# Lecture 07: Bias-Variance Tradeoff and Regularization

CS109A: Introduction to Data Science

Harvard University

- **Course:** CS109A: Introduction to Data Science
- **Lecture:** Lecture 07
- **Instructor:** Pavlos Protopapas, Kevin Rader, Chris Gumb
- **Objective:** Understanding the bias-variance tradeoff, diagnosing overfitting through coefficient analysis, and learning regularization techniques (Ridge and Lasso) to combat overfitting

## Contents

# 1 Introduction and Motivation

**Lecture Overview**

This lecture addresses one of the most fundamental concepts in machine learning: the **bias-variance tradeoff**. We'll understand why models fail, how to diagnose the problem, and introduce powerful techniques called **regularization** to fix it.

**Key Topics:**

- **Sources of Error**: Irreducible vs. reducible error
- **Bias-Variance Tradeoff**: The fundamental tension in model complexity
- **Diagnosing Overfitting**: Looking at coefficient magnitudes
- **Ridge Regression (L2)**: Penalizing squared coefficients
- **Lasso Regression (L1)**: Penalizing absolute coefficients (with feature selection!)
- **Hyperparameter Tuning**: Finding optimal $\lambda$ using cross-validation

## 1.1 The Big Picture

Our ultimate goal is **generalization**—building models that perform well on *new, unseen data*, not just the training data we happened to collect.

We've learned that:

- Too simple models **underfit**: They can't capture the true patterns
- Too complex models **overfit**: They memorize noise instead of learning patterns

Today we formalize this intuition mathematically and introduce techniques to navigate between these extremes.

# 2 Sources of Prediction Error

When our model makes errors on test data, those errors can come from different sources.

## 2.1 Irreducible Error (Aleatoric Error)

**Definition: Irreducible Error**

**Irreducible error** is the inherent noise in the data that cannot be eliminated no matter how good our model is. This is also called **aleatoric error**.

This error exists because:

- Measurements have inherent randomness
- There are unmeasured variables affecting the outcome
- The relationship itself may have stochastic components

**Example: Irreducible Error Analogy**

Imagine recording audio with the world's best microphone. Even with perfect equipment, you'll still pick up ambient noise—air molecules vibrating, electronic interference, etc.
No matter how much you improve the microphone (model), you can't eliminate environmental noise (irreducible error).
**Acceptance**: We acknowledge this error exists and focus on what we *can* control.

### 2.2 Reducible Error

The error we *can* control comes from our choice of model. This **reducible error** decomposes into two components:

1. **Bias**: Error from wrong assumptions in the model
2. **Variance**: Error from sensitivity to training data fluctuations

## 3 The Bias-Variance Tradeoff

### 3.1 Understanding Bias

**Definition: Bias**

**Bias** measures how far off the model's average prediction is from the true value.

- **High Bias**: Model consistently misses the target (inaccurate)
- **Low Bias**: Model's predictions are centered around the truth

High bias typically occurs when the model is **too simple** to capture the underlying patterns.

**Example: High Bias: The Linear Model on Curved Data**

Imagine the true relationship is curved (like a parabola), but you fit a straight line.
No matter how many times you collect new data and refit:

- Every fitted line will miss the curve
- The average of all fitted lines still misses the truth
- This systematic error is **bias**

The model is fundamentally incapable of capturing the true pattern—it's **underfitting**.

### 3.2 Understanding Variance

**Definition: Variance**

**Variance** measures how much the model's predictions change when trained on different datasets.

- **High Variance**: Predictions swing wildly with different training data
- **Low Variance**: Predictions are stable regardless of training data

High variance typically occurs when the model is **too complex** and learns noise specific to each training set.

**Example: High Variance: The Spaghetti Plot**

Professor Protopapas demonstrates this beautifully with a simulation:

**Experiment**: Take 2,000 different random samples from the same population. For each sample, fit a model. Plot all 2,000 fitted curves.

**Linear Model (Low Variance)**: All 2,000 lines cluster tightly together—very stable predictions across different training sets.

**Degree-10 Polynomial (High Variance)**: The 2,000 curves look like "spaghetti noodles"—wildly different predictions for each training set. The model is chasing the noise in each particular sample.

### 3.3 The Tradeoff Visualized

**The Fundamental Tradeoff**

As model complexity increases:

- **Bias decreases**: More complex models can fit more patterns
- **Variance increases**: More complex models are more sensitive to noise

As model complexity decreases:

- **Bias increases**: Simpler models miss true patterns
- **Variance decreases**: Simpler models are more stable

**Total Error** = $\text{Bias}^2$ + Variance + Irreducible Error

This creates a U-shaped curve when plotted against complexity. Our goal is to find the **sweet spot** at the bottom of the U.

### 3.4 The Dartboard Analogy

Think of model predictions like throwing darts at a target:

| | **Low Variance** | **High Variance** |
|---|---|---|
| **High Bias** | Darts clustered together, but away from bullseye (Underfitting) | Darts scattered everywhere, missing the bullseye (Worst case) |
| **Low Bias** | Darts clustered tightly around the bullseye (**Ideal model!**) | Darts scattered around the bullseye (Overfitting) |

**Table 1:** *The four scenarios of bias and variance*

## 4 Diagnosing Overfitting: Coefficient Analysis

How can we tell if our model is overfitting? Look at the **coefficients**!

### 4.1 The Key Insight

**Overfitting Symptom: Exploding Coefficients**

When a model overfits:

- Coefficients ($\beta$ values) become **extremely large**
- Coefficient values are **unstable**—they vary wildly across different training sets

This happens because the model is trying to fit every tiny wiggle in the training data, requiring extreme parameter values.

**Example: Coefficient Distributions: Linear vs. Polynomial**

From the 2,000-model simulation:

**Linear Model**:

- $\beta_0$ values range from roughly 0.0 to 1.25
- $\beta_1$ values similarly bounded
- Narrow violin plots (low variance in coefficients)

**Degree-10 Polynomial**:

- Some coefficients ($\beta_5, \beta_8, \beta_9$) reach values on the order of $10^9$ (one billion!)
- Extremely wide violin plots (high variance in coefficients)
- The y-axis label shows "1e9"—coefficients are astronomically large

### 4.2 The Solution Preview

This observation leads directly to our solution:

**If overfitting = large coefficients,**

**then preventing overfitting = keeping coefficients small!**

This is the core idea behind **regularization**.

## 5 Regularization: The Core Idea

### 5.1 Penalizing Complexity

**Definition: Regularization**

**Regularization** modifies the loss function to penalize model complexity, encouraging simpler models that generalize better.

Instead of minimizing just MSE, we minimize:

$$\mathcal{L}_{\text{regularized}} = \underbrace{\text{MSE}}_{\text{Fit the data}} + \underbrace{\lambda \cdot \text{Penalty}}_{\text{Keep model simple}}$$

The model must now balance two competing objectives:

1. Fit the training data well (minimize MSE)
2. Keep coefficients small (minimize penalty)

**Example: Game Time: How to Discourage Large Coefficients?**

Professor Protopapas poses this question to the class:

**Options**:

A. Divide all model parameters by large numbers

B. Make sure the causal relation between predictors and response is true

C. Discard any model with parameter values larger than one

D. Penalize the model with a penalty proportional to parameter values

**Answer**: D!

Why not A? Dividing just scales the data—nothing fundamentally changes.

Why not C? Discarding models creates bias and arbitrary cutoffs.

**The right approach**: Add a term to the loss function that grows when coefficients grow.

### 5.2 The Regularization Parameter $\lambda$

The hyperparameter $\lambda$ (lambda) controls the **strength** of regularization:

**Understanding $\lambda$**

- $\lambda = 0$: No regularization. This is just ordinary linear regression. Risk of overfitting if model is complex.
- $\lambda$ **small**: Mild regularization. Coefficients are somewhat constrained.
- $\lambda$ **large**: Strong regularization. Coefficients are heavily constrained, pushed toward zero.
- $\lambda \to \infty$: Extreme regularization. All coefficients become exactly zero. The model predicts just the mean (severe underfitting).

We need to find the $\lambda$ that balances bias and variance optimally.

## 6 Ridge Regression (L2 Regularization)

### 6.1 The Ridge Loss Function

**Definition: Ridge Regression**

**Ridge Regression** uses the **L2 norm** (sum of squared coefficients) as the penalty:

$$\mathcal{L}_{\text{Ridge}} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

Equivalently:

$$\mathcal{L}_{\text{Ridge}} = \text{MSE} + \lambda \|\boldsymbol{\beta}\|_2^2$$

where $\|\boldsymbol{\beta}\|_2^2 = \beta_1^2 + \beta_2^2 + \ldots + \beta_p^2$

**Important: Don't Regularize the Intercept!**

Notice the penalty sums from $j = 1$ to $p$, **not** including $\beta_0$ (the intercept).

**Why?** The intercept $\beta_0$ is just an overall offset—it doesn't relate to any predictor's influence. Overfitting comes from being too sensitive to predictors, not from the baseline level.

Always exclude $\beta_0$ from regularization.

### 6.2 Properties of Ridge Regression

**Ridge Regression Characteristics**

**How it shrinks coefficients**:

- Shrinks all coefficients **toward zero**
- But coefficients never become **exactly** zero
- Larger coefficients are penalized more heavily (due to squaring)

**Advantages**:

- **Closed-form solution**: Very fast to compute

$$\hat{\boldsymbol{\beta}}_{\text{Ridge}} = (X^T X + \lambda I)^{-1} X^T y$$

- Excellent for **multicollinearity**: Stabilizes estimates when predictors are correlated
- Keeps all predictors in the model (useful when you believe all matter)

**When to use**:

- When you believe all predictors contribute somewhat
- When predictors are highly correlated
- When you need fast computation

## 7 Lasso Regression (L1 Regularization)

### 7.1 The Lasso Loss Function

**Definition: Lasso Regression**

**Lasso Regression** (Least Absolute Shrinkage and Selection Operator) uses the **L1 norm** (sum of absolute values) as the penalty:

$$\mathcal{L}_{\text{Lasso}} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

Equivalently:

$$\mathcal{L}_{\text{Lasso}} = \text{MSE} + \lambda \|\boldsymbol{\beta}\|_1$$

where $\|\boldsymbol{\beta}\|_1 = |\beta_1| + |\beta_2| + \ldots + |\beta_p|$

## 7.2 Properties of Lasso Regression

**Lasso Regression Characteristics**

**How it shrinks coefficients**:

- Can shrink coefficients to **exactly zero**
- Effectively **removes** predictors from the model
- Performs automatic **feature selection**

**Advantages**:

- **Sparse solutions**: Produces interpretable models with fewer predictors
- **Feature selection**: Identifies which predictors actually matter
- Useful when you suspect only a few predictors are truly important

**Disadvantages**:

- **No closed-form solution**: Must use numerical optimization (slower)
- With correlated predictors, tends to pick one arbitrarily and zero others

**When to use**:

- When you have many predictors and suspect few are truly relevant
- When you want an interpretable model
- When you need automatic feature selection

## 7.3 Why Lasso Produces Zeros (Intuition)

**Example: Why L1 Creates Sparsity**

The geometric intuition involves the shape of the constraint regions:

**Ridge (L2)**: The constraint $\sum \beta_j^2 \leq c$ forms a **circle** (or sphere in higher dimensions). The MSE contours typically intersect this circle somewhere on the smooth boundary—rarely exactly at an axis.

**Lasso (L1)**: The constraint $\sum |\beta_j| \leq c$ forms a **diamond** (or cross-polytope). The corners of the diamond lie on the axes. The MSE contours are much more likely to hit a corner, where some coefficient is exactly zero.

This is why Lasso naturally produces sparse solutions!

| Aspect | Ridge (L2) | Lasso (L1) |
|---|---|---|
| Penalty term | $\sum \beta_j^2$ | $\sum \lvert\beta_j\rvert$ |
| Coefficient shrinkage | Toward zero, never exactly zero | Can be exactly zero |
| Feature selection | No (keeps all predictors) | Yes (automatic) |
| Solution type | Closed-form (analytical) | Numerical solver |
| Computation speed | Fast | Slower |
| Multicollinearity | Handles well | May pick one predictor arbitrarily |
| Interpretability | All predictors remain | Sparse, easier to interpret |
| Best when | All predictors matter | Few predictors matter |

**Table 2:** *Ridge vs. Lasso comparison*

# 8 Ridge vs. Lasso: Comparison

**Which to Choose?**

**The honest answer**: Try both and use cross-validation to compare!

**General guidelines**:

- If you have hundreds of predictors but suspect only a handful matter → **Lasso**
- If you believe all predictors contribute at least a little → **Ridge**
- If predictors are highly correlated → **Ridge** (more stable)
- If you need interpretability and feature selection → **Lasso**

There's also **Elastic Net** which combines both penalties—a topic for another day!

# 9 Finding Optimal $\lambda$: The Complete Procedure

$\lambda$ is a **hyperparameter**—it's not learned from data but must be set by us. We find it through validation or cross-validation.

**Very Important: Hyperparameter Tuning Rules**

- **NEVER** tune on training data (model will choose $\lambda = 0$)
- **NEVER** tune on test data (that's cheating—information leakage)
- **ALWAYS** use validation set or cross-validation

## 9.1 Method 1: Single Validation Set

1. **Split data**: Training / Validation / Test
2. **Choose $\lambda$ candidates**: Select a range to search
   - Typically: $\lambda \in \{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 100\}$
   - Use logarithmic spacing (orders of magnitude)
3. **For each** $\lambda$: Train model on training set, find $\boldsymbol{\beta}_\lambda$
   - Ridge: Use closed-form formula
   - Lasso: Use numerical solver

4. **Evaluate on validation set**: Compute MSE **without** the penalty term!

**Critical Point**

When evaluating models, compute **pure MSE only**—don't include the $\lambda \cdot$ penalty term.
Why? The penalty was a training tool to prevent overfitting. For evaluation, we care about actual prediction accuracy (how close to true values), not about coefficient sizes.

5. **Select best** $\lambda^*$: Choose the $\lambda$ with lowest validation MSE
6. **Refit (recommended)**: Combine training + validation data, retrain with $\lambda^*$
   - Model selection is done—no need to keep validation separate
   - More training data = better final model
7. **Final evaluation**: Report MSE on test set (use only once!)

### 9.2 Method 2: K-Fold Cross-Validation

A more robust approach that reduces dependence on a single validation split:

1. **Split data**: Training Pool (for CV) / Test
2. **Choose** $\lambda$ **candidates**: Same as before
3. **Divide training pool into K folds**: Typically $K = 5$ or $K = 10$
4. **For each** $\lambda$**, run K iterations**:
   - Iteration 1: Fold 1 = validation, Folds 2-K = training
   - Iteration 2: Fold 2 = validation, Folds 1,3-K = training
   - ... and so on ...
   - Record validation MSE for each iteration
5. **Average MSEs**: For each $\lambda$, compute mean of K validation MSEs
6. **Select best** $\lambda^*$: Choose $\lambda$ with lowest **average** validation MSE
7. **Refit on all training data**: Train final model using entire training pool with $\lambda^*$
   - Don't use one of the K models—retrain on all data
   - This ensures maximum information for final model
8. **Final evaluation**: Report MSE on test set

**Example: Why Refit After Cross-Validation?**

After K-fold CV, you have K different models (one per fold). Which do you use?
**Answer**: None of them! Each was trained on only $(K-1)/K$ of the data.
Instead, now that you've found $\lambda^*$, retrain a **new model** on the **entire** training pool. This final model benefits from all available training data.

### 9.3 Practical Implementation

```
from sklearn.linear_model import Ridge, Lasso, RidgeCV, LassoCV
from sklearn.model_selection import cross_val_score
import numpy as np
```

```python

# Define lambda values to search (sklearn calls it 'alpha')
alphas = np.logspace(-5, 2, 50)  # 50 values from 10^-5 to 10^2

# Method 1: Manual cross-validation
ridge = Ridge(alpha=1.0)
cv_scores = cross_val_score(ridge, X_train, y_train,
                            cv=5,
                            scoring='neg_mean_squared_error')
mean_mse = -cv_scores.mean()

# Method 2: Built-in CV (recommended)
# RidgeCV automatically finds best alpha
ridge_cv = RidgeCV(alphas=alphas, cv=5)
ridge_cv.fit(X_train, y_train)
print(f"Best alpha: {ridge_cv.alpha_}")

# Similarly for Lasso
lasso_cv = LassoCV(alphas=alphas, cv=5)
lasso_cv.fit(X_train, y_train)
print(f"Best alpha: {lasso_cv.alpha_}")

# Final evaluation on test set
test_mse = mean_squared_error(y_test, ridge_cv.predict(X_test))
```

Listing 1: Ridge and Lasso with Cross-Validation in sklearn

**Edge Cases in $\lambda$ Selection**

What if the optimal $\lambda$ is at the edge of your search range?
For example, if your lowest validation MSE occurs at $\lambda = 100$ (your maximum):
**Problem**: The true optimum might be at $\lambda = 1000$ or higher!
**Solution**: Expand your search range and try again. The optimal $\lambda$ should be in the "middle" of the U-shaped curve, not at the boundary.
If optimal is at $\lambda = 10^{-5}$ (your minimum), try smaller values.

## 10 Common Questions and Answers

**Q: Why don't we include the regularization term when evaluating on validation?**

A: The penalty term was a *training tool* to prevent overfitting—like training wheels on a bike. When we evaluate the model's true performance, we care about prediction accuracy (how close predictions are to actual values), not how "simple" the model is. MSE measures prediction quality; the penalty doesn't.

**Q: Should I use the same $\lambda$ search range for all problems?**

A: No! The appropriate range depends on your data scale and problem. Start with a wide range (e.g., $10^{-5}$ to $10^{5}$), visualize the results, and narrow down. If the optimum is at an edge, expand the range.

**Q: Can I tune K in K-fold CV?**

A: Technically yes, but it's usually not worth it. K=5 or K=10 work well in practice. The difference is rarely significant, and tuning K with another CV creates complexity. Just pick 5 or 10 and move on.

**Q: What about Elastic Net?**

A: Elastic Net combines Ridge and Lasso penalties:

$$\mathcal{L}_{\text{ElasticNet}} = \text{MSE} + \lambda_1 \sum |\beta_j| + \lambda_2 \sum \beta_j^2$$

It gets the best of both worlds: feature selection from Lasso and stability from Ridge. But now you have *two* hyperparameters to tune!

**Q: How much of data science is "art" vs. "science"?**

A: Professor Protopapas says roughly 70% science, 30% art. Not everything has proofs. With hyperparameters, you develop intuition over time. Start with reasonable defaults, use cross-validation, and accept that "good enough" is often the goal. You can't search infinite parameter spaces.

# 11 Quick Reference Summary

**Lecture 07 Quick Reference Card**

**1. Error Decomposition**

$$\text{Total Error} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

- High Bias = Underfitting (too simple)
- High Variance = Overfitting (too complex)

**2. Overfitting Diagnosis**

Overfitting $\Rightarrow$ Large, unstable coefficients
Solution: Penalize large coefficients!

**3. Regularization Formula**

$$\mathcal{L} = \text{MSE} + \lambda \cdot \text{Penalty}$$

- Ridge (L2): Penalty $= \sum \beta_j^2$ (shrinks toward zero)
- Lasso (L1): Penalty $= \sum |\beta_j|$ (can make zeros)

Don't regularize $\beta_0$!

**4. Finding $\lambda^*$**

1. Choose range: $\{10^{-5}, \ldots, 10^2\}$
2. For each $\lambda$: train, compute validation MSE (no penalty!)
3. Choose $\lambda^*$ with minimum validation MSE
4. Refit on train+validation with $\lambda^*$
5. Report on test (once!)

**5. Ridge vs. Lasso**

- Ridge: Fast, keeps all predictors, good for multicollinearity
- Lasso: Feature selection, sparse models, slower
- When unsure: Try both, compare with CV

# 12 Looking Ahead

With regularization in our toolkit, we now have powerful techniques to:

- Prevent overfitting without reducing model complexity
- Automatically select important features (Lasso)

- Handle multicollinearity (Ridge)

In upcoming lectures, we'll move beyond regression to **classification** problems, where we predict categories rather than continuous values. Many of the same principles—bias-variance tradeoff, regularization, cross-validation—will apply!