上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Lecture 13: Model Evaluation

Tao Huang

John Hopcroft Center, School of Computer Science, Shanghai Jiao Tong University

https://taohuang.info/cs3317

https://oc.sjtu.edu.cn/courses/89538

1. Evaluation Fundamentals

Recap: We Have Many Models

Module 3 (ML) gave us a full toolkit:

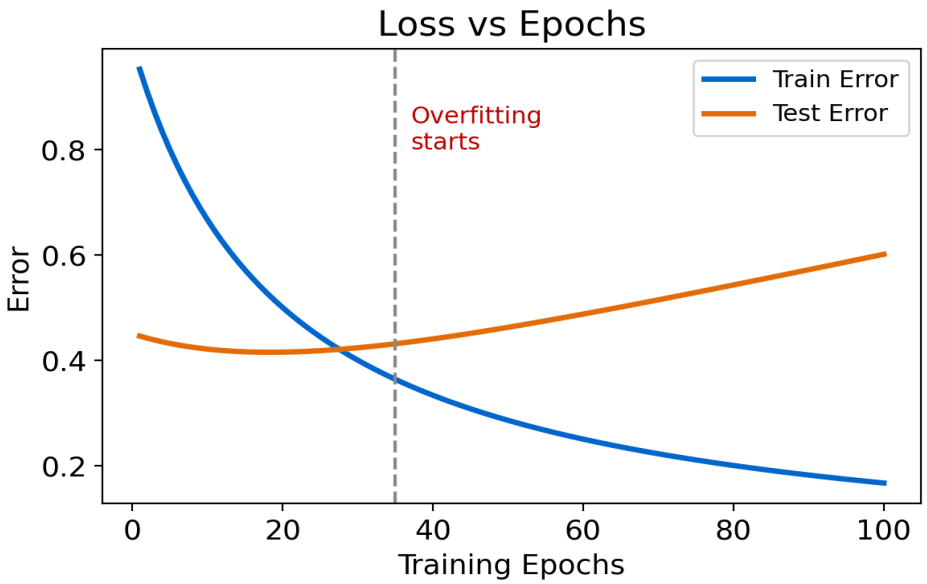
- L9: Linear Regression
- L9: Logistic Regression
- L13: SVM (linear, kernel)
- L14: Decision Trees, Random Forests, Gradient Boosting

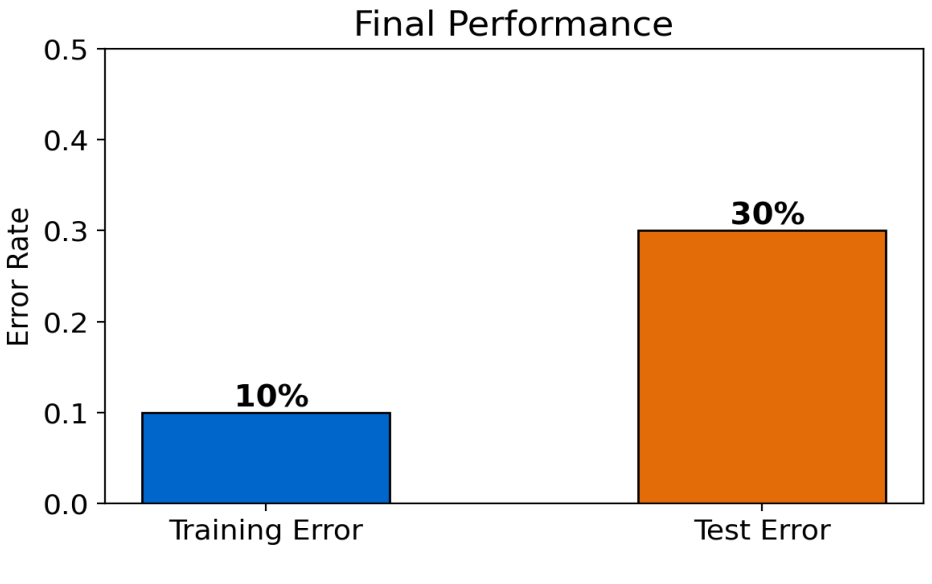
- Each has hyperparameters: C, γ, depth, #trees, η …

- **Question: How do we fairly compare them?**
- **How do we choose the best one?**

Why Not Just Use Training Accuracy?

- **We already know the answer from L8: overfitting!**
- A model that memorizes training data: 100% training accuracy
- But terrible performance on new data
- **Low training loss ≠ good model.**
- **We must evaluate on unseen data.**





Train / Test Split

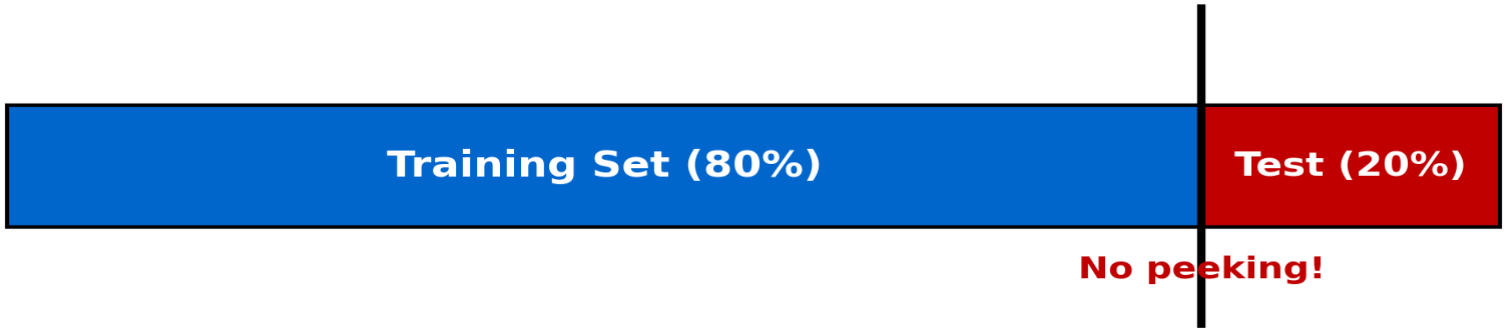
Simplest approach: split data into two parts

- Training set (80%): learn model parameters
- Test set (20%): evaluate performance

Problem:

- We can only evaluate once on the test set.
- If we tune hyperparameters based on test performance, we are cheating — information leaks from test to training.

The test set must be truly held out!



The Three-Way Split

Solution: Train / Validation / Test

- Training set (60%): learn parameters (w, b)
- Validation set (20%): tune hyperparameters (C, depth…)
- Test set (20%): final evaluation (touch only once!)

Workflow:

1. Train models with different hyperparams on train set
2. Evaluate each on validation set
3. Pick the best, then evaluate once on test set

Problem with Fixed Splits

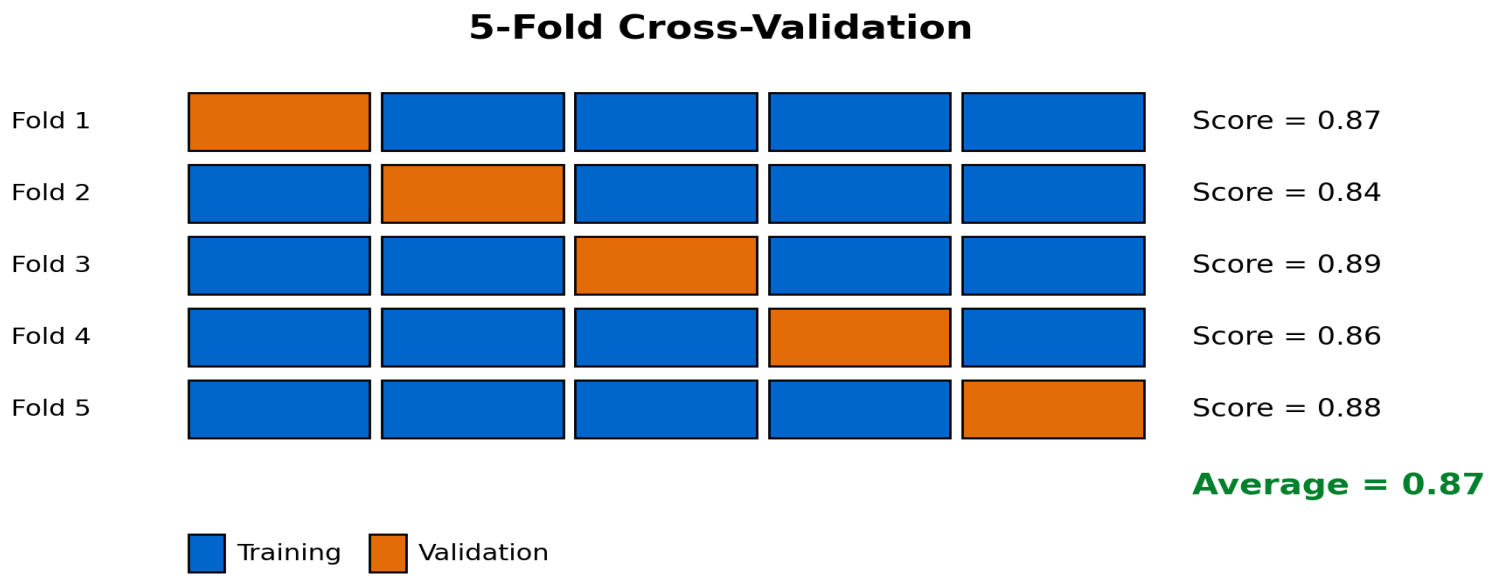
With a fixed validation set:

- Estimate depends on which data ended up in validation
- Small datasets: the split matters a lot (high variance)
- We waste 20% of data that could help training

- **Question: Can we use all data for both training and evaluation?**
- **Answer: Yes! → Cross-Validation**

Cross-Validation (K-Fold CV)

- **Split data into K equal folds (typically K=5 or 10)**
- For each fold i = 1, …, K:
  - Use fold i as validation
  - Use remaining K−1 folds as training
  - Record validation score
- **Final score = average of all K scores**
- Every data point is used for both training and validation (but never at the same time!)



🧠 Think: Why K=5 or K=10?

- K=2: only 50% data for training → model too weak
- K=n (Leave-One-Out): nearly all data for training,
  but very expensive and high variance
- K=5 or K=10: good bias-variance balance

- Computational cost: K × cost of single training run

Discussion:

- What if K=n and you have 1 million samples?

  → 1 million training runs. Not practical!

- **5-fold CV is the most common default in practice.**

Stratified Cross-Validation

Problem: imbalanced classes (e.g., 90% neg, 10% pos)

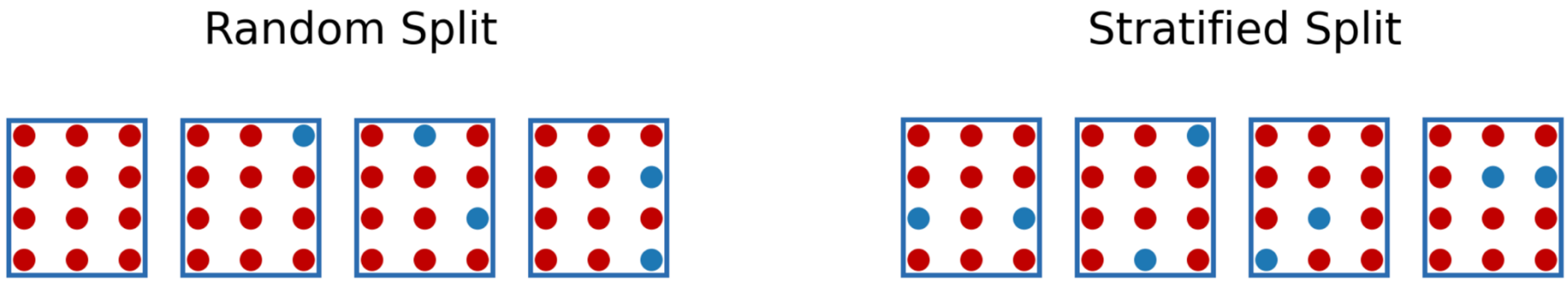
- A random fold might have 0 positive examples!

  → Meaningless evaluation on that fold

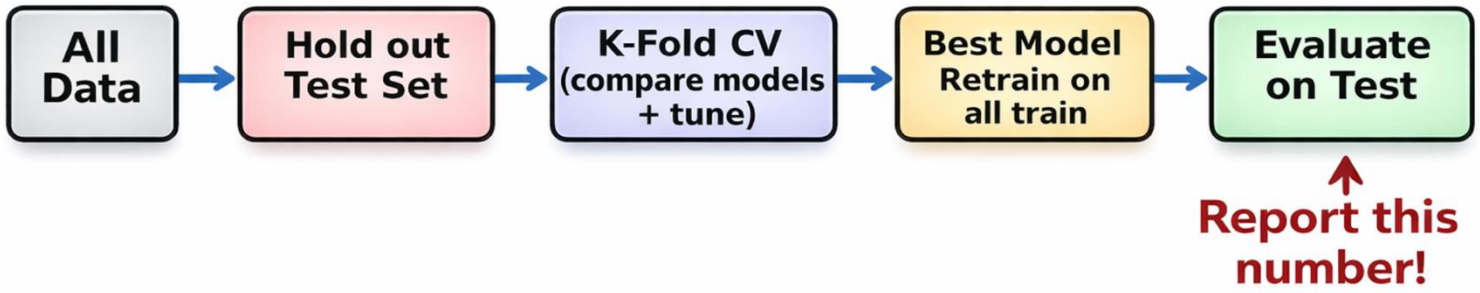
Solution: Stratified K-fold

- Each fold preserves the original class distribution.

Rule: Always use stratified CV for classification.



The Full Evaluation Pipeline



- **Step 1: Hold out test set (never touch until end)**
- **Step 2: K-fold CV on remaining data to:**
  - Compare models (LR vs SVM vs RF vs …)
  - Tune hyperparameters (C, depth, etc.)
- **Step 3: Retrain best model on all training data**
- **Step 4: Evaluate once on test set**

→ Report this number as final performance

2. Classification Metrics

Beyond Accuracy

Accuracy = correct / total

- Seems reasonable, but can be very misleading!

Example: disease screening, 1% prevalence

- Model always predicts healthy: 99% accuracy!
- But misses every sick patient — completely useless.

We need metrics that capture different types of errors.

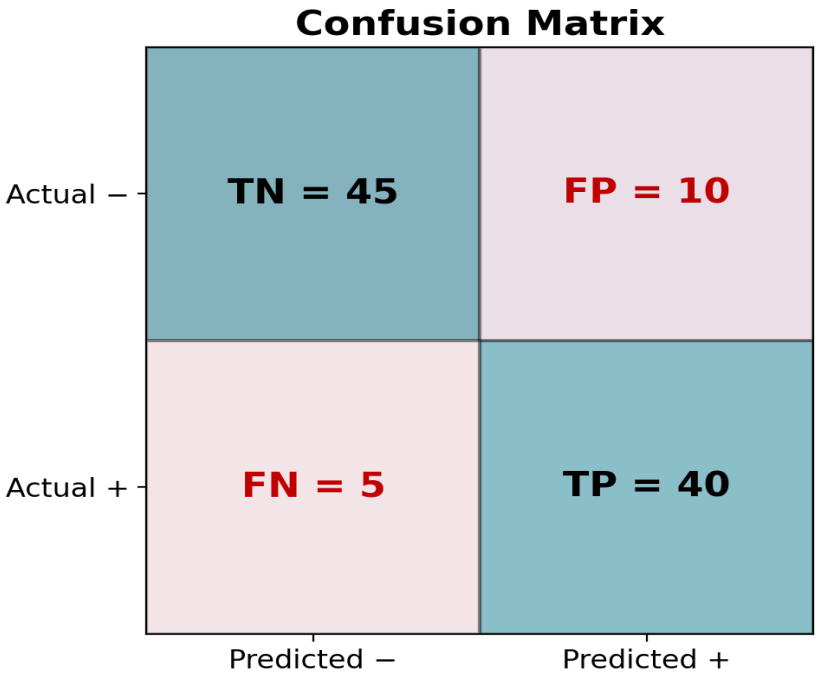
The Confusion Matrix

Four outcomes for binary classification:

- True Positive (TP): predicted +, actually +
- True Negative (TN): predicted −, actually −
- False Positive (FP): predicted +, actually −

  (Type I error / false alarm)
- False Negative (FN): predicted −, actually +

  (Type II error / missed detection)



🧠 Think: Spam Detection Errors

In spam detection:

- FP = legitimate email classified as spam

  → User misses important email!

- FN = spam email classified as legitimate

  → User sees annoying spam

Which error is worse? Why?

- Missing important email (FP) is usually worse!
- Different applications weight errors differently.

Precision and Recall

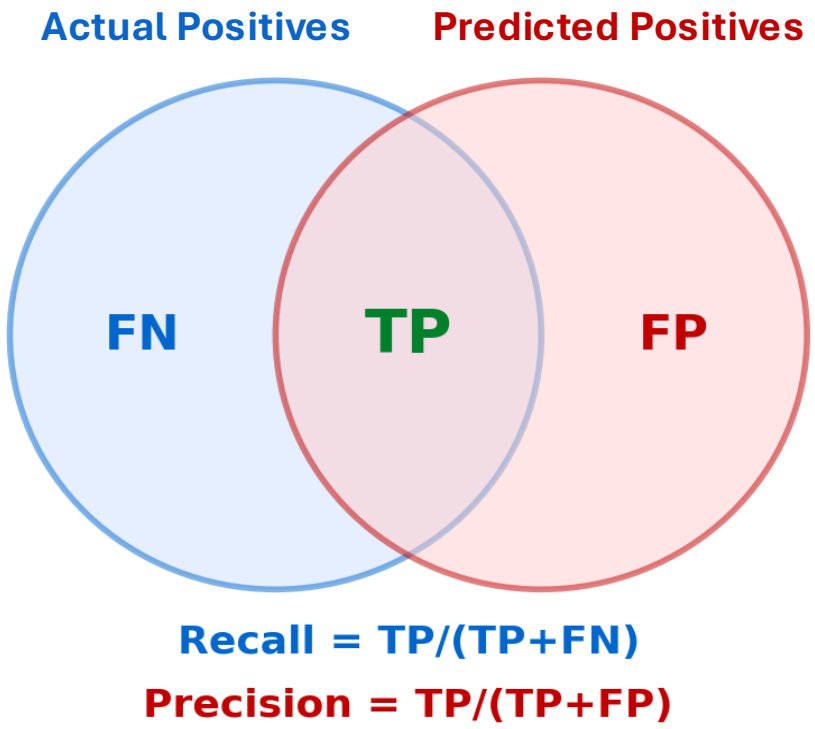
Precision = TP / (TP + FP)

- Of all predicted positives, how many are correct?
- High precision = few false alarms

Recall = TP / (TP + FN)

- Of all actual positives, how many did we catch?
- High recall = few missed positives

There is a fundamental tension between the two.



Precision-Recall Tradeoff

By adjusting the decision threshold:

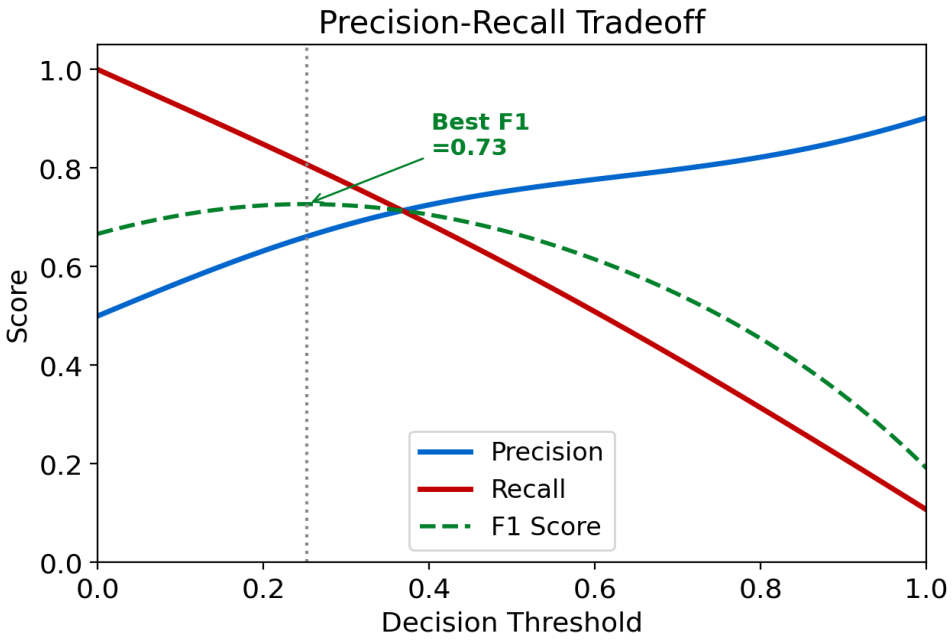
- Lower threshold → predict more positives

  Recall ↑ but Precision ↓

- Higher threshold → predict fewer positives

  Precision ↑ but Recall ↓

- You cannot maximize both simultaneously (unless your model is perfect).



F1 Score

F1 = 2 × (Precision × Recall) / (Precision + Recall)

Harmonic mean（调和平均值） of precision and recall.

- F1 = 1: both precision and recall are perfect
- F1 = 0: either precision or recall is 0

Why harmonic mean (not arithmetic mean（算数平均值）)?

- It penalizes extreme imbalance:
  P=1.0, R=0.01 → arithmetic=0.505, harmonic=0.02
- Harmonic mean forces both to be high!

Example: Compute All Metrics

Given: TP=40, FP=10, FN=5, TN=45

- Accuracy = (40+45)/100 = 85.0%
- Precision = 40/(40+10) = 80.0%
- Recall = 40/(40+5) = 88.9%
- F1 = 2×(0.80×0.889)/(0.80+0.889) = 84.2%

Now try: TP=1, FP=0, FN=49, TN=50

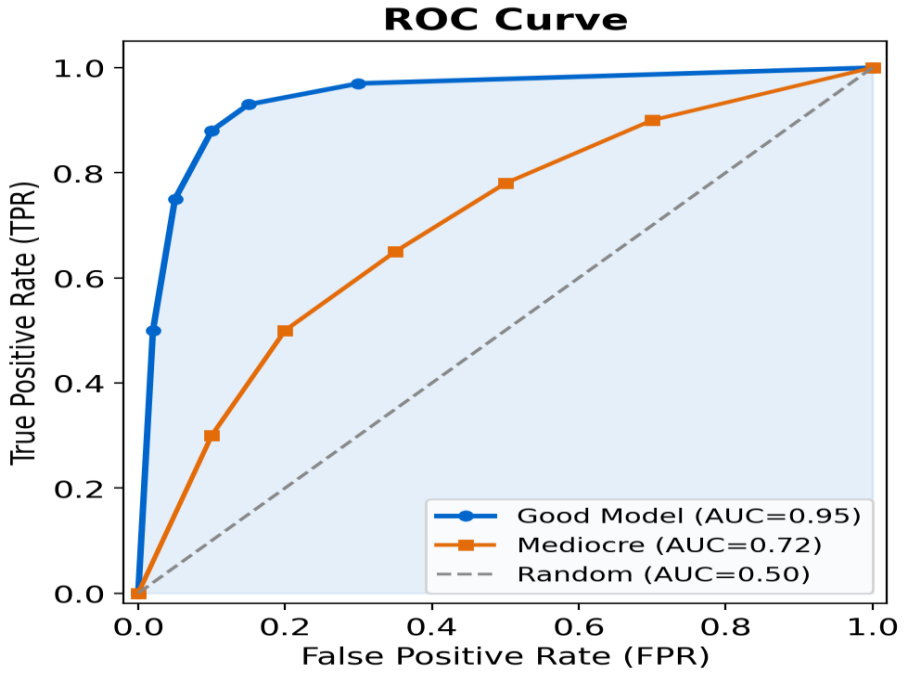
- Accuracy = 51%, Precision = 100%
- Recall = 2%, F1 = 3.9%
- High precision but terrible recall!

ROC Curve

- ROC = Receiver Operating Characteristic

- **Plot TPR vs FPR at various thresholds:**
  - TPR = TP/(TP+FN) = Recall
  - FPR = FP/(FP+TN)

- Good model: curve hugs top-left corner
  - Random model: diagonal line (TPR = FPR)
  - Perfect model: goes straight up then right



AUC: Area Under the ROC Curve

AUC summarizes the ROC curve in one number:

- AUC = 1.0: perfect model
- AUC = 0.5: random guessing
- AUC = 0.0: perfectly wrong (flip predictions!)

Interpretation:

- Probability that a random positive is scored higher than a random negative.

- **AUC is threshold-independent — great for model comparison.**

🧠 When to Use Which Metric?

- Accuracy: balanced classes, equal error costs
- Precision: when FP is costly
  (spam filter: don't lose real email)
- Recall: when FN is costly
  (cancer screening: don't miss patients)
- F1: single number balancing P and R
- AUC: model comparison, imbalanced data

No single best metric — it depends on the application!

Metrics for Regression

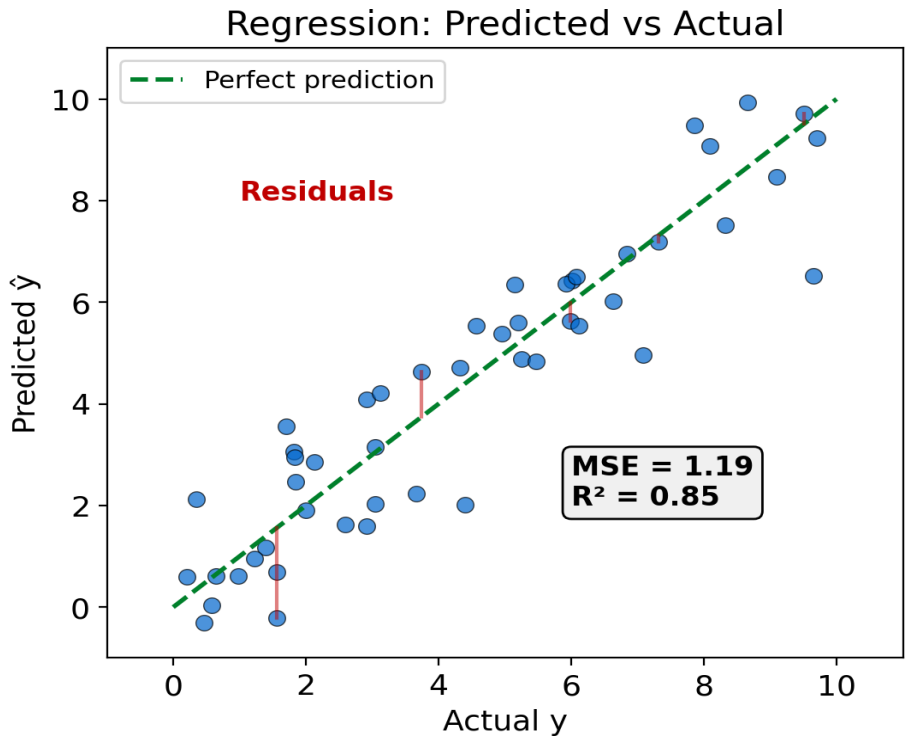
- **MSE =** $\frac{1}{n}\sum_i (y_i - \hat{y}_i)^2$

Penalizes large errors heavily

- **MAE =** $\frac{1}{n}\sum_i |y_i - \hat{y}_i|$

More robust to outliers

- **R^2 =** $1 - \frac{\text{MSE}}{\text{Var}(y)}$
  - Fraction of variance explained
  - R^2=1: perfect, R^2=0: predicting the mean



3. Hyperparameter Tuning

Parameters vs Hyperparameters

Parameters: learned from data

- w, b — set by optimization (GD, SGD)

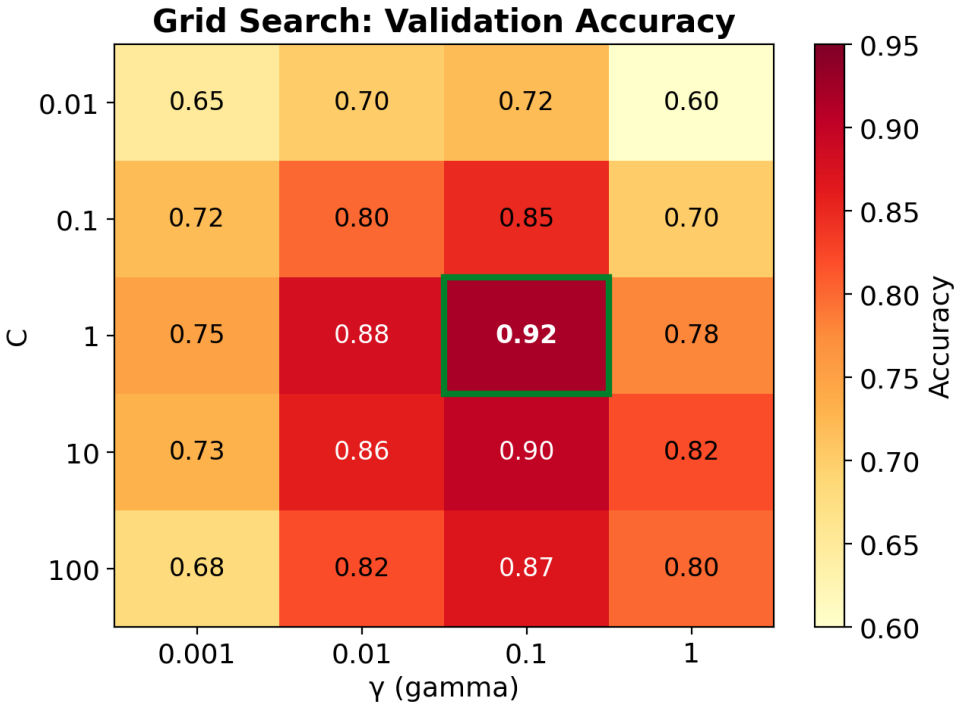
Hyperparameters: set before training

- SVM: C, γ
- Decision Tree: max depth, min samples
- Random Forest: number of trees, max features
- Learning rate η

We need a principled way to choose hyperparameters.

Grid Search

Define a grid of values to try:



For each combination:

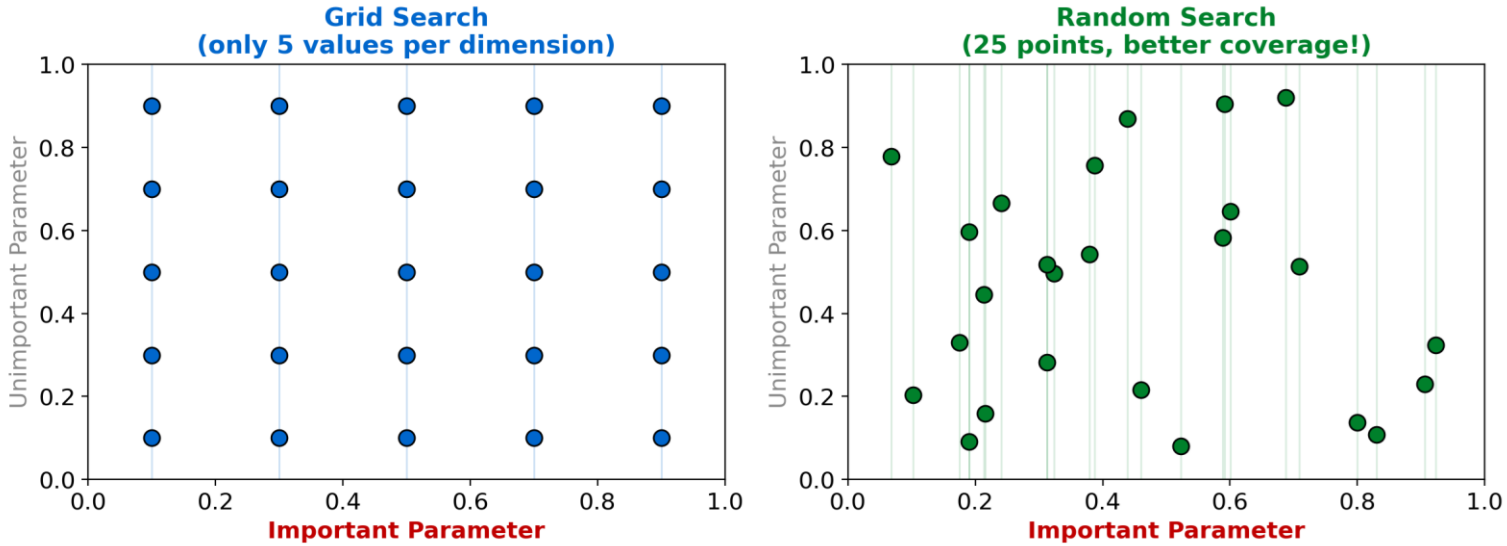
- Run K-fold CV, record average score
- Pick the combination with best CV score

Cost: |grid| × K training runs

- 5 × 4 × 5-fold = 100 training runs!

Random Search

- **Instead of trying all combinations, sample randomly.**
- **Surprisingly effective!**
- Often finds good hyperparameters faster than grid search.
- **Why?**
  - Not all hyperparameters are equally important.
  - Grid search wastes runs on unimportant dimensions.
  - Random search covers the important dimensions better.



🧠 Practical Tips for Tuning

1. Start coarse, then refine

First: $C \in \{0.01, 1, 100\}$. Then zoom in.

2. Use log scale for C, γ, learning rate

Try 10^{-3}, 10^{-2}, 10^{-1}, 1, 10

3. Focus on the important ones first

Some parameters matter more than others

4. Use scikit-learn: GridSearchCV, RandomizedSearchCV

5. Modern: Bayesian optimization (Optuna, Hyperopt)

Smarter than random — learns from previous trials

The Danger of Overfitting to Validation

- **Too many hyperparameter combinations?**

→ You might overfit to the validation set!

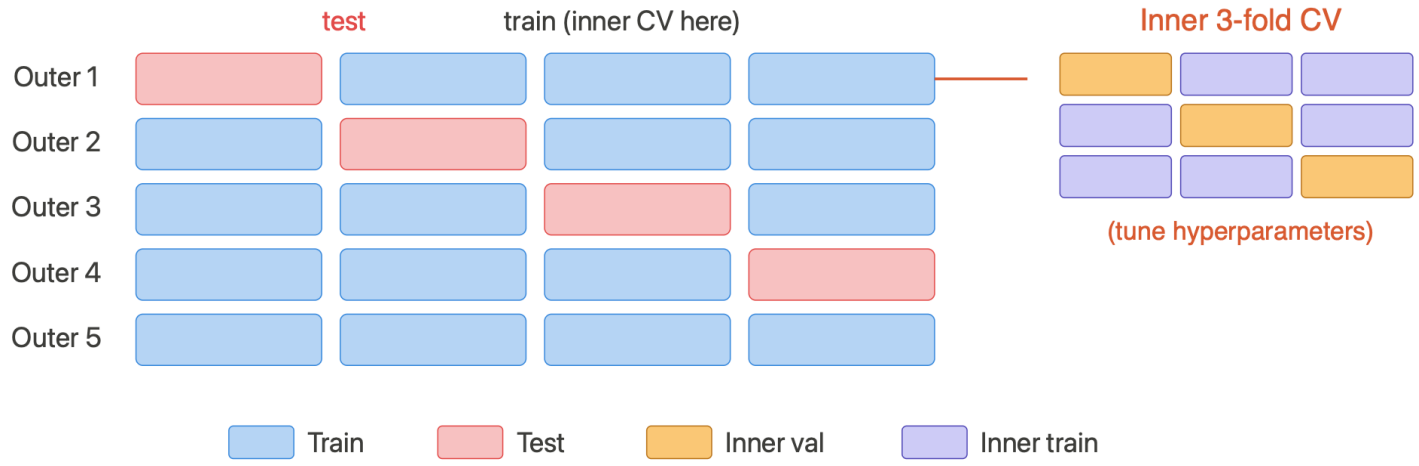
- Each time you check validation score, you leak information.
- With enough tries, even random noise can be exploited.

- **Solution:**
  - Keep test set sacred — never peek
  - Report only the final test score
  - Use nested cross-validation for rigorous estimates

Nested Cross-Validation

Gold standard for small datasets:

- Outer loop (5-fold): split into train/test folds
  For each outer fold:
  - Inner loop (3-fold): tune hyperparameters on train part
  - Evaluate tuned model on outer test fold
- Result: unbiased estimate of tuned model performance
- Expensive: 5 × 3 × |grid| training runs
- But gives the most honest evaluation.



🧠 Complete Pipeline: Putting It All Together

1. Split: 80% development, 20% test (hold out)
2. On development set: stratified 5-fold CV

3. For each model (LR, SVM, RF, GBM):
   a. Grid/random search over hyperparameters
   b. Record best CV score

4. Select best model + hyperparameters
5. Retrain on full development set
6. Evaluate once on test set → report this!

This is the standard procedure in ML research.

Common Mistakes to Avoid

Data leakage:

- Using test data during training or feature engineering
- Fix: fit scaler on train, apply to test

Peeking at test set:

- Evaluating multiple times and reporting the best
- Fix: touch test set exactly once

Wrong metric:

- Using accuracy on imbalanced data
- Fix: use F1, AUC, or precision/recall

Not using stratified splits for imbalanced classes

Summary: The Evaluation Toolkit

5 Key Concepts:

1. Cross-Validation — robust estimation using K folds
2. Confusion Matrix — TP, TN, FP, FN for all error types
3. Precision / Recall / F1 — beyond accuracy
4. ROC / AUC — threshold-independent comparison
5. Hyperparameter Tuning — grid/random search + CV

Always match the metric to the application.

Module 3 Complete!

The full classical ML pipeline:

- L8: ML Paradigm (data, features, loss, risk)
- L9: Linear Models (regression + classification)
- L10: Optimization (GD, SGD)
- L13: Support Vector Machines
- L14: Decision Trees & Ensembles
- L15: Model Evaluation (this lecture)

You now have everything to build, train, and evaluate classical ML models.

Next Module: Deep Learning

Module 4 starts next week!

- L16: Perceptron & MLP
- L17: Backpropagation
- L18: Training Deep Networks

Key question:

- What if the features $\phi(x)$ are also learned from data?