Lecture 11: Model Validation & Optimization

Luu Minh Sao Khue

How to evaluate, tune, and improve models: cross-validation, hyperparameter search, early stopping, regularization, and bias–variance analysis.

1. Learning Objectives

After this lecture, you will be able to:

- Understand why a model must be validated and tuned before deployment.
- Explain k-fold and stratified cross-validation mathematically.
- Decompose prediction error into bias, variance, and noise.
- Apply L1/L2 regularization to control complexity.
- Perform grid search, random search, and early stopping.
- Interpret learning curves and identify overfitting/underfitting.
- Use feature selection to build parsimonious, interpretable models.

2. Motivation and Intuition

Machine learning models aim to generalize — to perform well on unseen data. However, any model can memorize training data if allowed to grow too complex. We therefore separate data into parts for training, validation, and testing to simulate unseen conditions.

$$\begin{aligned}\text{Training set} &\rightarrow \text{fit parameters}\\ \text{Validation set} &\rightarrow \text{choose hyperparameters}\\ \text{Test set} &\rightarrow \text{report generalization error}\end{aligned}$$

Naively evaluating only on the training data yields an optimistic and misleading estimate of true performance. The goal of model validation is to *estimate test error without actually seeing the test data.*

3. Cross-Validation: Reliable Error Estimation

3.1 Train–Validation–Test Split

Simplest approach: divide the dataset into three disjoint parts:

$$n = n_{\text{train}} + n_{\text{val}} + n_{\text{test}}.$$

Train on the first, tune on the second, and evaluate once on the last. This works when n is large, but wastes data if n is small.

3.2 k-Fold Cross-Validation

To use all data efficiently, we partition the dataset into k equal folds. For each iteration $i = 1, \dots, k$:

$$\text{Train on } D \setminus D_i, \quad \text{Validate on } D_i.$$

Compute the validation loss E_i for each fold and average:

$$\hat{E}_{CV} = \frac{1}{k} \sum_{i=1}^{k} E_i.$$

Common choices: $k = 5$ or $k = 10$.

Advantages:

- All observations are used for both training and validation.
- Reduces variability of the estimated generalization error.
- Works for any estimator that can be refit on subsets.

3.3 Stratified Cross-Validation

In classification, folds must preserve the class proportions to avoid biased estimates:

$$P(y = c \mid \text{fold } i) \approx P(y = c \mid D).$$

3.4 Leave-One-Out (LOO) CV

Extreme case $k = n$: each sample validated once.

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

where $\hat{f}_{-i}$ is trained without point i. LOO is nearly unbiased but computationally heavy.

3.5 Repeated CV

Repeat k-fold CV with different random splits to reduce dependence on a particular partition.

4. Bias–Variance Decomposition and Regularization

Model complexity determines generalization. Let the target be $y = f(x) + \varepsilon$, $E[\varepsilon] = 0$, $\mathrm{Var}(\varepsilon) = \sigma^2$. For estimator $\hat{f}$:

$$E[(y - \hat{f}(x))^2] = \underbrace{(E[\hat{f}(x)] - f(x))^2}_{\text{Bias}^2} + \underbrace{E[(\hat{f}(x) - E[\hat{f}(x)])^2]}_{\text{Variance}} + \sigma^2.$$

Interpretation:

- High bias $\Rightarrow$ underfitting, overly rigid model.
- High variance $\Rightarrow$ overfitting, overly sensitive model.
- Regularization balances bias and variance.

4.1 Regularization as Complexity Control

For linear model $y = X\beta + \varepsilon$, the regularized objective is

$$J(\beta) = \|y - X\beta\|^2 + \lambda\Omega(\beta),$$

where Ω is a penalty.

- Ridge: $\Omega(\beta) = \|\beta\|_2^2$, smooth shrinkage.
- Lasso: $\Omega(\beta) = \|\beta\|_1$, induces sparsity.
- Elastic Net: $\Omega(\beta) = \alpha\|\beta\|_1 + (1 - \alpha)\|\beta\|_2^2$.

As λ increases, coefficients shrink toward zero, increasing bias but lowering variance. The optimal λ minimizes validation error, not training error.

5. Hyperparameter Tuning

5.1 Grid Search

We define a discrete grid of possible hyperparameter values $\Theta = \{\theta_1, \dots, \theta_m\}$ and compute CV score for each:

$$\hat{\theta} = \arg\min_{\theta \in \Theta} \hat{E}_{CV}(\theta).$$

The method is simple but grows exponentially with the number of parameters (the "curse of dimensionality").

5.2 Random Search

Instead of exhaustively enumerating Θ, we sample N combinations from prior distributions $p(\theta)$. Bergstra and Bengio (2012) showed random search often finds near-optimal settings faster because only a few hyperparameters matter significantly.

5.3 Bayesian Optimization (concept)

Later methods use probabilistic surrogate models (e.g., Gaussian Processes) to predict promising regions of hyperparameter space, trading off exploration and exploitation.

5.4 Early Stopping

For iterative learners (boosting, gradient descent), validation loss often decreases, then rises as overfitting begins. Define patience p; stop training if no improvement for p iterations:

$$L_{val}^{(t)} > L_{val}^{(t-p)} \Rightarrow \text{stop}.$$

This prevents overfitting and saves computation.

6. Learning Curves

A learning curve plots performance against training set size or training iteration. Let $E_{train}(m)$ and $E_{val}(m)$ be errors for training size m.

- **Underfitting:** both high and similar errors.
- **Overfitting:** large gap, training low, validation high.
- **Good fit:** both low and close together.

Learning curves answer key questions:

1. Would more data help? (gap shrinks with more data)
2. Is the model too complex or too simple?

7. Feature Selection

Selecting informative variables improves model interpretability and reduces variance.

7.1 Categories

- **Filter methods:** based on statistical tests or correlation (e.g., remove features with $|r| < 0.1$).
- **Wrapper methods:** evaluate subsets using CV (forward, backward, recursive elimination).
- **Embedded methods:** selection built into training (e.g., Lasso zeros out coefficients).

7.2 Forward Selection Example

Start with no features; add the one yielding the lowest CV error until no further improvement. If S is current set, choose:

$$j^* = \arg\min_{j \notin S} \hat{E}_{CV}(S \cup \{j\}).$$

8. Python Example: Cross-Validation and Optimization

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import (train_test_split, KFold,
                                     GridSearchCV, RandomizedSearchCV,
                                     learning_curve)
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from scipy.stats import loguniform

# ====== Load Data ======
X, y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# ====== Pipeline: scaling + model ======
pipe = Pipeline([
    ("scaler", StandardScaler()),
    ("clf", LogisticRegression(max_iter=2000, solver="lbfgs"))
])

# ====== k-Fold Cross-Validation ======
cv = KFold(n_splits=5, shuffle=True, random_state=42)

# ====== Grid Search ======
param_grid = {"clf__C": [0.001, 0.01, 0.1, 1, 10],
              "clf__penalty": ["l2"]}
grid = GridSearchCV(pipe, param_grid, cv=cv, scoring="accuracy")
grid.fit(X_train, y_train)
print("Best parameters (grid):", grid.best_params_)
print("CV Accuracy:", grid.best_score_)

# ====== Random Search ======
param_dist = {"clf__C": loguniform(1e-3, 1e2)}
rand = RandomizedSearchCV(pipe, param_distributions=param_dist,
                          n_iter=10, cv=cv, scoring="accuracy",
                          random_state=42)
rand.fit(X_train, y_train)
print("Best parameters (random):", rand.best_params_)

# ====== Learning Curve ======
train_sizes, train_scores, val_scores = learning_curve(
    grid.best_estimator_, X_train, y_train,
    cv=cv, scoring="accuracy",
    train_sizes=np.linspace(0.1, 1.0, 6)
)
```

```
plt.figure(figsize=(6,4))
plt.plot(train_sizes, train_scores.mean(axis=1), 'o-', label="Train")
plt.plot(train_sizes, val_scores.mean(axis=1), 'o-', label="Validation")
plt.xlabel("Training size")
plt.ylabel("Accuracy")
plt.title("Learning Curves")
plt.legend()
plt.tight_layout()
plt.show()
```

This script demonstrates:

- 5-fold CV to estimate accuracy.
- Grid and random search for hyperparameters.
- Visualization of learning curves to assess generalization.

9. Practical Tips

- Always perform preprocessing (e.g., scaling) inside the CV pipeline to prevent data leakage.
- Stratify folds for imbalanced classification.
- Use random search for large hyperparameter spaces.
- Monitor validation loss for early stopping.
- Plot learning curves for insight into data sufficiency.
- Regularization reduces variance and improves robustness.
- Keep a separate test set for final unbiased evaluation.

10. Summary

Model validation is essential to measure how well a model generalizes. Cross-validation provides an almost unbiased estimate of test error. Regularization, early stopping, and feature selection are tools to reduce overfitting. Hyperparameter optimization (grid, random, or Bayesian) systematically explores model settings. Learning curves visually explain under- and overfitting behavior.

- k-fold CV averages validation error over multiple splits.
- Regularization trades variance for bias.
- Grid search is exhaustive; random search is efficient.
- Early stopping prevents overfitting during training.
- Learning curves show whether adding data helps.
- Feature selection improves interpretability.

11. Exercises

1. Derive the bias–variance decomposition formally for the expected prediction error.
2. Implement 10-fold cross-validation manually using loops.
3. Compare grid and random search on an SVM classifier.
4. Tune λ for Ridge and Lasso using CV and plot error vs. λ.
5. Plot and interpret learning curves for two models of different complexity.
6. Apply recursive feature elimination (RFE) with logistic regression.
7. Discuss the pros/cons of early stopping compared with explicit regularization.