Lecture 06: Model Selection and Cross-Validation

CS109A: Introduction to Data Science

Harvard University

■ **Course:**	CS109A: Introduction to Data Science
■ **Lecture:**	Lecture 06
■ **Instructor:**	Pavlos Protopapas, Kevin Rader, Chris Gumb
■ **Objective:**	Understanding interaction terms, polynomial regression, model selection techniques, and cross-validation for choosing optimal models

Contents

1 Introduction and Motivation

Lecture Overview

This lecture marks a crucial turning point in our machine learning journey. We move beyond simple linear models to tackle more complex relationships in data, while also confronting the central challenge of machine learning: **overfitting**.

Key Topics:

- **Interaction Terms**: Modeling synergy effects between predictors
- **Polynomial Regression**: Capturing nonlinear relationships
- **Overfitting**: Understanding why more complex models aren't always better
- **Model Selection**: Techniques for choosing the best model
- **Cross-Validation**: A robust method for estimating model performance

1.1 The Progression of Model Building

Professor Protopapas emphasizes a fundamental philosophy for approaching any data science project:

The Incremental Approach to Modeling

1. **Start simple**: Begin with the simplest possible model (e.g., linear regression)
2. **Fully understand it**: Interpret coefficients, check assumptions
3. **Ask "Can we do better?"**: Identify limitations of the current model
4. **Add complexity gradually**: Only add features when justified by data

"I dislike with passion" jumping straight to complex models like visual transformers or diffusion models just because you found a tutorial that works. Start with EDA and simple models first!

This approach prevents us from building overly complex models that we don't understand and that may actually perform worse on new data.

1.2 The Mantra of This Lecture

Throughout this lecture, one phrase keeps recurring:

"Too many predictors and collinearity lead to overfitting."

And by the end, we'll extend this to:

"Too many predictors, collinearity, too many interaction terms, and too high polynomial degree lead to overfitting."

2 Understanding Overfitting

2.1 What Is Overfitting?

Before diving into complex models, we need to understand the primary enemy we'll be fighting: **overfitting**.

Definition: Overfitting

Overfitting occurs when a model learns not only the true underlying patterns in the training data but also the **noise** and **outliers**. The model essentially "memorizes" the training data rather than learning generalizable patterns.
Consequence: The model performs excellently on training data but poorly on new, unseen data.

Example: Student Analogy for Overfitting

Imagine overfitting were a student. Which behavior best describes it?

A. Studying just the night before the test

B. **Memorizing every lecture note word for word ← This is overfitting!**

C. Only studying one chapter for all subjects

D. Taking extensive notes but forgetting to understand concepts

The overfitting "student" memorizes everything perfectly but cannot answer questions that are even slightly different from what they memorized. They haven't learned the **concepts**—they've just memorized the **data**.

Example: TA Analogy for Overfitting

If a teaching assistant (TA) were an overfitting model, their grading behavior would be:
"Subtract points for every answer that does not include the word 'overfitting'."
This TA has learned a very specific pattern from limited examples and applies it rigidly, failing to generalize to what good answers actually look like.

2.2 When Does Overfitting Occur?

Overfitting happens when we give our model "too much power." And as the saying goes:

"With great power comes great responsibility."

Factors That Lead to Overfitting

1. **Too many predictors**: High-dimensional feature space
2. **Too high polynomial degree**: Excessively flexible curves
3. **Too many interaction terms**: Modeling spurious relationships
4. **Collinearity**: Redundant information confuses the model

The more complex the model, the more tendency it has to overfit. But if we remove all model power, we get **underfitting**—the model is too simple to capture the true patterns.

2.3 Generalization Error

The ultimate goal of any machine learning model is to perform well on **unseen data**—data the model has never seen during training.

Definition: Generalization Error

Generalization error measures how well a model performs on new, unseen data. It is the error we ultimately care about, not the training error.
Goal of model selection: Find the model that minimizes generalization error (equivalently, the model that minimizes overfitting).

3 Assumptions of Linear Regression Revisited

Before extending linear regression to handle nonlinearity, we need to understand its underlying assumptions. These assumptions come from the two key decisions we made when setting up linear regression:

1. We assumed a **linear relationship** between predictors and response
2. We chose **MSE (Mean Squared Error)** as our loss function

3.1 The Four Key Assumptions

Linear Regression Assumptions from MSE Loss

1. **Linearity**: The relationship between X and Y is linear
   - This is explicit in our model: $Y = \beta_0 + \beta_1 X_1 + \dots$
2. **Independence**: Errors are independent of each other
   - MSE simply *sums* squared errors
   - If errors were correlated, simple summation would be inappropriate
3. **Homoscedasticity**: Constant variance of errors across all values of X
   - MSE treats all errors *equally* (no weighting)
   - If variance varied, we should weight errors differently
4. **Normality**: Errors are normally distributed
   - Squaring errors connects to normal distribution assumptions
   - This will be explained in more detail in Lecture 9

3.2 Additional Considerations

Two more assumptions are often mentioned (though they don't directly violate linear regression assumptions):

- **Fixed X (No measurement error)**: We assume predictors are measured without error. If there's error in X, Bayesian approaches are needed.
- **No multicollinearity**: Low correlation between predictors. Multicollinearity won't break linear regression but will cause problems with interpretation.

3.3 Diagnosing Assumption Violations: Residual Analysis

How do we know if our assumptions hold? The primary diagnostic tool is **residual analysis**.

Definition: Residuals

The **residual** for observation i is the difference between the actual value and the predicted value:

$$e_i = y_i - \hat{y}_i$$

Residuals represent what our model *couldn't explain*.

3.3.1 Residual Plot Interpretation

We plot residuals (e) against either the predictor (X) or fitted values ($\hat{Y}$):

Reading Residual Plots

Good Model (Assumptions Satisfied):

- Residuals scattered randomly around zero
- No discernible pattern (looks like "white noise")
- Histogram of residuals resembles a bell curve (normal distribution)

Bad Model (Linearity Violated):

- Residuals show a U-shape, S-shape, or other systematic pattern
- This indicates the model is missing a nonlinear trend in the data

Bad Model (Homoscedasticity Violated):

- Residuals show a "fanning" or "funnel" pattern
- Spread of residuals increases (or decreases) with X
- This is called **heteroscedasticity**

Professor Protopapas offers a memorable rule:

*"If you see **any** pattern in your residual plot—even if you imagine dragons flying over castles—that means your assumptions are violated."*

3.3.2 The Histogram Trap

Is a Bell-Shaped Histogram Enough?

If residuals are symmetric (histogram looks normal) but have different spreads at different values of X, the normality assumption may appear satisfied while homoscedasticity is violated.

For example: residuals near $X = 0$ might be small, but residuals for large X might be huge. The marginal histogram (combining all residuals) might still look bell-shaped!

Lesson: Always examine residual plots alongside histograms.

4 Extending Linear Regression: Interaction Terms

Now we begin extending our linear models to handle more complex relationships.

4.1 The Synergy Effect

In reality, the effect of one predictor often depends on the value of another predictor. This is called a **synergy effect** or **interaction effect**.

Example: TV and Radio Advertising

Consider predicting sales based on advertising budgets:
Without considering interaction: If I increase the TV budget by \$1,000, sales increase by β_1 (some fixed amount), regardless of what the radio budget is.
With interaction: If I increase TV budget by \$1,000 *while also having high radio spending*, the effect on sales might be *greater* than if radio spending were low. The two advertising channels might have a synergistic effect.

4.2 Modeling Interactions

To capture interaction effects, we multiply predictors together:

Definition: Interaction Term

For predictors X_1 and X_2, the **interaction term** is $X_1 \times X_2$.
The model becomes:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 (X_1 \times X_2)$$

Note: The term $X_1 \times X_2$ is **nonlinear** in the predictors (though still linear in the coefficients β).

4.3 Detailed Example: Credit Card Balance

Let's examine the credit card balance dataset, predicting balance from income and student status.

4.3.1 Model Without Interaction

$$\text{Balance} = \beta_0 + \beta_1 \times \text{Income} + \beta_2 \times \text{Student}$$

Where Student is a dummy variable: 0 for non-student, 1 for student.

For non-students (Student = 0):

$$\text{Balance} = \beta_0 + \beta_1 \times \text{Income}$$

For students (Student = 1):

$$\text{Balance} = (\beta_0 + \beta_2) + \beta_1 \times \text{Income}$$

Interpretation Without Interaction

Both students and non-students have the **same slope** (β_1).
This means: when income increases by \$1,000, balance increases by exactly β_1 dollars—regardless of whether you're a student or not.
Graphically: Two **parallel lines** with different intercepts.

4.3.2 Model With Interaction

$$\text{Balance} = \beta_0 + \beta_1 \times \text{Income} + \beta_2 \times \text{Student} + \beta_3 \times (\text{Income} \times \text{Student})$$

For non-students (Student = 0):

$$\text{Balance} = \beta_0 + \beta_1 \times \text{Income}$$

For students (Student = 1):

$$\text{Balance} = (\beta_0 + \beta_2) + (\beta_1 + \beta_3) \times \text{Income}$$

Interpretation With Interaction

Now students and non-students have **different slopes**!

- Non-students: slope $= \beta_1$
- Students: slope $= \beta_1 + \beta_3$

If $\beta_3 > 0$: Students increase their balance *more* for each additional \$1,000 of income (perhaps they spend more freely).
If $\beta_3 < 0$: Students increase their balance *less* per income increase (perhaps they're more cautious savers).
Graphically: Two lines with different slopes that may cross.

Where Do These Coefficients Come From?

We don't choose $\beta_0, \beta_1, \beta_2, \beta_3$—**the data tells us**!
We fit the model to our training data, and the optimization process finds the coefficient values that minimize MSE.
If $\beta_3 \approx 0$, the data is telling us there's no significant interaction effect.

5 Polynomial Regression

What if the relationship between X and Y is fundamentally curved, not just complicated by interactions?

5.1 When Linear Isn't Enough

Example: Curved Relationships

Imagine plotting your data and seeing this:

- Blue dots follow a clear curved pattern
- The best linear fit (straight line) misses the curve
- Residuals show a systematic U-shaped pattern

What we need is a model that can fit a curved line—a polynomial!

5.2 The Polynomial Model

Definition: Polynomial Regression

A polynomial regression model of degree M is:

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \ldots + \beta_M X^M$$

This allows the model to fit curves of varying complexity depending on M.

5.3 The Key Insight: It's Still Linear Regression!

Here's the crucial trick that makes polynomial regression easy:

Polynomial Regression is a Special Case of Multiple Linear Regression

The Trick: Define new "fake" predictors:

$$\begin{aligned}\tilde{X}_1 &= X\\ \tilde{X}_2 &= X^2\\ \tilde{X}_3 &= X^3\\ &\vdots\\ \tilde{X}_M &= X^M\end{aligned}$$

Now our polynomial model looks like:

$$Y = \beta_0 + \beta_1 \tilde{X}_1 + \beta_2 \tilde{X}_2 + \ldots + \beta_M \tilde{X}_M$$

This is exactly a **multiple linear regression** with M predictors!

Why this matters: We can use the exact same formulas and algorithms we developed for linear regression. The normal equation still works:

$$\hat{\beta} = (\tilde{X}^T \tilde{X})^{-1} \tilde{X}^T y$$

5.4 Implementation in Python

```python
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

# Step 1: Create the polynomial features (design matrix)
poly = PolynomialFeatures(degree=3)
X_poly = poly.fit_transform(X)  # Creates columns: 1, X, X^2, X^3

# Step 2: Fit regular linear regression
model = LinearRegression()
model.fit(X_poly, y)

# That's it! Same optimization, same formulas, curvy results.
```

Listing 1: Polynomial Regression in sklearn

5.5 Important Warnings

PolynomialFeatures Creates More Than You Expect

Warning 1: Intercept Column

`PolynomialFeatures` by default creates a column of 1s (the intercept term). If you also use `LinearRegression` with its default `fit_intercept=True`, you'll have **two intercepts**!

Solution: Either:

- Use `PolynomialFeatures(include_bias=False)` and `fit_intercept=True`, OR
- Use `PolynomialFeatures(include_bias=True)` and `fit_intercept=False`

TL;DR: If using polynomial features, set `fit_intercept=False`.

Interaction Terms Are Included!

Warning 2: Multiple Predictors

If you have multiple predictors (e.g., X_1 and X_2) and use `PolynomialFeatures(degree=2)`, you get:

- $1, X_1, X_2$ (degree 0 and 1 terms)
- X_1^2, X_1X_2, X_2^2 (degree 2 terms, including interaction!)

For degree 3, you'd also get $X_1^3, X_1^2X_2, X_1X_2^2, X_2^3$, etc.

This can lead to explosion of features and potential overfitting!

Note: sklearn doesn't have a simple flag to exclude interaction terms. You'd need to manually remove columns.

Feature Scaling is Critical

Warning 3: Numerical Stability

Consider $X = 100$:

- $X^2 = 10,000$
- $X^3 = 1,000,000$
- $X^{10} = 10^{20}$

These vastly different scales cause numerical instability when computing $(\tilde{X}^T\tilde{X})^{-1}$.

Solution: Always standardize your features before polynomial expansion!

```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Then apply PolynomialFeatures to X_scaled
```

5.6 Choosing the Polynomial Degree

The degree M is a **hyperparameter**—a choice we must make before training.

- M too low (e.g., $M = 1$): **Underfitting**—model can't capture the curve
- M too high (e.g., $M = 50$): **Overfitting**—model fits every little wiggle and noise
- M just right: Captures the true underlying trend without fitting noise

How do we find the right M? That's the topic of model selection!

6 Model Selection: Finding the Right Balance

Model selection is the process of choosing among different model candidates to find the one that will perform best on new data.

6.1 The Train-Validation-Test Split

The Three Data Splits

1. **Training Set**: Used to *train* the model (find the β coefficients)
2. **Validation Set**: Used to *select* the best model/hyperparameters
3. **Test Set**: Used *once at the very end* to report final model performance

Very Important: The Sacred Test Set

"There's a special place in hell for people who use test data to choose the model."
—Professor Protopapas

The test set must be kept completely separate until final evaluation. Do NOT:

- Use it to tune hyperparameters
- Use it to compare different models
- Look at it during model development

Best practice: Have someone else hold the test set and only evaluate your final model once.

6.2 Model Selection Methods

6.2.1 Exhaustive Search

With J predictors, we could have 2^J possible models (each predictor is either in or out).

Example: Exhaustive Search Complexity

- $J = 3$ predictors $\rightarrow 2^3 = 8$ models (manageable)
- $J = 10$ predictors $\rightarrow 2^{10} = 1,024$ models
- $J = 20$ predictors $\rightarrow 2^{20} = 1,048,576$ models!

Exhaustive search is only practical for very small feature spaces.

6.2.2 Greedy Algorithms

Greedy algorithms make locally optimal choices at each step, without looking at the big picture.

Definition: Forward Selection

Forward Selection is a greedy algorithm for feature selection:

1. Start with model M_0 containing no predictors (just intercept)
2. For each of the J predictors, try adding it to the model
3. Keep the one that reduces validation error the most $\rightarrow$ Model M_1
4. Repeat: try adding each remaining predictor to M_1
5. Keep going until you've tried models with all predictors
6. Select the model with lowest validation error among $M_0, M_1, \ldots, M_J$

Complexity: $O(J^2)$ instead of $O(2^J)$—much faster!

Limitation: May miss optimal combinations (e.g., $X_1 + X_2$ might be best, but if X_3 alone beats X_1 alone, we'd never try $X_1 + X_2$).

Other greedy approaches include:

- **Backward Elimination**: Start with all predictors, remove the least helpful one at each step
- **Stepwise Selection**: Combination of forward and backward

6.3 Hyperparameter Tuning with Validation

For polynomial regression, we tune the degree M using the validation set:

1. For each candidate degree $M \in \{1, 2, 3, \ldots, 10\}$:
   (a) Train the model on the training set
   (b) Compute MSE on the validation set
2. Plot validation MSE vs. degree M
3. Select the M with lowest validation MSE

6.3.1 The U-Shaped Curve

Training Error vs. Validation Error

Training Error (MSE on training data):

- Decreases monotonically as model complexity increases
- More complex model can always fit training data better (or at least as well)
- Can reach zero with enough complexity (just memorize the data!)

Validation Error (MSE on validation data):

- Initially decreases as complexity captures true patterns
- Reaches a minimum at the "sweet spot"
- Then **increases** as model starts fitting noise (overfitting)

The validation error curve is typically **U-shaped**. We choose the complexity at the bottom of the U.

7 Cross-Validation: A Robust Approach

7.1 The Problem with a Single Validation Set

Using a single validation set has a critical flaw:

Example: Overfitting to the Validation Set

Suppose the true underlying relationship is cubic (degree 3).
But by chance, our randomly chosen validation points happen to lie almost perfectly on a straight line!
When we compare models:

- Degree 1 model: Low validation error (lucky fit)
- Degree 3 model: Higher validation error (validation points don't follow cubic well)

We'd incorrectly choose degree 1!
The problem: We avoided overfitting to training data, but ended up **overfitting to the validation data**.

7.2 The Solution: K-Fold Cross-Validation

The remedy is to validate on **multiple different splits** and average the results.

Definition: K-Fold Cross-Validation

K-Fold Cross-Validation procedure:

1. Set aside the test set (never touch it during CV)
2. Divide the remaining data into K equal "folds" (typically $K = 5$ or $K = 10$)
3. For $i = 1$ to K:
   (a) Use fold i as the validation set

   (b) Use all other $K-1$ folds as the training set

   (c) Train the model and compute validation error MSE_i

4. Compute the average: $CV = \frac{1}{K}\sum_{i=1}^{K} MSE_i$

This CV score is a more robust estimate of how the model will perform on new data.

7.2.1 Visual Representation of 5-Fold CV

Iteration	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
1	**Val**	Train	Train	Train	Train
2	Train	**Val**	Train	Train	Train
3	Train	Train	**Val**	Train	Train
4	Train	Train	Train	**Val**	Train
5	Train	Train	Train	Train	**Val**

Each data point gets to be in the validation set exactly once!

7.3 Using Cross-Validation for Model Selection

To select the best hyperparameter (e.g., polynomial degree M):

1. For each candidate $M \in \{1, 2, 3, \dots, 10\}$:
   (a) Perform K-fold CV
   (b) Record the CV score (average validation MSE)
2. Select the M with the lowest CV score
3. (Optional) Retrain on all training data with chosen M
4. Evaluate final performance on test set

7.4 Leave-One-Out Cross-Validation (LOOCV)

An extreme case of K-fold CV where $K = N$ (number of data points):

- Each iteration uses just 1 point for validation, $N-1$ for training
- Repeated N times
- **Pros**: Very low bias in estimate
- **Cons**: Computationally expensive (train N models!)

In practice, $K = 5$ or $K = 10$ provides a good balance between bias and computational cost.

7.5 Implementation: The Negative MSE Trick

sklearn Uses Negative MSE

sklearn's cross-validation functions are designed to **maximize** a scoring metric (like accuracy). But we want to **minimize** MSE!

Solution: Use `scoring='neg_mean_squared_error'`

The function returns *negative* MSE values. To get actual MSE:

```python
cv_results = cross_validate(model, X, y, cv=5,
                            scoring='neg_mean_squared_error')
actual_mse = -cv_results['test_score'].mean()
```

```python
from sklearn.model_selection import cross_validate
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline
import numpy as np

# Find best polynomial degree using CV
degrees = range(1, 11)
cv_scores = []

for degree in degrees:
    # Create pipeline: Scale -> Polynomial -> Linear Regression
    model = make_pipeline(
        StandardScaler(),
        PolynomialFeatures(degree=degree, include_bias=False),
        LinearRegression()
    )

    # 5-fold cross-validation
    cv_results = cross_validate(
        model, X, y, cv=5,
        scoring='neg_mean_squared_error',
        return_train_score=True
    )

    # Convert to positive MSE and store
    cv_scores.append(-cv_results['test_score'].mean())

# Find best degree
best_degree = degrees[np.argmin(cv_scores)]
print(f"Best polynomial degree: {best_degree}")
```

Listing 2: Complete Cross-Validation Example

8 Interpreting Models: Beyond Numbers

Even when a model has good MSE or R^2, we must interpret it to ensure it makes sense.

Example: When Numbers Lie

Case 1: Model for TV budget (X) vs. Sales (Y):

$$Y = -0.05X + 6.2$$

Problem: Negative slope means more TV spending leads to *less* sales. Does that make sense? Probably not! Either:

- The data is wrong
- The model is misspecified
- There's some confounding factor we're missing

Case 2: Another model:

$$Y = 0.02X - 0.5$$

Problem: Negative intercept means when $X = 0$ (no TV spending), sales are *negative*. That's impossible!

Lesson: Always sanity-check your model coefficients against domain knowledge.

The Complete Model Evaluation Checklist

1. Check quantitative metrics (MSE, R^2)
2. Examine residual plots for assumption violations
3. Interpret coefficients—do they make intuitive sense?
4. Consider the domain context—is the model telling a plausible story?
5. Validate on held-out data to ensure generalization

9 Dealing with Categorical Variables (Reminder)

9.1 Two Categories: Dummy Variables

For a categorical variable with 2 categories (e.g., Student: Yes/No):

- Create one dummy variable: 0 = No, 1 = Yes

9.2 More Than Two Categories: One-Hot Encoding

Example: Encoding Ethnicity

Variable: Ethnicity with categories Asian, Caucasian, African-American

Wrong approach: Code as 0, 1, 2

- This implies ordering ($0 < 1 < 2$)

- Implies equal "distances" between categories
- Neither makes sense for categorical data!

Correct approach: One-hot encoding

- Create 3 dummy variables: `Is_Asian`, `Is_Caucasian`, `Is_AfricanAmerican`
- Each observation has exactly one 1 and two 0s

But wait: We can drop one column! If we know someone is not Asian and not Caucasian, they must be African-American.
So we use only 2 dummy variables, and the third category becomes the "baseline."

Interpreting Coefficients with Dummy Variables

Model: Balance $= \beta_0 + \beta_1 \times \text{Is\_Asian} + \beta_2 \times \text{Is\_Caucasian}$
Interpretations:

- β_0: Average balance for African-Americans (the baseline)
- $\beta_0 + \beta_1$: Average balance for Asians
- $\beta_0 + \beta_2$: Average balance for Caucasians
- β_1: Difference in balance between Asians and African-Americans
- β_2: Difference in balance between Caucasians and African-Americans

10 Quick Reference Summary

Lecture 06 Quick Reference Card

1. Overfitting

- Model memorizes training data including noise
- Low training error, high test/validation error
- Caused by: too many predictors, high polynomial degree, too many interactions
- Solution: Model selection with validation/cross-validation

2. Interaction Terms

- Model: $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 (X_1 \times X_2)$
- Captures synergy effects between predictors
- Changes the slope of one variable based on another

3. Polynomial Regression

- Model: $Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \ldots + \beta_M X^M$
- Special case of multiple linear regression (linear in β!)
- Warnings: Scale features, watch for intercept duplication

4. Data Splits

- **Training**: Fit model parameters (β)
- **Validation**: Choose hyperparameters/model
- **Test**: Final performance report (use ONCE!)

5. K-Fold Cross-Validation

- Split data into K folds, rotate validation fold
- Average K validation scores for robust estimate
- Avoids overfitting to a single validation split
- Typical K: 5 or 10

11 Common Questions and Answers

Q: How can polynomial regression be "linear" when it fits curves?

A: It's linear *in the coefficients*. The model $Y = \beta_0 + \beta_1 X + \beta_2 X^2$ is nonlinear in X (the graph is curved), but it's a linear combination of the β terms. We can treat X^2 as a new variable $\tilde{X}$, and then it's just $Y = \beta_0 + \beta_1 X + \beta_2 \tilde{X}$—standard multiple linear regression!

Q: What's the difference between validation and test sets?

A: Purpose!

- **Validation**: Used repeatedly during development to compare models and tune hyperparameters
- **Test**: Used exactly once at the end to report final performance

Think of validation as "practice exams" and test as the "final exam."

Q: How do I choose K for K-fold CV?

A: There's no perfect answer, but $K = 5$ or $K = 10$ are standard choices. They balance:

- Bias (larger K = less bias, since training sets are larger)
- Variance (larger K = more variance, since validation sets are smaller)
- Computational cost (larger K = more model fits)

Q: Should I always standardize my features?

A: Not always required, but usually a good idea:

- Required for: Polynomial regression, regularization, KNN, SVM, neural networks
- Optional for: Simple/multiple linear regression (doesn't affect predictions, only coefficient interpretation)

When in doubt, standardize!

Q: My model has high R^2 but coefficients don't make sense. What's wrong?

A: Several possibilities:

- Multicollinearity between predictors
- Overfitting to noise
- Data errors or preprocessing issues
- Confounding variables not in the model

Always combine quantitative metrics with qualitative interpretation!

12 Looking Ahead

In the next lecture (Lecture 07), we will explore:

- **Regularization**: A powerful technique to prevent overfitting by penalizing large coefficients
- **Ridge Regression**: L2 regularization
- **Lasso Regression**: L1 regularization (also performs feature selection!)
- **Bias-Variance Tradeoff**: The fundamental tension in machine learning

These techniques give us another tool (beyond cross-validation) to combat overfitting.