Sum of Squares TSS = Σᵢ(yᵢ - ȳ)² Total Sum of Squares ```

**Interpretation**: - R² = 1: Perfect fit - R² = 0: Model no better than predicting mean - R² < 0: Model worse than mean (rare with linear regression)

**Meaning**: Proportion of variance in y explained by the model

## Adjusted R²
```

``` Adjusted $R^2 = 1 - [(1 - R^2)(n - 1)/(n - k - 1)]$

where: n = number of observations k = number of predictors ```

**Why Needed**: - $R^2$ always increases when adding features (even irrelevant ones) - Adjusted $R^2$ penalizes unnecessary predictors - Better for model comparison

**Interpretation**: - Accounts for model complexity - Can decrease when adding irrelevant features - Use for comparing models with different numbers of predictors

### Which Metric to Use?

| Metric | Use When | Advantage | |--------|----------|-----------| | MSE | Outliers important | Penalizes large errors heavily | | RMSE | Need interpretable scale | Same units as target | | MAE | Robust metric needed | Less sensitive to outliers | | $R^2$ | Compare models | Standardized (0 to 1) | | Adj $R^2$ | Multiple predictors | Prevents overfitting |

---

## 15. Python Implementation

### Complete Example

```python

# 1. Import Libraries

import numpy as np import matplotlib.pyplot as plt from sklearn.linear_model import LinearRegression from sklearn.model_selection import train_test_split from sklearn.metrics import mean_squared_error, r2_score

# 2. Generate Dataset

np.random.seed(42) X = np.random.rand(100, 1) * 100 # 100 samples, 1 feature y = 3.5 * X + np.random.randn(100, 1) * 20 # y = 3.5x + noise

# 3. Split Data

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

# 4. Create and Train Model

model = LinearRegression() model.fit(X_train, y_train)

# 5. Make Predictions

y_train_pred = model.predict(X_train) y_test_pred = model.predict(X_test)

# 6. Evaluate Model

train_mse = mean_squared_error(y_train, y_train_pred) test_mse = mean_squared_error(y_test, y_test_pred) train_r2 = r2_score(y_train, y_train_pred) test_r2 = r2_score(y_test, y_test_pred)

print(f"Coefficients: {model.coef_[0][0]:.4f}") print(f"Intercept: {model.intercept_[0]:.4f}") print(f"\nTraining MSE: {train_mse:.4f}") print(f"Testing MSE: {test_mse:.4f}") print(f"Training $R^2$: {train_r2:.4f}") print(f"Testing $R^2$: {test_r2:.4f}")

# 7. Visualize Results

plt.figure(figsize=(10, 6)) plt.scatter(X_train, y_train, color='blue', label='Training Data', alpha=0.6) plt.scatter(X_test, y_test, color='green', label='Test Data', alpha=0.6) plt.plot(X, model.predict(X), color='red', linewidth=2, label='Regression Line') plt.xlabel('X') plt.ylabel('y') plt.title('Linear Regression: Training and Test Data') plt.legend() plt.grid(True, alpha=0.3) plt.show() ```

### Multiple Linear Regression

```python from sklearn.datasets import fetch_california_housing from sklearn.linear_model import LinearRegression, Ridge, Lasso import pandas as pd

# 1. Load Dataset

housing = fetch_california_housing() X = housing.data y = housing.target feature_names = housing.feature_names

# 2. Split Data

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

# 3. Train Multiple Models

models = { 'OLS': LinearRegression(), 'Ridge': Ridge(alpha=1.0), 'Lasso': Lasso(alpha=0.1) }

results = {} for name, model in models.items(): model.fit(X_train, y_train) y_pred = model.predict(X_test) results[name] = { 'MSE': mean_squared_error(y_test, y_pred), '$R^2$': r2_score(y_test, y_pred) }

# 4. Compare Results

results_df = pd.DataFrame(results).T print(results_df)
```

5. Feature Importance (coefficients)

coefficients = pd.DataFrame({ 'Feature': feature_names, 'OLS': models['OLS'].coef_, 'Ridge': models['Ridge'].coef_, 'Lasso': models['Lasso'].coef_ }) print("\nCoefficients:") print(coefficients.sort_values('OLS', ascending=False)) ```

Cross-Validation for Hyperparameter Tuning

```python from sklearn.model_selection import cross_val_score from sklearn.linear_model import Ridge

# Test different alpha values

alphas = [0.001, 0.01, 0.1, 1, 10, 100] cv_scores = []

for alpha in alphas: model = Ridge(alpha=alpha) scores = cross_val_score(model, X_train, y_train, cv=5, scoring='r2') cv_scores.append(scores.mean())

# Plot results

plt.figure(figsize=(10, 6)) plt.plot(alphas, cv_scores, marker='o') plt.xscale('log') plt.xlabel('Alpha (regularization strength)') plt.ylabel('Cross-Validation $R^2$ Score') plt.title('Ridge Regression: Choosing Optimal Alpha') plt.grid(True) plt.show()

best_alpha = alphas[np.argmax(cv_scores)] print(f"Best alpha: {best_alpha}") ```

---

## 16. Advantages and Limitations

### Advantages ✓

**1. Simplicity and Interpretability** - Easy to understand and explain - Coefficients show direct impact of each feature - Great starting point for analysis

**2. Computational Efficiency** - Fast training, even on large datasets - Low memory requirements - Suitable for real-time applications

**3. Statistical Foundation** - Well-established mathematical theory - Confidence intervals for predictions - Hypothesis testing available

**4. Robustness** - Relatively stable with proper assumptions - Less prone to overfitting than complex models - Works well as baseline model

**5. No Hyperparameters** - (For basic OLS) No tuning required - Reproducible results - Easy to implement

**6. Extrapolation** - Can make predictions outside training range - (With caution) based on learned relationship

### Limitations ✗

**1. Linearity Assumption** - Cannot capture non-linear relationships directly - May underfit complex
```

data - Requires feature engineering for non-linear patterns

2. Sensitivity to Outliers - Especially with L_2 loss - Single outlier can skew entire model - May need robust regression methods

3. Multicollinearity Issues - Unstable coefficients when features correlated - Difficult to interpret individual effects - Requires regularization or feature selection

4. Feature Engineering Required - Features must be in suitable form - May need transformations - Manual effort for non-linear relationships

5. Homoscedasticity Requirement - Assumes constant error variance - Violated in many real-world scenarios - May need weighted regression

6. Limited Explanatory Power - Cannot capture complex interactions automatically - May miss important patterns - Advanced models needed for deeper insights

7. Assumption Violations - Real data often violates assumptions - Requires careful diagnostics - May need alternative methods

When to Use Linear Regression

Good Fit ✓ - Linear relationships present - Interpretability important - Quick baseline needed - Large datasets with simple patterns - Well-behaved, clean data

Poor Fit ✗ - Non-linear relationships - Complex interactions between features - Many outliers present - High-dimensional data ($d >> n$) - Strong multicollinearity

---

Summary and Key Takeaways

Core Concepts

1. **Linear regression models linear relationships** between inputs and outputs
2. **Loss functions** (L_1, L_2, L_∞) measure prediction errors differently
3. **Optimization** can be direct (closed-form) or iterative (gradient descent)
4. **Regularization** (Ridge, Lasso, Elastic Net) prevents overfitting
5. **Evaluation metrics** (MSE, R^2, etc.) assess model performance

Best Practices

- ✓ Always check assumptions before applying
- ✓ Visualize data and residuals
- ✓ Use train/test split for validation
- ✓ Consider regularization with many features
- ✓ Compare multiple models
- ✓ Interpret coefficients carefully

Next Steps

After mastering linear regression: 1. **Logistic Regression**: For classification problems 2. **Polynomial Regression**: For non-linear relationships 3. **Neural Networks**: For complex patterns 4. **Ensemble Methods**: For improved predictions

---

References and Further Reading

Academic Sources

1. Bishop, C. M. and Nasrabadi, N. M. (2006). *Pattern Recognition and Machine Learning*. Springer. (Chapter 3)
2. Hastie, T., Tibshirani, R., Friedman, J. H. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer. (Chapters 2-3)
3. James, G., Witten, D., Hastie, T., Tibshirani, R. (2013). *An Introduction to Statistical Learning*. Springer. (Chapter 3)

Online Resources

1. Scikit-learn Documentation: Linear Models https://scikit-learn.org/stable/modules/linear_model.html
2. Stanford CS229: Machine Learning - Linear Regression http://cs229.stanford.edu/

Mathematical Background

1. Convex Optimization by Boyd & Vandenberghe For understanding optimization methods
2. Linear Algebra by Gilbert Strang For matrix formulation and theory

---

Appendix: Implementation Details

A. Computing Gradients

For L_2 loss $J(w) = \frac{1}{2}n\|Xw - y\|_2^2$:

``` Step 1: Expand $J(w) = (1/2n)(Xw - y)^\top(Xw - y) = (1/2n)(w^\top X^\top Xw - 2y^\top Xw + y^\top y)$

Step 2: Differentiate (using $\partial/\partial w[w^\top Aw] = 2Aw$ for symmetric A) $\nabla J(w) = (1/n)(X^\top Xw - X^\top y) = (1/n)X^\top(Xw - y)$ ```

## B. Normal Equation Derivation

Setting $\nabla J(w) = 0$:

$X^\top(Xw - y) = 0 \; X^\top Xw = X^\top y \; w = (X^\top X)^{-1}X^\top y$

Pseudoinverse when $X^\top X$ is singular: $w = X^+ y$ where $X^+ = (X^\top X)^{-1}X^\top$

## C. Computational Complexity

| Operation | Complexity | Notes |
|---|---|---|
| $X^\top X$ | $O(nd^2)$ | Matrix multiplication |
| $(X^\top X)^{-1}$ | $O(d^3)$ | Matrix inversion |
| Normal Eq. | $O(nd^2 + d^3)$ | Dominated by inversion when d large |
| Gradient | O(nd) | For one iteration |
| Gradient Descent | O(knd) | k iterations |

Use gradient descent when d is large.

---

**End of Lecture Notes**

*This comprehensive guide combines theoretical foundations with practical implementation, suitable for*
```

both academic study and applied machine learning.