Cross Validation & Hyperparameter Tuning

CMSC 173 - Machine Learning

Course Lecture

Outline

How to Validate Models?

The Model Validation Problem

Central Question: How do we assess model performance?

Training Error

- Error on data used to train the model
- Always optimistically biased
- Cannot be used for model selection
- Misleading indicator

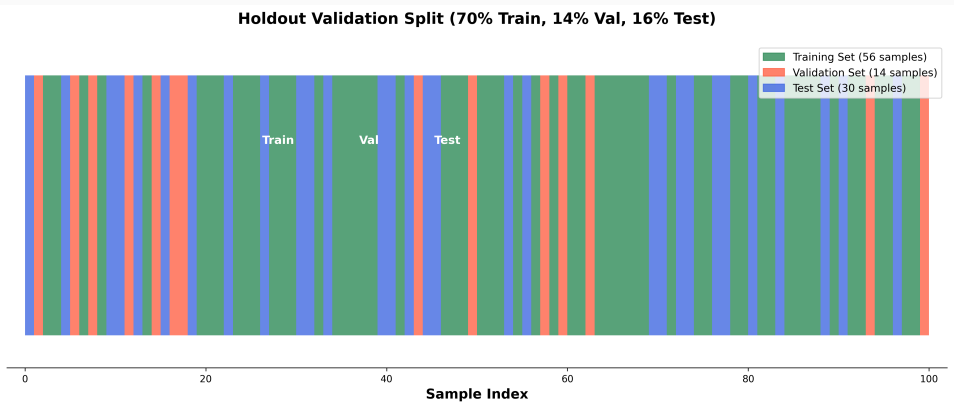
Test Error

- Error on unseen data
- True indicator of generalization
- Should only be used once
- Gold standard

Problem

We need to estimate generalization performance **without** using the test set during model development and hyperparameter tuning.

The Three-Way Data Split



Key Insight

Validation set acts as a proxy for the test set during model development, allowing us to:

- Compare different models
- Tune hyperparameters
- Avoid overfitting to the test set

Holdout Validation

Holdout Validation Method

Procedure:

1. Randomly split data into training and validation sets
2. Train model on training set
3. Evaluate on validation set
4. Select best performing model/hyperparameters
5. Final evaluation on held-out test set

Common Split Ratios:

- 70% Train, 15% Validation, 15% Test
- 60% Train, 20% Validation, 20% Test
- 80% Train, 10% Validation, 10% Test

Mathematical Formulation

Given dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$:

$$\mathcal{D}_{train} \cup \mathcal{D}_{val} \cup \mathcal{D}_{test} = \mathcal{D} \quad (1)$$

$$\mathcal{D}_{train} \cap \mathcal{D}_{val} \cap \mathcal{D}_{test} = \emptyset \quad (2)$$

$$|\mathcal{D}_{train}| = \alpha n \quad (3)$$

$$|\mathcal{D}_{val}| = \beta n \quad (4)$$

$$|\mathcal{D}_{test}| = (1 - \alpha - \beta)n \quad (5)$$

where typically $\alpha \in [0.6, 0.8]$

Advantage

Simple, fast, and provides unbiased estimate when validation set is large enough.

Holdout Validation: Advantages & Disadvantages

Advantages

- **Computational efficiency**: Train model only once
- **Simple implementation**: Straightforward to understand and code
- **Fast evaluation**: Quick assessment of model performance
- **Realistic simulation**: Mimics real-world deployment scenario

When to Use

- Large datasets (n $>$ 10,000)
- Computationally expensive models
- Time-sensitive applications

Disadvantages

- **High variance**: Performance estimate depends on specific split
- **Data inefficiency**: Reduces training data size
- **Unreliable for small datasets**: Estimates can be very noisy
- **Potential bias**: Unlucky splits can mislead model selection

When NOT to Use

- Small datasets (n $<$ 1,000)
- Need for robust performance estimates
- Critical applications requiring high confidence

Key Takeaway

Holdout validation trades robustness for computational efficiency.

K-Fold Cross-Validation

K-Fold Cross-Validation Method

Core Idea

Use **all data** for both training and validation by systematically rotating which portion serves as the validation

K-Fold Cross-Validation: Mathematical Formulation

Algorithm:

1. Partition dataset $\mathcal{D}$ into k equal-sized folds: $\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2 \cup \cdots \cup \mathcal{D}_k$
2. For each fold $i = 1, \ldots, k$:
   - Training set: $\mathcal{D}_{train}^{(i)} = \mathcal{D} \setminus \mathcal{D}_i$
   - Validation set: $\mathcal{D}_{val}^{(i)} = \mathcal{D}_i$
   - Train model $f^{(i)}$ on $\mathcal{D}_{train}^{(i)}$
   - Compute validation error: $E^{(i)} = L(f^{(i)}, \mathcal{D}_{val}^{(i)})$
3. Estimate generalization error: $\hat{E}_{CV} = \frac{1}{k} \sum_{i=1}^{k} E^{(i)}$

Error Estimation

$$\hat{E}_{CV} = \frac{1}{k} \sum_{i=1}^{k} L(f^{(i)}, \mathcal{D}_i)$$

$$\text{Var}(\hat{E}_{CV}) = \frac{1}{k^2} \sum_{i=1}^{k} \text{Var}(E^{(i)})$$

Standard Error

$$SE(\hat{E}_{CV}) = \sqrt{\frac{1}{k} \sum_{i=1}^{k} \left(E^{(i)} - \hat{E}_{CV} \right)^2}$$

Confidence interval: $\hat{E}_{CV} \pm 1.96 \cdot SE$

Choice of k in K-Fold Cross-Validation

Common Values:

- **k = 5**: Good balance, widely used
- **k = 10**: Most popular, good bias-variance tradeoff
- **k = n (LOOCV)**: Lowest bias, highest variance

Bias-Variance Tradeoff:

$$\text{Bias} \propto \frac{k-1}{k} \text{ (decreases with k)} \tag{6}$$

$$\text{Variance} \propto \text{correlation between folds} \tag{7}$$

$$\text{Computation} \propto k \text{ (increases with k)} \tag{8}$$

Choosing k

Small k (k = 3-5):

- Higher bias
- Lower variance
- Less computation
- Good for large datasets

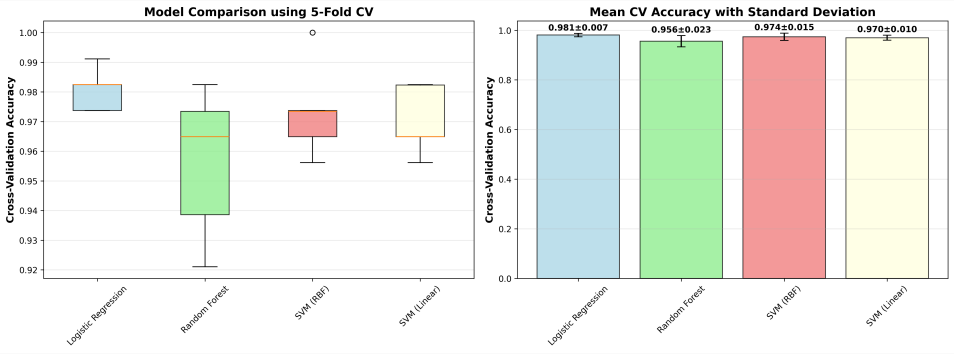
Large k (k = 10-20):

- Lower bias
- Higher variance
- More computation
- Good for small datasets

Recommendation

Use **k = 10** as default choice. It provides good bias-variance tradeoff for most practical scenarios.

K-Fold vs Holdout: Performance Comparison

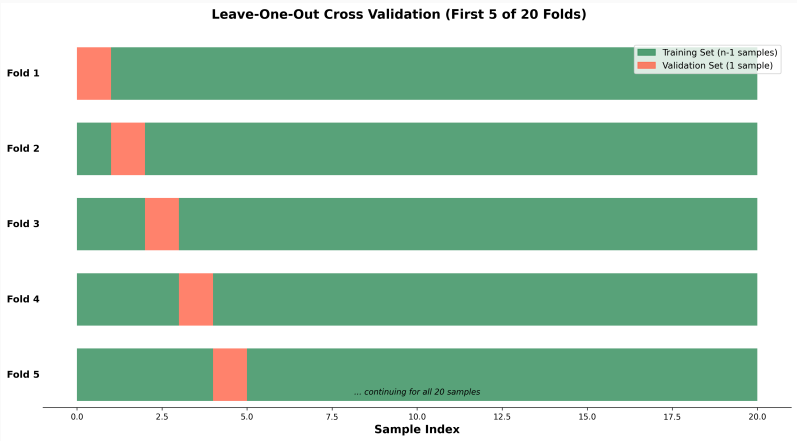


Key Observation

K-fold cross-validation provides more robust and reliable performance estimates, especially important for model comparison and selection.

Other Variants

Leave-One-Out Cross-Validation (LOOCV)



Mathematical Formulation

LOOCV is k-fold CV with $k = n$:

$$\hat{E}_{LOOCV} = \frac{1}{n} \sum_{i=1}^{n} L(f^{(-i)}, (x_i, y_i))$$

Properties

- **Bias**: Nearly unbiased (uses n-1 samples)
- **Variance**: High (high correlation between folds)
- **Computation**: Expensive (n model fits)
- **Deterministic**: No randomness in splits

Stratified K-Fold Cross-Validation

Key Advantage

Maintains class distribution across folds, crucial for:

Specialized Cross-Validation Methods

Time Series Cross-Validation

Problem: Standard CV violates temporal order

Solution: Forward chaining

- Fold 1: Train[1:100], Test[101:150]
- Fold 2: Train[1:150], Test[151:200]
- Fold 3: Train[1:200], Test[201:250]

Preserves: Temporal dependencies

Group K-Fold

Problem: Data points are grouped (e.g., patients, images from same source)

Solution: Ensure entire groups stay together in splits

Prevents: Data leakage between folds

Monte Carlo Cross-Validation

Method: Random sampling approach

- Randomly split data multiple times
- Average performance across all splits
- More flexible than k-fold

Advantage: Can control train/validation ratio

Nested Cross-Validation

Purpose: Unbiased model selection + evaluation

Structure:

- Outer loop: Performance estimation
- Inner loop: Hyperparameter tuning

Result: True generalization estimate

Guideline

Choose validation method based on data characteristics and problem constraints.

Hyper-parameter Search Methods

The Hyperparameter Optimization Problem

Goal: Find optimal hyperparameters λ^*

$$\lambda^* = \arg\min_{\lambda \in \Lambda} \hat{E}_{CV}(\lambda) \tag{9}$$

$$\hat{E}_{CV}(\lambda) = \frac{1}{k} \sum_{i=1}^{k} L(f_\lambda^{(i)}, \mathcal{D}_i) \tag{10}$$

where:

- λ: hyperparameter vector
- Λ: search space
- $f_\lambda^{(i)}$: model trained with λ on fold i

Examples

SVM: $\lambda = (C, \gamma)$

$$\Lambda = [10^{-3}, 10^{3}] \times [10^{-6}, 10^{1}]$$

Random Forest:

$$\lambda = (n_{est}, \text{depth}, \text{features})$$

Neural Network:

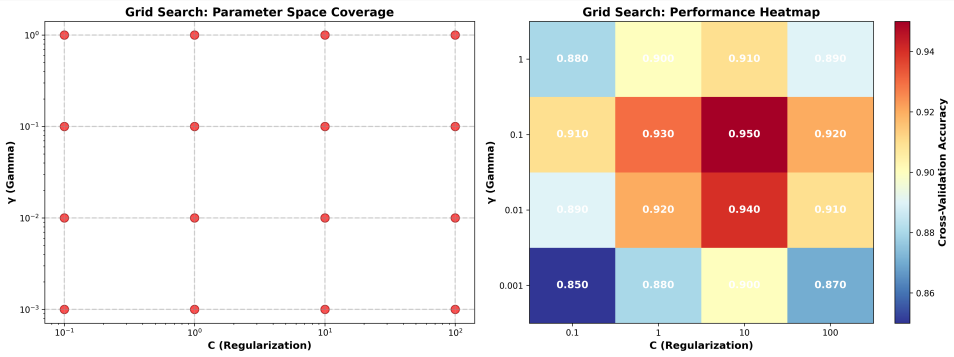
$$\lambda = (lr, \text{batch}, \text{layers}, \text{dropout})$$

Challenge

Hyperparameter spaces are often high-dimensional, mixed-type, and expensive to evaluate.

Grid Search

Grid Search Method



Core Principle

Exhaustive search over a discrete grid of hyperparameter combinations.

Grid Search: Mathematical Formulation

Algorithm:

1. Define search grid: $\Lambda_{grid} = \{\lambda_1, \lambda_2, \ldots, \lambda_m\}$
2. For each $\lambda_j \in \Lambda_{grid}$:
   - Compute $\hat{E}_{CV}(\lambda_j)$ using k-fold cross-validation
   - Store result: $(\lambda_j, \hat{E}_{CV}(\lambda_j))$
3. Return: $\lambda^* = \arg\min_{\lambda_j} \hat{E}_{CV}(\lambda_j)$

Grid Construction

For parameter λ_i with range $[a_i, b_i]$:

Linear spacing:

$$\lambda_i^{(j)} = a_i + \frac{j-1}{n_i - 1}(b_i - a_i)$$

Log spacing (preferred for scale-invariant parameters):

$$\lambda_i^{(j)} = a_i \cdot \left(\frac{b_i}{a_i}\right)^{\frac{j-1}{n_i-1}}$$

Computational Complexity

Total evaluations: $\prod_{i=1}^{p} n_i$

With k-fold CV:

$$\text{Total cost} = k \cdot \prod_{i=1}^{p} n_i \cdot C_{train}$$

where:

- p: number of hyperparameters
- n_i: grid points for parameter i
- C_{train}: cost of training one model

Grid Search: Advantages & Disadvantages

Advantages

- **Exhaustive**: Guarantees finding the best combination on the grid
- **Parallel**: Evaluations are independent
- **Reproducible**: Deterministic results
- **Simple**: Easy to implement and understand
- **Complete coverage**: Systematic exploration

Best Practices

- Use log-uniform grids for scale parameters (C, γ, learning rate)
- Start with coarse grid, refine around promising regions
- Use domain knowledge for range selection

Disadvantages

- **Curse of dimensionality**: Exponential growth with parameters
- **Computational cost**: Can be prohibitively expensive
- **Grid limitations**: Optimal values might lie between grid points
- **Uniform sampling**: Wastes effort in unpromising regions
- **Manual tuning**: Requires careful grid design

When NOT to Use

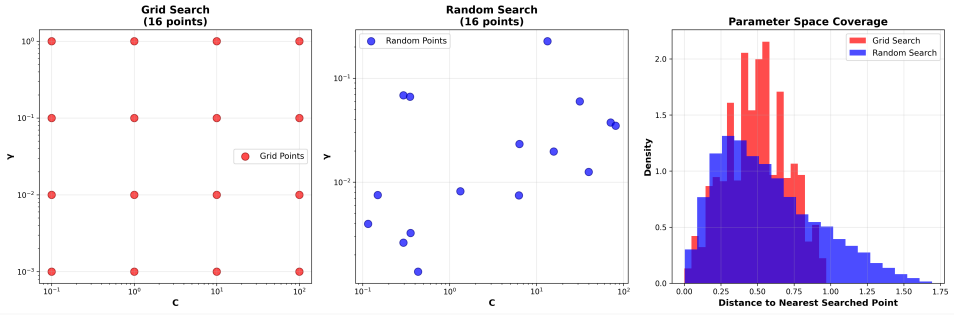
- High-dimensional hyperparameter spaces ($p > 4$)
- Expensive model training (C_{train} very large)
- Continuous optimization required

Rule of Thumb

Grid search is practical for $\leq$ 3-4 hyperparameters with reasonable computational budget.

Random Search

Random Search vs Grid Search



Key Insight

Random search often finds better solutions than grid search with the same computational budget, especially in high dimensions.

Random Search: Method & Theory

Algorithm:

1. Define parameter distributions: $\lambda_i \sim p_i(\lambda_i)$
2. For $j = 1$ to m (budget):
   - Sample: $\lambda^{(j)} \sim \prod_{i=1}^{p} p_i(\lambda_i)$
   - Evaluate: $\hat{E}_{CV}(\lambda^{(j)})$
3. Return: $\lambda^* = \arg\min_j \hat{E}_{CV}(\lambda^{(j)})$

Common Distributions

Continuous parameters:

- Uniform: $\lambda \sim U(a, b)$
- Log-uniform: $\log \lambda \sim U(\log a, \log b)$
- Normal: $\lambda \sim \mathcal{N}(\mu, \sigma^2)$

Discrete parameters:

- Uniform: $\lambda \sim \text{Uniform}\{v_1, v_2, \ldots, v_k\}$

Theoretical Advantage

If only d out of p parameters matter:

Grid search: Need n^p points for n-point resolution

Random search: Need only n points for same effective resolution in important dimensions

Probability of good solution:

$$P(\text{success}) = 1 - (1 - \epsilon)^m$$

where ϵ is fraction of good region

Random Search: Advantages & Best Practices

Advantages

- **Dimension-friendly**: Scales well to high dimensions
- **Efficient**: Often finds good solutions quickly
- **Flexible**: Works with any parameter distribution
- **Parallel**: Evaluations are independent
- **Anytime**: Can stop early if budget is limited
- **Robust**: Less sensitive to irrelevant parameters

Disadvantages

- **No guarantee**: May miss optimal regions
- **Random**: Results vary between runs
- **Distribution-dependent**: Requires good prior knowledge

Best Practices

- **Use log-uniform** for scale parameters:

$$C \sim \text{LogUniform}(10^{-3}, 10^{3})$$

- **Set reasonable bounds** based on domain knowledge
- **Run multiple times** and take the best result
- **Start with random search**, then refine with local methods

Implementation

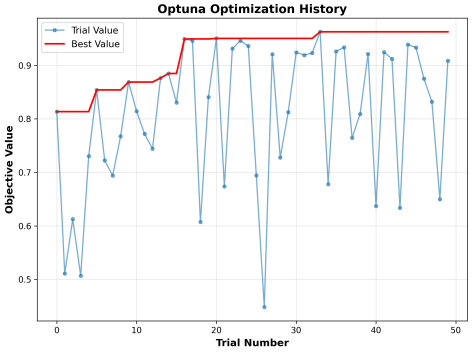
1: Set parameter ranges and distributions
2: **for** $i = 1$ to n_{trials} **do**
3: Sample $\lambda^{(i)}$ from distributions
4: Evaluate $f(\lambda^{(i)})$ using CV
5: **end for**
6: **return** $\arg\min_i f(\lambda^{(i)})$

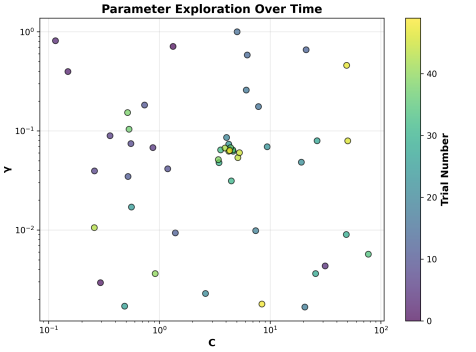
Recommendation

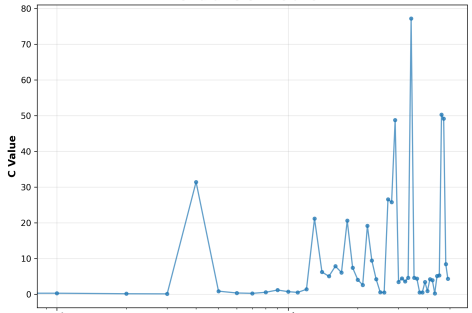
Use random search as default for > 3 hyperparameters or when computational budget is limited.

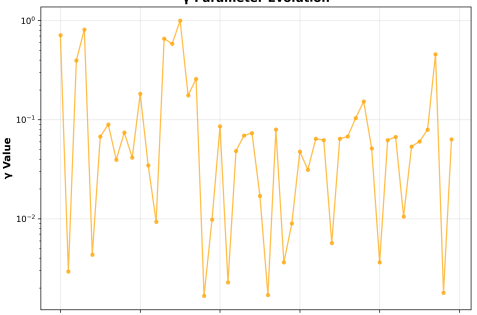
Bayesian Optimization (Optuna)

Bayesian Optimization Intuition









Bayesian Optimization: Mathematical Framework

Core Components:

1. Surrogate Model

Gaussian Process: $f(\lambda) \sim \mathcal{GP}(m(\lambda), k(\lambda, \lambda'))$

Predictive distribution:

$$\mu_n(\lambda) = k_n^T K_n^{-1} \mathbf{y}_n \tag{11}$$

$$\sigma_n^2(\lambda) = k(\lambda, \lambda) - k_n^T K_n^{-1} k_n \tag{12}$$

where:

- k_n: kernel vector at λ
- K_n: kernel matrix of observed points
- $\mathbf{y}_n$: observed function values

2. Acquisition Function

Expected Improvement (EI):

$$EI(\lambda) = \mathbb{E}[\max(f^* - f(\lambda), 0)] \tag{13}$$

$$= (f^* - \mu(\lambda))\Phi(z) + \sigma(\lambda)\phi(z) \tag{14}$$

where $z = \frac{f^* - \mu(\lambda)}{\sigma(\lambda)}$, f^* is current best

Upper Confidence Bound (UCB):

$$UCB(\lambda) = \mu(\lambda) + \kappa\sigma(\lambda)$$

Algorithm:

1. Initialize with random evaluations
2. Fit GP to observed data $\{(\lambda_i, f(\lambda_i))\}_{i=1}^n$
3. Find $\lambda_{n+1} = \arg\max_\lambda \text{Acquisition}(\lambda)$
4. Evaluate $f(\lambda_{n+1})$ and update data
5. Repeat until budget exhausted

Optuna: Tree-Structured Parzen Estimator (TPE)

TPE Algorithm:

1. Split observations by performance quantile γ:
   - Good: $\mathcal{L} = \{\lambda : f(\lambda) < Q_\gamma\}$
   - Bad: $\mathcal{G} = \{\lambda : f(\lambda) \geq Q_\gamma\}$
2. Model densities:
   - $p(\lambda|\mathcal{L})$: density of good configurations
   - $p(\lambda|\mathcal{G})$: density of bad configurations
3. Acquisition function:

$$a(\lambda) = \frac{p(\lambda|\mathcal{L})}{p(\lambda|\mathcal{G})}$$

4. Select: $\lambda_{next} = \arg\max_\lambda a(\lambda)$

TPE Advantages

- **Efficient**: Lower computational cost than GP
- **Flexible**: Handles mixed parameter types
- **Scalable**: Works well in high dimensions
- **Robust**: Less sensitive to hyperparameters

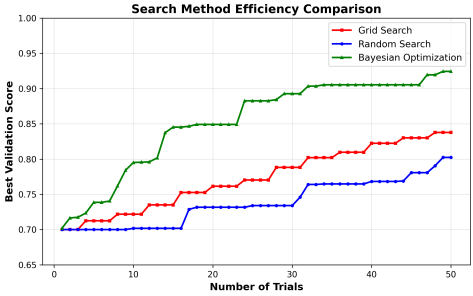
Optuna Features

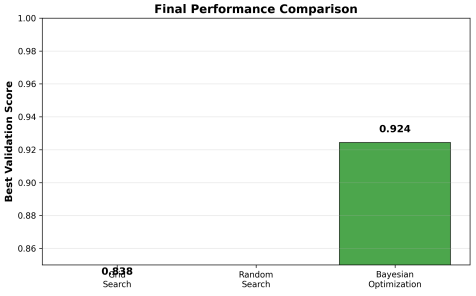
- Pruning: Early stopping of unpromising trials
- Multi-objective optimization
- Distributed optimization
- Integration with ML frameworks

Key Insight

TPE focuses sampling on regions where good configurations are dense relative to bad ones.

Hyperparameter Search: Performance Comparison





Method Comparison

- **Grid Search**: Systematic but inefficient
- **Random Search**: Better exploration, good baseline
- **Bayesian Optimization**: Fastest convergence, most efficient

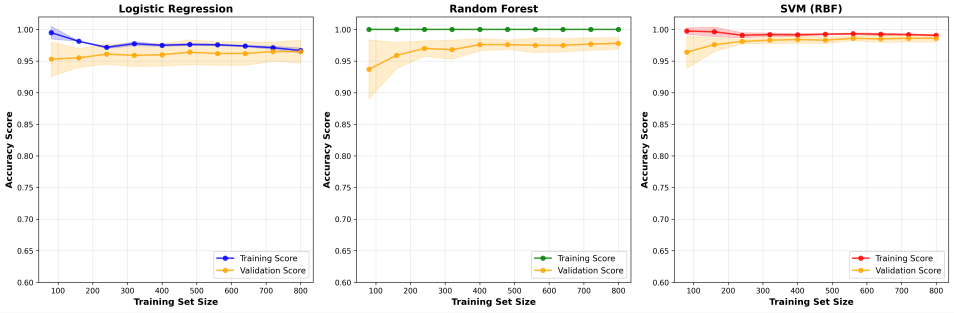
Recommendations

- **Few parameters (≤ 3)**: Grid search
- **Many parameters (> 3)**: Random search
- **Expensive evaluations**: Bayesian optimization (Optuna)
- **Mixed types**: Optuna TPE

Best Practice

Learning Curves & Validation Curves

Learning Curves



Purpose

Learning curves help diagnose **bias** (underfitting) vs **variance** (overfitting) and determine if more data would help.

Learning Curve Analysis

High Bias (Underfitting)

Symptoms:

- Training and validation curves converge
- Both curves plateau at suboptimal level
- Large training error
- Small gap between curves

Solutions:

- Increase model complexity
- Add more features
- Reduce regularization
- Use more flexible model

High Variance (Overfitting)

Symptoms:

- Large gap between training and validation curves
- Training error very low
- Validation error high
- Curves don't converge

Solutions:

- Get more training data
- Reduce model complexity
- Increase regularization
- Use ensemble methods

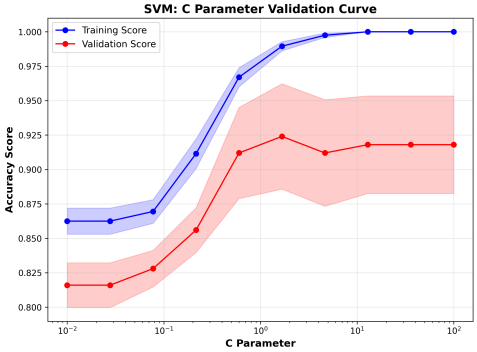
Mathematical Insight

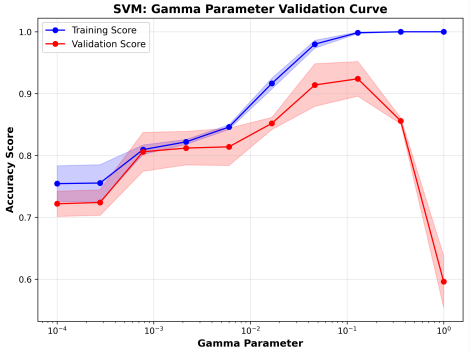
For a learning algorithm with m training examples:

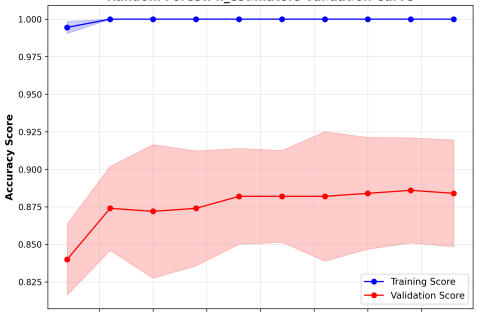
$$\text{Generalization Error} = \text{Bias}^2 + \text{Variance} + \text{Noise}$$

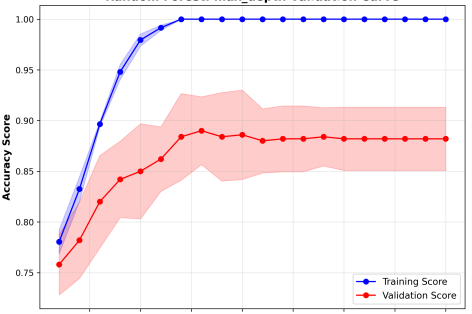
Learning curves show how this decomposes as m increases.

Validation Curves









Interpreting Validation Curves

Typical Pattern Analysis:

Regularization Parameters

(e.g., SVM C parameter, Ridge α)

Low values (high regularization):

- High bias, low variance
- Training and validation errors both high
- Underfitting

High values (low regularization):

- Low bias, high variance
- Large gap between curves
- Overfitting

Optimal region: Minimum validation error

Model Complexity Parameters

(e.g., Random Forest depth, SVM γ)

Mathematical relationship:

$$\text{Complexity} \propto \text{Overfitting Risk}$$

Sweet spot identification:

1. Find minimum validation error
2. Check if training error is reasonable
3. Ensure curves are not diverging rapidly

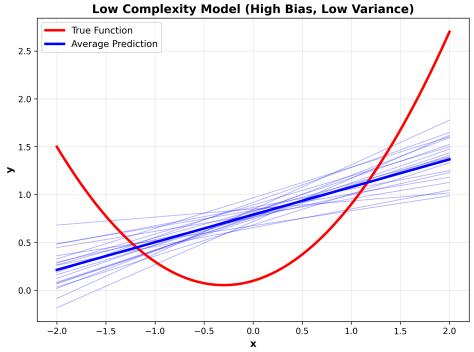
Cross-validation confidence: Error bars show ± 1 standard deviation across CV folds

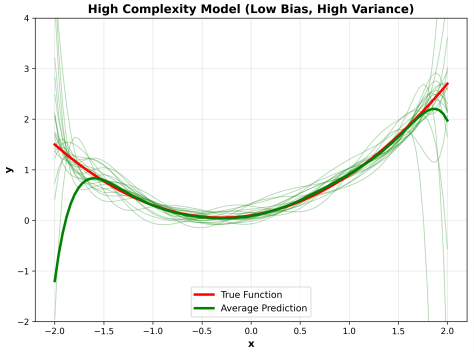
Best Practice

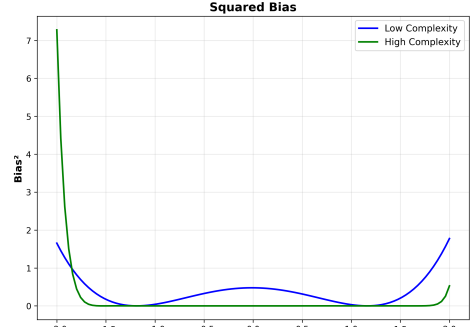
Always plot both training and validation curves with error bars to make informed hyperparameter choices.

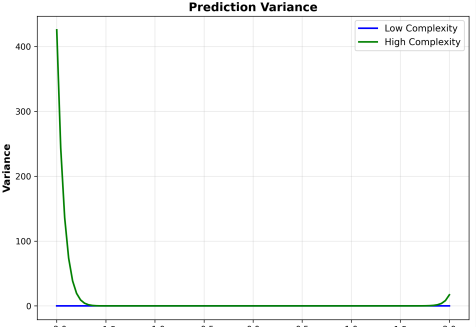
Bias-Variance Tradeoff

Bias-Variance Decomposition









Bias-Variance Mathematical Framework

Decomposition of Expected Test Error:

For a target function $f(x)$ and prediction $\hat{f}(x)$:

$$\mathbb{E}[(\hat{f}(x) - f(x))^2] = \text{Bias}^2[\hat{f}(x)] + \text{Var}[\hat{f}(x)] + \sigma^2$$

where:

$$\text{Bias}[\hat{f}(x)] = \mathbb{E}[\hat{f}(x)] - f(x) \tag{15}$$

$$\text{Var}[\hat{f}(x)] = \mathbb{E}[(\hat{f}(x) - \mathbb{E}[\hat{f}(x)])^2] \tag{16}$$

$$\sigma^2 = \text{Irreducible error (noise)} \tag{17}$$

Cross-Validation & Bias-Variance

CV helps estimate:

- **Bias**: Through training error analysis
- **Variance**: Through CV fold variability
- **Optimal complexity**: Bias-variance tradeoff point

$$\hat{E}_{CV} \approx \text{Bias}^2 + \text{Variance}^2$$

Model Selection Strategy

1. Compute $\hat{E}_{CV}(\lambda)$ for different complexities
2. Plot learning/validation curves
3. Identify minimum of validation curve
4. Check bias-variance indicators:
   - Training error (bias)
   - CV std deviation (variance)

Best Practices & Guidelines

Cross-Validation Best Practices

Data Splitting

- **Stratify** for classification tasks
- **Preserve temporal order** for time series
- **Group-aware splitting** for clustered data
- **Test set isolation**: Never use for model selection

Hyperparameter Search

- Start with **random search** for baseline
- Use **Bayesian optimization** for expensive evaluations
- **Log-uniform sampling** for scale parameters
- **Early stopping** for unpromising configurations

Statistical Considerations

- Report **mean** $\pm$ **std** of CV scores
- Use **paired t-tests** for model comparison
- Account for **multiple testing** when comparing many models
- Consider **McNemar's test** for classification

Computational Efficiency

- **Parallel evaluation** of CV folds
- **Caching** of expensive preprocessing steps
- **Progressive validation** for large datasets
- **Approximate methods** when exact CV is too expensive

Golden Rule

Never use the test set for any decision making during model development. Reserve it solely for final performance evaluation.

Common Pitfalls & How to Avoid Them

Data Leakage

Problem: Information from validation/test leaks into training

Examples:

- Scaling on entire dataset before splitting
- Feature selection using all data
- Temporal leakage in time series

Solution: Apply preprocessing within each CV fold

Look-ahead Bias

Problem: Using future information for prediction

Example: Standard CV on time series data

Solution: Time series CV with forward chaining

Selection Bias

Problem: Multiple testing without correction

Example: Comparing 100 models, reporting best CV score

Solution: Bonferroni correction or separate validation set for selection

Inappropriate CV Method

Problems:

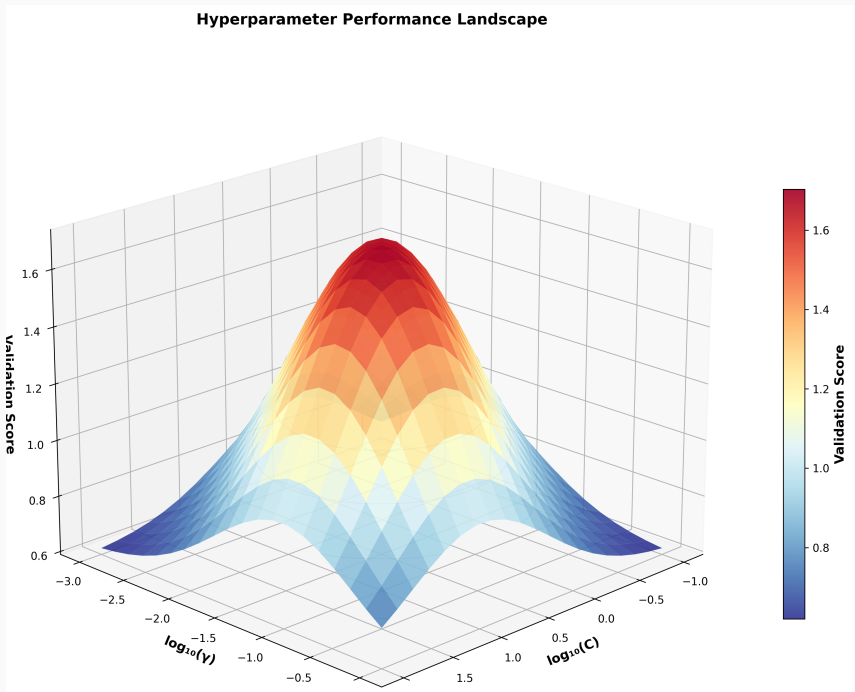
- LOOCV with large datasets (unstable)
- Standard CV with imbalanced data
- Ignoring data structure (groups, time)

Solution: Choose CV method based on data characteristics

Validation

Always ask: *"Does my validation procedure realistically simulate deployment conditions?"*

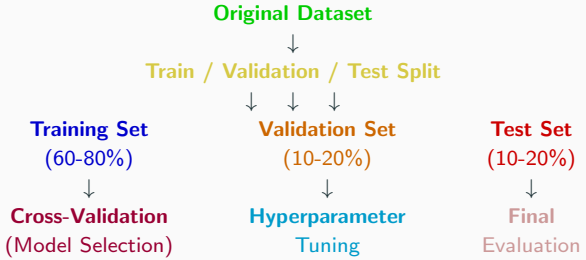
Hyperparameter Search Space



Summary

Summary: Model Validation Hierarchy

Model Validation Hierarchy:



Best Practice Workflow

1. Split data
2. Use CV for model selection
3. Tune hyperparameters on validation set
4. Final evaluation on test set (once!)

Key Takeaways

Validation Methods

- **Holdout**: Fast but high variance
- **K-fold CV**: Best general-purpose method
- **LOOCV**: Unbiased but expensive
- **Stratified**: Essential for imbalanced data
- **Time series CV**: Preserves temporal order

Method Selection Guide

- Small data (n ¡ 1000): 10-fold CV or LOOCV
- Large data (n ¿ 10000): 5-fold CV or holdout
- Imbalanced: Stratified k-fold
- Time series: Forward chaining
- Grouped data: Group k-fold

Hyperparameter Search

- **Grid search**: $\leq$3 parameters, comprehensive
- **Random search**: ¿3 parameters, efficient baseline
- **Bayesian optimization**: Expensive evaluations, smart search
- **Use proper distributions**: Log-uniform for scale parameters

Critical Principles

- Never use test set for model selection
- Avoid data leakage at all costs
- Report confidence intervals (mean $\pm$ std)
- Match validation to deployment conditions
- Use learning curves to diagnose bias/variance

Remember

Cross-validation is not just about getting a number—it's about making principled decisions about model selection, hyperparameter tuning, and understanding model behavior.

Questions?