CSE 403: Machine Learning

Chapter 5: Regularization & Evaluations

Md Atikuzzaman
Lecturer
Department of Computer Science & Engineering
atik@cse.green.edu.bd

GREEN UNIVERSITY OF BANGLADESH

Outline

Two Failure Cases of Supervised Learning

Underfitting

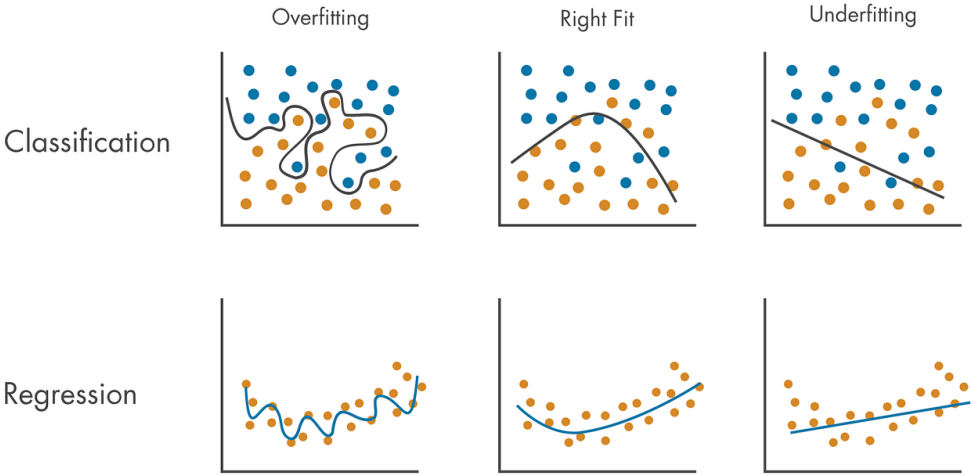
- Model is too simple
- Fails to capture the true pattern
- High training error
- High test error
- **High Bias**
- Low Variance
- Model assumptions are too strong

Overfitting

- Model is too complex
- Learns noise and random fluctuations
- Very low training error
- Poor generalization on test data
- Low Bias
- **High Variance**
- Model is overly sensitive to training data

Bias–Variance Tradeoff: A good model balances bias and variance so that it captures the true pattern while still generalizing well to unseen data.

Underfitting vs Just Right vs Overfitting



Why Model Evaluation Matters

- Training performance alone can be misleading
- A model may memorize instead of learn
- We need objective ways to compare models
- Evaluation helps us:
  - estimate generalization performance
  - tune hyperparameters
  - detect overfitting and underfitting
  - choose the most appropriate model

The goal of evaluation is not to prove a model is perfect, but to estimate how well it will perform on future unseen data.

Training Error vs Generalization Error

Training Error

Error measured on the same data used to fit the model.

Generalization Error

Expected error on new unseen data from the same distribution.

- Low training error does not guarantee low test error
- Reliable evaluation tries to estimate generalization error

Why Do We Need Evaluation?

The Goal

The goal of a classifier is to **generalize** to new, unseen data. High accuracy on the data it was trained on is meaningless.

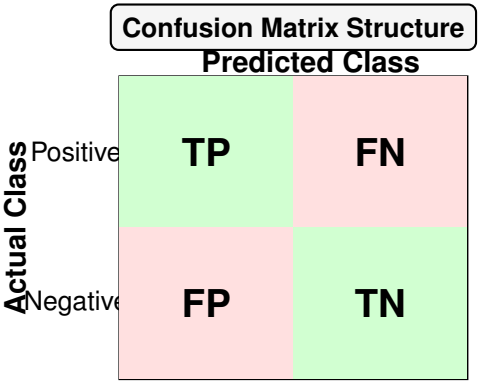
Warning: Never Test on Your Training Set!

A model will "memorize" the training data (overfitting). This leads to a 100% (or very high) accuracy on training data, but it will fail miserably on new data.

The Solution

Always assess a classifier's accuracy using a **separate test set** (or validation set) of class-labeled tuples that the model has **never seen before**.

Confusion Matrix for Binary Classification



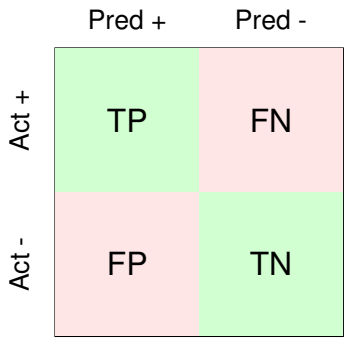
Interpretation

- **TP:** actual positive predicted positive
- **TN:** actual negative predicted negative
- **FP:** actual negative predicted positive
- **FN:** actual positive predicted negative

Why Important?

Most classification metrics are derived from these four quantities.

Metrics Derived from the Confusion Matrix



Key Metrics

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{Specificity} = \frac{TN}{TN + FP}$$

$$F_1 - Score = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Accuracy

How can we measure accuracy?

Accuracy is the percent of all predictions that were correct.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Example

A model classifies 100 tumors (60 malignant, 40 benign):

		Pred: Malignant	Pred: Benign
Actual	Malignant	TP = 50	FN = 10
Actual	Benign	FP = 5	TN = 35

$$\text{Accuracy} = \frac{50 + 35}{50 + 10 + 5 + 35} = \frac{85}{100} = \mathbf{85}\%$$

The Pitfall: Imbalanced Data

Accuracy can be very misleading.

Consider a dataset of 1000 people, where 990 are healthy and 10 have a rare disease.

Let's build a "dumb" classifier that **always predicts "Healthy"**.

"Dumb" Model Confusion Matrix

		Pred: Sick	Pred: Healthy
Actual	Sick	TP = 0	FN = 10
Actual	Healthy	FP = 0	TN = 990

$$\text{Accuracy} = \frac{0 + 990}{0 + 10 + 0 + 990} = \frac{990}{1000} = \mathbf{99}\%$$

This model has 99% accuracy but is **completely useless**, as it fails to identify a single sick patient.

Other Metrics to Consider (1/2)

Precision (Positive Predictive Value)

Of all the times the model predicted "Yes", what percentage was correct?

$$\text{Precision} = \frac{TP}{TP + FP}$$

Use when: The cost of a **False Positive** is high. (e.g., spam detection: you don't want to mark a real email as spam).

Recall (Sensitivity / True Positive Rate)

Of all the actual "Yes" cases, what percentage did the model find?

$$\text{Recall} = \frac{TP}{TP + FN}$$

Use when: The cost of a **False Negative** is high. (e.g., medical diagnosis: you don't want to miss a sick patient).

Other Metrics to Consider (2/2)

F1-Score (Harmonic Mean)

A single score that balances both Precision and Recall. It is the harmonic mean, which punishes extreme (very low) values.

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

This is often the best "single number" metric for imbalanced classes.

Specificity (True Negative Rate)

Of all the actual "No" cases, what percentage did the model correctly identify?

$$\text{Specificity} = \frac{TN}{TN + FP}$$

This is the "Recall" for the negative class.

Methods for Estimating a Classifier's Accuracy

How do we split our data to get a reliable estimate of performance?

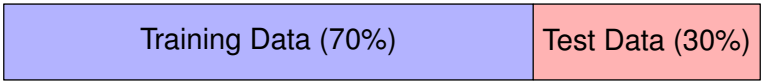
1. The Holdout Method
2. Cross-Validation
3. The Bootstrap

All these methods are ways to generate a test set, which is then used to build the confusion matrix and calculate the metrics.

Estimator 1: The Holdout Method

The simplest method.

1. **Split:** Divide the initial labeled data into two disjoint sets:
   - Training Set (e.g., 70-80% of data)
   - Test Set (e.g., 20-30% of data)
2. **Train:** Build the model using *only* the training set.
3. **Test:** Evaluate the model on the *test set* to get a performance estimate.



Problems

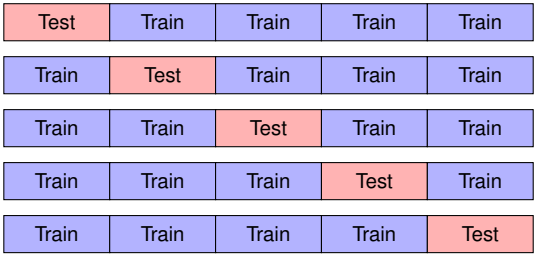
- **Wastes data:** We don't get to train on 30% of our data.
- **High Variance:** The estimate depends heavily on *which* 30% ended up in the test set. A "lucky" or "unlucky" split can give a misleadingly high or low score.

Estimator 2: k-Fold Cross-Validation

The standard solution to the Holdout method's problems.

Process (e.g., 5-Fold CV)

1. **Split:** Divide the data into k (e.g., 5) equal "folds".
2. **Iterate:** Run k experiments:
   - **Fold 1:** Test on F 1, Train on F 2, 3, 4, 5.
   - ... ...
   - **Fold 5:** Test on F 5, Train on F 1, 2, 3, 4.
3. **Average:** The final performance is the **average** of the 5 individual scores.



Advantages

Much more robust estimate. Mitigates the "unlucky split" problem. Uses all data for both training and testing.

Estimator 3: The Bootstrap

Process (Used when the dataset is small)

1. **Sample:** Given a dataset D of n tuples, create a new "bootstrap sample" D_i by sampling n tuples from D **with replacement**.
2. **Split:** D_i becomes the training set. The tuples from D that were *not* selected for D_i form the test set (the "out-of-bag" samples).
3. **Train & Test:** Train the model on D_i and test it on the out-of-bag samples.
4. **Repeat:** Repeat this process many times (e.g., $b = 200$ times).
5. **Average:** The final accuracy is the average of all b test accuracies.

Example of Bootstrap Sampling (n=10)

Original D: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
Sample D_1: {2, 1, 7, 10, 2, 5, 9, 1, 6, 5} $\rightarrow$ *Train*
Test Set 1: {3, 4, 8} $\rightarrow$ *Test*

Comparing Classifiers (1/2): ROC Curves

Receiver Operating Characteristic (ROC) Curve

Many classifiers don't just output "Yes" or "No". They output a *probability* (e.g., 0.85). We then use a **threshold** (e.g., > 0.5) to make the decision.
An ROC curve visualizes a classifier's performance across **all possible thresholds**.

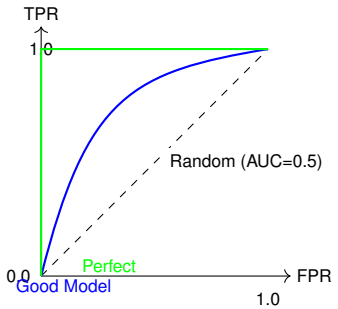
ROC Axes:

- **Y-Axis:** True Positive Rate (Recall)

$$\text{TPR} = \frac{TP}{TP + FN}$$

- **X-Axis:** False Positive Rate

$$\text{FPR} = \frac{FP}{FP + TN}$$



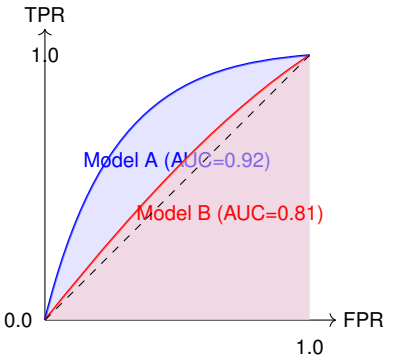
Comparing Classifiers (2/2): AUC

Area Under the Curve (AUC)

It's hard to compare two ROC curves just by looking. The **Area Under the Curve (AUC)** converts the entire curve into a single number.

Interpreting AUC:

- **AUC = 1.0:** Perfect classifier.
- **AUC = 0.5:** Random guessing.
- **AUC < 0.5:** Worse than random (model is backwards).



Comparison

You have two models:

- Model A (AUC = 0.92)
- Model B (AUC = 0.81)

Conclusion: Model A is better at distinguishing between "Yes" and "No"

Common Evaluation Metrics for Regression

- **MSE:**

$$\frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - \hat{y}^{(i)})^2$$

- **RMSE:**

$$\sqrt{\frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - \hat{y}^{(i)})^2}$$

- **MAE:**

$$\frac{1}{m}\sum_{i=1}^{m}|y^{(i)} - \hat{y}^{(i)}|$$

Coefficient of Determination: R^2

Definition

$$R^2 = 1 - \frac{\sum_{i=1}^{m}(y^{(i)} - \hat{y}^{(i)})^2}{\sum_{i=1}^{m}(y^{(i)} - \bar{y})^2}$$

where

$$\bar{y} = \frac{1}{m}\sum_{i=1}^{m} y^{(i)}$$

- Measures the fraction of variance explained by the model
- Closer to 1 usually indicates better fit

Comparing Regression Metrics

Metric	Interpretation	Outlier Sensitivity
MSE	Average squared error	High
RMSE	Error in original unit	High
MAE	Average absolute error	Medium
R^2	Explained variance	Depends on context

- Use multiple metrics for a fuller view
- Pick metrics based on the real application cost

Summary of Evaluation

- **Never test on training data.** Always use a separate test set.
- **Accuracy is not enough.** It is misleading for imbalanced datasets.
- **Use the right metric.**
  - Use **Precision** if False Positives are "expensive".
  - Use **Recall** if False Negatives are "expensive".
  - Use **F1-Score** for a balance.
- **Use the right estimation method.**
  - **Holdout** is simple but has high variance.
  - **k-Fold Cross-Validation** is the industry standard.
  - **Bootstrap** is good for very small datasets.
- **Use ROC/AUC to compare models.** It provides a comprehensive view of a model's performance across all thresholds.

Regularization: Motivation and Intuition

- Complex models can fit noise in training data
- We want to discourage overly flexible solutions
- Regularization adds a **penalty term** to the objective

Generic Regularized Objective

$$J_{\text{reg}}(\theta) = J(\theta) + \lambda\,\Omega(\theta)$$

- $J(\theta)$ = data fitting term
- $\Omega(\theta)$ = complexity penalty
- $\lambda \geq 0$ controls regularization strength

Why Regularization Helps

Without Regularization	With Regularization
Parameters may become very large	Penalizes extreme parameter values
Model may fit noise	Simpler model
High variance	Better generalization

L2 Regularization (Ridge)

Objective

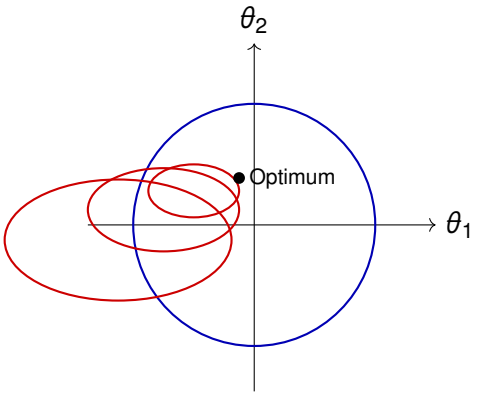
$$J_{L2}(\theta) = J(\theta) + \lambda \sum_{j=1}^{d} \theta_j^2$$

- Penalizes large weights quadratically
- Encourages smaller, smoother parameter values
- Usually does not force coefficients exactly to zero

L2 regularization is also called weight decay in many ML contexts.

Geometric Intuition of L2 Regularization

- L2 constrains parameters inside a circular or spherical region
- The optimal solution tends to have small distributed weights



L1 Regularization (Lasso)

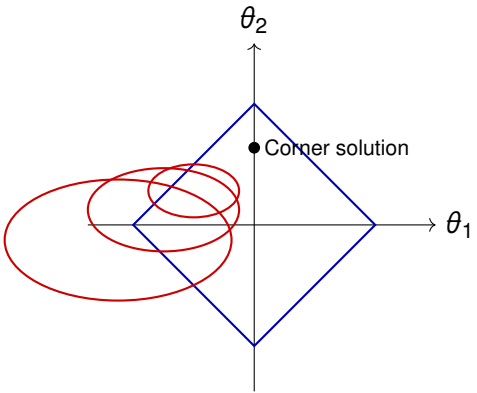
Objective

$$J_{L1}(\theta) = J(\theta) + \lambda \sum_{j=1}^{d} |\theta_j|$$

- Penalizes absolute values of weights
- Encourages sparsity
- Can drive some coefficients exactly to zero

Why L1 Produces Sparse Models

- L1 constraint has diamond-shaped geometry
- Optimization often hits corners
- Corners correspond to zero values for some coefficients



L1 vs L2 Regularization

Property	L1	L2
Penalty type	$\sum \lvert\theta_j\rvert$	$\sum \theta_j^2$
Encourages sparsity	Yes	No
Produces zero coefficients	Often	Rarely
Stability	Lower	Higher
Common use	Feature selection	Weight shrinkage

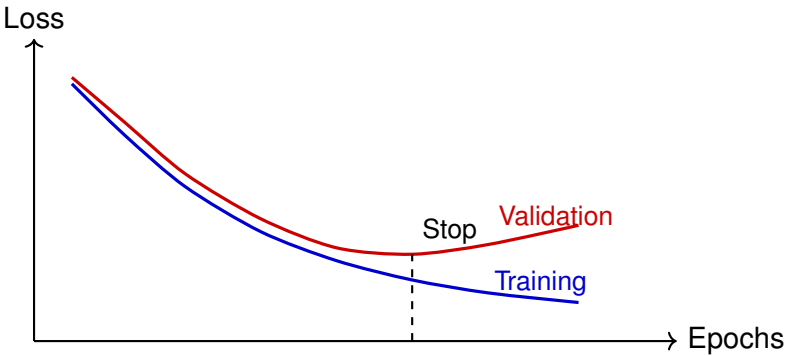
Early Stopping

Idea

Stop training when validation performance stops improving, even if training loss continues decreasing.

- Useful in iterative learning algorithms
- Acts as an implicit regularizer
- Prevents the model from overfitting the training data

Training Loss vs Validation Loss



What are Ensemble Methods?

The Core Idea: "Wisdom of the Crowd"

Ensemble methods combine the predictions of multiple machine learning models (called "weak learners") to create a single, more robust, and accurate model (a "strong learner").

Why use them?

- **Reduce Variance:** Less sensitive to the specific training data. (e.g., Bagging)
- **Reduce Bias:** Correct for the errors of previous models. (e.g., Boosting)
- **Improve Performance:** By blending the strengths of different models. (e.g., Stacking)

Model 1 $\rightarrow$ Prediction 1
Model 2 $\rightarrow$ Prediction 2
Model 3 $\rightarrow$ Prediction 3

$\Downarrow$

Final Ensemble Prediction

Bagging (Bootstrap Aggregating)

How it Works: Parallel Training

The goal is to reduce variance by averaging out the noise.

1. **Bootstrap:** Create N random subsets of the training data (sampling *with replacement*).
2. **Train:** Train N models (e.g., Decision Trees) in parallel, one on each subset.
3. **Aggregate:** Combine the results.
   - **Classification:** Majority Voting
   - **Regression:** Averaging

Key Example: Random Forest

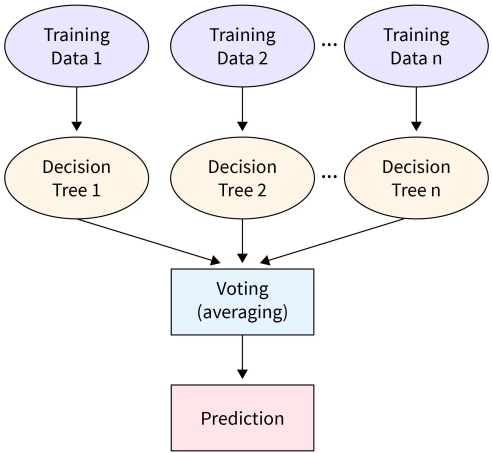
A Random Forest is a Bagging method using Decision Trees. It adds one more layer of randomness: at each split, the tree only considers a *random subset of features*.

Random Forest

What is Random Forest?

Random Forest is an **ensemble learning method** that builds many decision trees and combines their predictions.

- Uses **Bagging (Bootstrap Sampling)**
- Each tree is trained on a different random subset of the data
- At each split, a **random subset of features** is considered
- Final prediction is obtained by combining all trees



Boosting

How it Works: Sequential Training

The goal is to reduce bias by learning from mistakes.

1. Train a simple model (Learner 1) on the data.
2. Identify misclassified samples. Increase their "weight" or importance.
3. Train a new model (Learner 2) that is forced to focus on the high-weight (difficult) samples.
4. Repeat, with each new learner correcting the errors of the previous ones.
5. **Aggregate:** Combine all learners using a weighted vote (better-performing learners get a higher say).

Key Examples

AdaBoost (Adaptive Boosting), **Gradient Boosting** (GBM), **XGBoost**

Stacking (Stacked Generalization)

How it Works: Learning to Combine

The goal is to blend different models to capture different patterns in the data.

Level 0: Base Learners

- Train several *different* models (e.g., SVM, Random Forest, k-NN) on the training data.
- Generate their predictions for the data (often using k-fold cross-validation to prevent data leakage).

Level 1: Meta-Learner

- The predictions from Level 0 are now used as input features for a new, final model.
- This "meta-learner" (e.g., Logistic Regression) learns the best way to combine the base learner predictions.

(Data $\rightarrow$ [SVM, RF, k-NN]) $\rightarrow$ (Predictions) $\rightarrow$ [Logistic Regression] $\rightarrow$ Final Output

Ensemble Comparison

Feature	Bagging	Boosting	Stacking
Model Building	Parallel	Sequential	Parallel, then Sequential
Main Goal	Reduce Variance	Reduce Bias	Improve Predictions
Model Type	Homogeneous	Homogeneous	Heterogeneous
Key Idea	Averaging	Weighted Voting	Learning to Combine
Example	Random Forest	XGBoost	Custom Blends

The Challenge: Imbalanced Data

What is it?

A classification dataset where one class is much more frequent than the other.

- **Majority Class:** The common one.
- **Minority Class:** The rare one.

Examples

- Fraud Detection (99.9% Not Fraud)
- Medical Diagnosis (98% Healthy)
- Ad Click-Through (99.5% No Click)

The "Accuracy Paradox"

- Imagine a 99% / 1% split.
- A "dumb" model that always predicts the **majority class** is 99% accurate.
- ...but it is completely useless, as it never finds the minority class (which is usually the one we care about!)

Solution: Stop using accuracy. Focus on metrics like **Precision**, **Recall**, **F1-Score**, and **AUC-ROC**.

Strategy 1: Data-Level Solutions (Resampling)

Undersampling

- **What:** Randomly remove samples from the majority class.
- **Pro:** Reduces dataset size, speeds up training.
- **Con:** Can lose important information and patterns from the majority class.

Oversampling

- **What:** Randomly duplicate samples from the minority class.
- **Pro:** No information is lost.
- **Con:** Can lead to overfitting, as the model sees the same exact samples multiple times.

A Better Way: SMOTE

SMOTE (Synthetic Minority Over-sampling Technique)

- Instead of duplicating, it creates new synthetic samples.
- It selects a minority sample, finds its neighbors, and creates a new point on the line segment between them.
- This provides new, plausible data and avoids simple overfitting.

Strategy 2 : Algorithm Solutions

Algorithm-Level (Cost-Sensitive Learning)

- **What:** Modify the model's loss function to penalize misclassifying the minority class more heavily.
- **Example:** Tell the model that a "False Negative" (missing a fraud case) is 100$\times$ worse than a "False Positive" (flagging a good transaction).
- Many models have a `class_weight='balanced'` parameter that does this automatically.

Strategy 3: Ensemble Solutions

Ensemble-Level Solutions

- **Balanced Bagging (e.g., Balanced Random Forest):**
- Each "bag" (bootstrap sample) is created by undersampling the majority class.
- The final ensemble is trained on many different, balanced datasets.
- **Boosting (e.g., AdaBoost):**
- Boosting algorithms naturally increase the weight of misclassified (hard) samples.
- The minority class is often "hard," so the model automatically learns to focus on it.

Key Takeaways

- **Underfitting** happens when the model is too simple. **Overfitting** happens when it is too complex.
- The real objective of machine learning is **generalization**, not memorization.
- Proper **evaluation** requires unseen data.
- For classification, common metrics include accuracy, precision, recall, F1-score, and ROC-AUC.
- For regression, common metrics include MSE, RMSE, MAE, and R^2.
- **Holdout**, **k-fold cross-validation**, and **bootstrap** are standard ways to estimate performance.
- **Regularization** controls complexity:
  - L2 shrinks weights smoothly
  - L1 encourages sparsity
  - early stopping halts before overfitting grows
- **Ensembles** combine multiple learners to improve robustness.
- For **imbalanced data**, use appropriate metrics and resampling or cost-sensitive methods.

References

1. T. Hastie, R. Tibshirani, J. Friedman, *The Elements of Statistical Learning*, Springer.
2. C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer.
3. I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning*, MIT Press.
4. K. P. Murphy, *Machine Learning: A Probabilistic Perspective*, MIT Press.