Introduction to Data Science

- Cross-validation, Bootstrap, and Decision Trees

Mauricio Molina

Keio University, Faculty of Economics

Jun 25, 2025

Contents

1. Introduction
2. Cross-Validation
3. Bootstrap
4. Comparison
5. Decision Trees

Resampling Methods

- Techniques that repeatedly draw samples from training data
- Key purposes:
  - Model assessment: Evaluate performance
  - Model selection: Choose flexibility level
  - Parameter uncertainty: Quantify variability
- Two main approaches:
  1. **Cross-Validation**: Estimate test error
  2. **Bootstrap**: Quantify parameter variability
- Computationally intensive but feasible with modern computing

Test Error vs. Training Error

- **Training error**: Error on same data used for training
- **Test error**: Error on new, unseen data
- Key problem: Training error $\ll$ test error (especially with flexible models)
- Challenge: Test data often unavailable
- Solution: Estimate test error using training data

Validation Set Approach

Procedure:

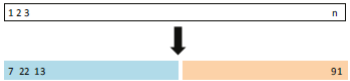
1. Randomly split data into training/validation sets
2. Fit model on training set
3. Calculate error on validation set

Pros:

- Simple to implement
- Computationally cheap

Cons:

- High variability (depends on split)
- Overestimates test error (uses less data)



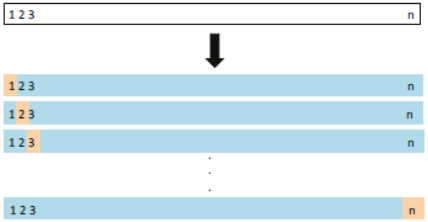
Leave-One-Out Cross-Validation (LOOCV)

Procedure:

1. For each observation i:
   - Train on all data except i
   - Predict for i
   - Compute error $e_i = (y_i - \hat{y}_i)^2$
2. LOOCV estimate: $\text{CV}_{(n)} = \frac{1}{n} \sum e_i$

Advantages:

- Less bias (uses $n - 1$ samples)
- No randomness in estimates



A set of n data points is repeatedly split into a training set (shown in blue) containing all but one observation, and a validation set that contains only that observation (shown in beige). The test error is then estimated by averaging the n resulting MSEs. The first training set contains all but observation 1, the second training set contains all but observation 2, and so forth.

k-Fold Cross-Validation

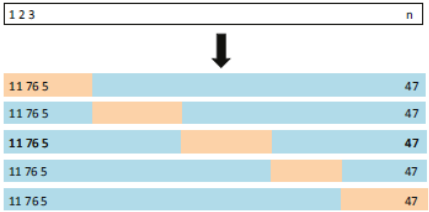
Procedure:

1. Randomly split data into k folds
2. For each fold i:
   - Train on other $k-1$ folds
   - Validate on fold i
3. CV estimate: $\text{CV}_{(k)} = \frac{1}{k}\sum \text{MSE}_i$

Typical choices: $k = 5$ or $k = 10$

Advantages:

- Lower variance than LOOCV
- Balanced bias-variance tradeoff



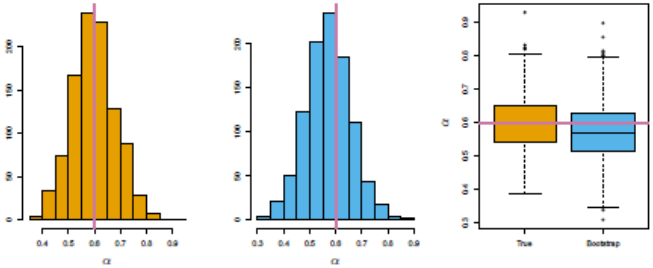
A schematic display of 5-fold CV. A set of n observations is randomly split into five non-overlapping groups. Each of these fifths acts as a validation set (shown in beige), and the remainder as a training set (shown in blue). The test error is estimated by averaging the five resulting MSE estimates.

Bias-Variance Trade-off

- **Validation set**: High bias (small training set)
- **LOOCV**: Low bias but high variance (correlated predictions)
- **k-Fold**: Balanced approach ($k = 5$ or 10 optimal)

The Bootstrap Concept

- Powerful method to quantify uncertainty
- Key idea: Resample with replacement from original data
- Creates "new" datasets without collecting new data
- Measures variability of estimates
- Formula: $\widehat{SE}(\hat{\alpha}) = \sqrt{\frac{1}{B-1}\sum_{r=1}^{B}(\hat{\alpha}^{*r} - \bar{\alpha}^{*})^2}$



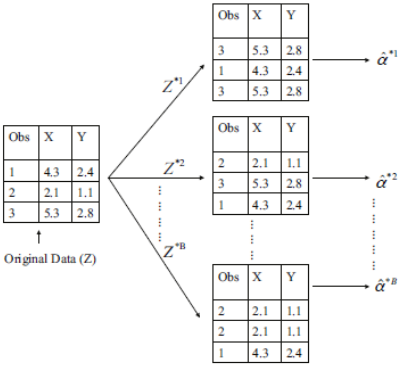
Bootstrap Example: Investment Allocation

- Problem: Allocate funds between assets X and Y
- Minimize risk: $\min_\alpha \text{Var}(\alpha X + (1-\alpha)Y)$
- Optimal: $\alpha = \frac{\sigma_Y^2 - \sigma_{XY}}{\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY}}$
- Bootstrap approach:
  1. Estimate $\hat{\sigma}_X^2, \hat{\sigma}_Y^2, \hat{\sigma}_{XY}$ from data
  2. Generate bootstrap samples
  3. Compute $\hat{\alpha}^{*b}$ for each sample
  4. Estimate $SE(\hat{\alpha})$ from bootstrap distribution

Bootstrap Procedure

Steps:

1. Start with dataset of size n
2. Generate bootstrap sample Z^{*b} by sampling n observations *with replacement*
3. Compute estimate $\hat{\alpha}^{*b}$ for each sample
4. Repeat B times ($B \geq 1000$)
5. Calculate standard error of bootstrap estimates



Bootstrap samples ($n = 3$)

Cross-Validation vs. Bootstrap

Characteristic	**Cross-Validation**	**Bootstrap**
Primary purpose	Estimate test error	Quantify uncertainty
Data splitting	Partitioned	Resampled
Repeats	k or n times	B times ($B \gg n$)
Error metric	MSE/Misclassification	Parameter variability
Key advantage	Model selection	General applicability
Computational cost	Moderate	High ($B \geq 1000$)

Tree Models

Tree models, also called *Classification and Regression Trees* (**CART**), decision trees, or just trees, are an effective and popular classification (and regression) method. Tree models, and their more powerful descendants random forests and boosted trees, form the basis for the most widely used and powerful predictive modeling tools in data science for regression and classification. A tree model is a set of "*if–then–else*" rules that are easy to understand and to implement.

In contrast to linear and logistic regression, trees have the ability to discover hidden patterns corresponding to complex interactions in the data. However, unlike KNN or naive Bayes, simple tree models can be expressed in terms of predictor relationships that are easily interpretable.

Key Terms for Trees

- **Recursive partitioning:** Repeatedly dividing and subdividing the data with the goal of making the outcomes in each final subdivision as homogeneous as possible.
- **Split value:** A predictor value that divides the records into those where that predictor is less than the split value, and those where it is greater.
- **Node:** In the decision tree, or in the set of corresponding branching rules, a node is the graphical or rule representation of a split value.
- **Leaf:** The end of a set of if–then rules, or branches of a tree—the rules that bring you to that leaf provide one of the classification rules for any record in a tree.
- **Loss:** The number of misclassifications at a stage in the splitting process; the more losses, the more impurity.
- **Impurity:** The extent to which a mix of classes is found in a subpartition of the data (the more mixed, the more impure).
  Synonym: Heterogeneity *Antonyms:* Homogeneity, Purity
- **Pruning:** The process of taking a fully grown tree and progressively cutting its branches back to reduce overfitting.

Decision Tree Illustration

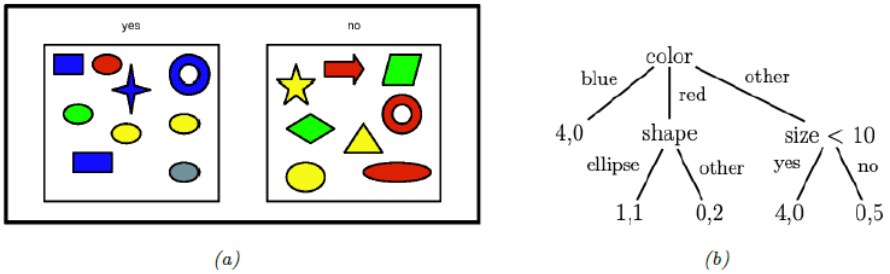


Figure: (a) A set of shapes with corresponding binary labels. The features are: color (values "blue", "red", "other"), shape (values "ellipse", "other"), and size (real-valued). (b) A hypothetical classification tree fitted to this data. A leaf labeled as (n_1, n_0) means that there are n_1 positive examples that fall into this partition, and n_0 negative examples.

Source: *Probabilistic Machine Learning, An Introduction*

Regression Trees

Tree-based methods for prediction are called regression trees (for continuous outcomes) or classification trees (for categorical outcomes). Regression trees partition the predictor space into distinct regions and make constant predictions within each region. They are especially useful when:

- There are complex interactions among predictors.
- We want a model that is easy to interpret via "if–then" rules.
- We prefer a nonparametric approach that does not assume linearity.

Predicting Baseball Players' Salaries

- **Data:** Hitters data set (baseball players' statistics).
- **Goal:** Predict log(Salary) (measured in thousands of dollars) using:
  - *Years* (number of years in the major leagues)
  - *Hits* (number of hits in the previous year)
- **Preprocessing:**
  1. Remove observations with missing Salary.
  2. Apply log-transform to Salary so the distribution is approximately bell-shaped.
  3. Interpret predicted values by back-transforming: if $\hat{y} = \log(\text{Salary})$, then

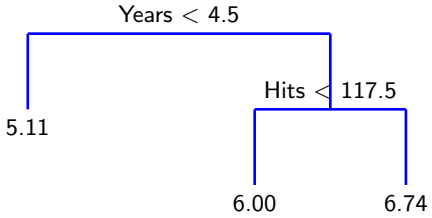
$$\text{Salary} = e^{\hat{y}} \times 1000.$$

Regression Tree for Hitters Data

At a given internal node, the label (of the form $X_j < t_k$) indicates the left-hand branch emanating from that split, and the right-hand branch corresponds to $X_j \geq t_k$.

The split at the top of the tree results in two large branches. The left-hand branch corresponds to $\text{Years} < 4.5$, and the right-hand branch corresponds to $\text{Years} \geq 4.5$.

The tree has two internal nodes and three terminal nodes, or leaves. The number in each leaf is the mean of the response for the observations that fall there.



Tree Terminology

Terminal Node (Leaf): A final region R_m of the predictor space. In this example:

$$R_1 = \{X \mid \texttt{Years} < 4.5\},$$
$$R_2 = \{X \mid \texttt{Years} \geq 4.5,\ \texttt{Hits} < 117.5\},$$
$$R_3 = \{X \mid \texttt{Years} \geq 4.5,\ \texttt{Hits} \geq 117.5\}.$$

Internal Node: A splitting rule (e.g. `Years < 4.5` or `Hits < 117.5`) that divides a parent node into two child nodes.

Branch: The connection between parent and child nodes, representing one side of a split.

Each leaf's predicted value is the mean of log(Salary) within that region:

$$\hat{y}_{R_1} = 5.107 \quad (\Rightarrow \$165{,}174),$$
$$\hat{y}_{R_2} = 5.999 \quad (\Rightarrow \$402{,}834),$$
$$\hat{y}_{R_3} = 6.740 \quad (\Rightarrow \$845{,}346).$$

Regions Defined by the Tree

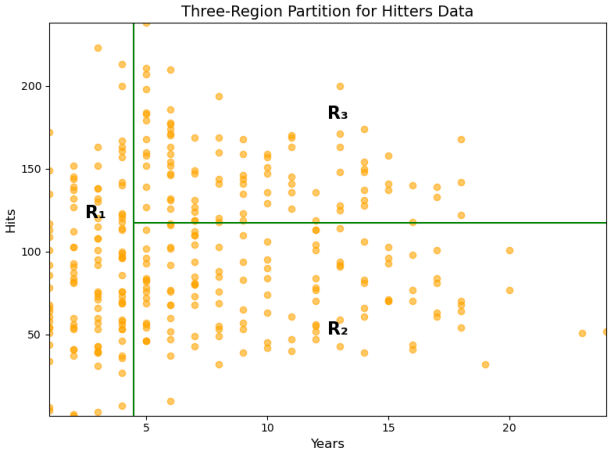


Figure: Partition of the predictor space (Years vs. Hits) into three regions R_1, R_2, R_3.

Interpretation of the Regression Tree

- **First Split:** *Years* is the most important predictor. Players with fewer than 4.5 years in the league (region R_1) earn lower salaries on average.
- **Second Split (for Years ≥ 4.5):** Among more experienced players, *Hits* further stratifies salaries:
  - *Hits* < 117.5 (region R_2): moderate salary, approximately \$402,834.
  - *Hits* ≥ 117.5 (region R_3): higher salary, approximately \$845,346.
- **Advantages:**
  - Easy-to-interpret "if–then" rules.
  - Captures nonlinear effects and interactions automatically.
  - Graphical representation facilitates communication of the model.
- **Note:** A single regression tree may be too simple and prone to overfitting or high variance; ensemble methods like bagging or random forests can improve performance.

Prediction via Stratification of the Feature Space

- The process of building a regression tree involves two main steps:
  1. Divide the predictor space (all possible values of $X_1, X_2, \ldots, X_p$) into J distinct, non-overlapping regions $R_1, R_2, \ldots, R_J$.
  2. For each region R_j, predict the response as the mean of the training-observations in R_j.
- Example: If splitting yields two regions R_1 and R_2 with means 10 and 20, then

$$\hat{f}(x) = \begin{cases} 10, & x \in R_1, \\ 20, & x \in R_2. \end{cases}$$

Choosing the Regions to Minimize RSS

- We restrict regions $R_1, \ldots, R_J$ to be axis-aligned rectangles ("boxes") in $\mathbb{R}^p$.
- Goal: Find a partition into J boxes that minimizes the residual sum of squares (RSS):

$$\text{RSS} = \sum_{j=1}^{J} \sum_{i\,:\,x_i \in R_j} \left(y_i - \hat{y}_{R_j}\right)^2, \quad \hat{y}_{R_j} = \text{mean}\{y_i : x_i \in R_j\}.$$

- Searching over *all* possible J-box partitions is computationally infeasible.
- Instead, use a top-down, greedy approach called *recursive binary splitting*.

Recursive Binary Splitting: First Split

- At each step, consider all predictors X_j ($j = 1, \ldots, p$) and all possible cutpoints s.
- Define two candidate regions for predictor j and cutpoint s:

$$R_1(j, s) = \{X : X_j < s\}, \quad R_2(j, s) = \{X : X_j \geq s\}.$$

- For each (j, s), compute

$$\mathrm{RSS}(j, s) = \sum_{i: x_i \in R_1(j,s)} \left(y_i - \hat{y}_{R_1}\right)^2 + \sum_{i: x_i \in R_2(j,s)} \left(y_i - \hat{y}_{R_2}\right)^2,$$

  where $\hat{y}_{R_1}$ and $\hat{y}_{R_2}$ are the means of y_i in each half-space.
- Choose the pair (j^*, s^*) that minimizes $\mathrm{RSS}(j, s)$.

Recursive Binary Splitting: Subsequent Steps

- After the first split, we have two regions. Next, search within each existing region for the best further split:

$$\mathrm{RSS}(j, s \,|\, R_k) = \sum_{i:x_i \in R_{k,1}(j,s)} (y_i - \hat{y}_{R_{k,1}})^2 + \sum_{i:x_i \in R_{k,2}(j,s)} (y_i - \hat{y}_{R_{k,2}})^2,$$

  where $R_{k,1}(j, s) = \{x \in R_k : x_j < s\}$, and $R_{k,2}(j, s) = \{x \in R_k : x_j \geq s\}$.
- At each iteration:
  1. Consider all existing regions.
  2. For each region, scan all (j, s) pairs to find the split that yields the largest RSS reduction within that region.
  3. Choose the single best split across all regions and add two new branches to the tree.
- Continue until a stopping criterion is met (e.g., minimum node size, maximum tree depth, or no further RSS improvement).

Tree Pruning: Motivation

- A tree grown via recursive splitting often overfits the training data, leading to high variance and poor test performance.
- A smaller tree (fewer terminal nodes) can reduce variance at the cost of a bit of bias, improving generalization.
- Simple early-stopping (requiring each split to decrease RSS by a large threshold) may discard splits that later enable better splits downstream.
- Instead, build a large tree T_0 first and then prune it back to find an optimal subtree.

Cost Complexity Pruning

- Define a sequence of subtrees of T_0 indexed by a nonnegative tuning parameter α.
- For any subtree T with $|T|$ terminal nodes $\{R_m\}_{m=1}^{|T|}$, define its cost-complexity measure as:

$$C_\alpha(T) = \sum_{m=1}^{|T|} \sum_{i:x_i \in R_m} \left(y_i - \hat{y}_{R_m}\right)^2 + \alpha\,|T|,$$

  where $\hat{y}_{R_m}$ is the mean response within region R_m.
- When $\alpha = 0$, the best subtree is T_0 itself (no penalty on complexity). As α increases, subtrees with fewer leaves become preferred.
- The sequence of optimal subtrees (for $\alpha = 0, \alpha_1, \alpha_2, \ldots$) can be obtained by *weakest link pruning*, which prunes branches in a nested fashion.

Selecting α via Cross-Validation

1. Grow a large tree T_0 on the training data, stopping only when each terminal node has fewer than a minimum number of observations.
2. Apply cost complexity pruning to T_0 to obtain a sequence of best subtrees $\{T(\alpha)\}$ as α varies.
3. Use K-fold cross-validation on the training set to estimate the error for each $T(\alpha)$:
   - Split training data into K folds.
   - For each fold $k = 1, \ldots, K$:
     1. Grow and prune a tree on the $K - 1$ other folds, yielding $\{T_{-k}(\alpha)\}$.
     2. Compute mean squared error (MSE) on the left-out kth fold for each α.
   - Average MSE across folds for each α, then choose α^* minimizing the cross-validated MSE.
4. Finally, prune the full-data tree T_0 to the subtree $T(\alpha^*)$.

Algorithm 8.1: Building a Regression Tree

1. **Grow a large tree** on the training data via recursive binary splitting:
   - Continue splitting until each terminal node has fewer than a pre-specified minimum number of observations.
2. **Cost complexity pruning** on the large tree T_0 to obtain subtrees $\{T(\alpha)\}$:
   - Compute $C_\alpha(T)$ for each candidate subtree.
   - Prune "weakest links" iteratively to create a nested sequence of subtrees.
3. **Choose** α via K-fold cross-validation on the training set (Steps 1 & 2 inside each fold).
4. **Return** the subtree $T(\alpha^*)$ corresponding to the chosen α^*, and use it for prediction.

Pruning Example on Hitters Data

- Fit a large regression tree (e.g. using nine predictors) on half the Hitters data (132 observations).
- Vary α to obtain subtrees with different numbers of terminal nodes.
- Use six-fold cross-validation (since $132 \div 6 = 22$ exactly) to estimate CV error vs. α.
- Compare CV error curve (green) with test error (orange) as a function of tree size (number of leaves).
- The CV error is minimized at a three-node subtree (pruned tree shown in Figure 8.1).
- The pruned three-node tree often generalizes best, balancing bias–variance.

Assignment: Validation Set Approach

Answer all the following questions in the Jupyter Notebook format. Show your Python code (when requested) and a short explanation for every result. Upload the completed `.ipynb` to K-LMS by next Tuesday at midnight.

Q1: Using the `Auto` data set, apply the validation-set approach to compare polynomial models for predicting `mpg` from `horsepower`. Do the following:

1. Split `Auto` into a training set and a validation set of size 196 each, by calling `train_test_split(Auto, test_size=196, random_state=0)`.
2. On the training set, fit three models:
   - Linear: $\text{mpg} \sim \text{horsepower}$
   - Quadratic: $\text{mpg} \sim \text{horsepower} + \text{horsepower}^2$
   - Cubic: $\text{mpg} \sim \text{horsepower} + \text{horsepower}^2 + \text{horsepower}^3$

   For each model, compute the validation-set MSE.
3. Repeat steps 1–2 with `random_state` $= 3$. Report the new MSEs and identify which degree now gives the lowest error.
4. (Optional) Briefly explain why the chosen "best" degree may differ between the two random splits.

Q2: Using the `Boston` data set and `medv` as the response, carry out the following:

1. Split the data into training and test sets with `test_size=0.3` and `random_state=0`.
2. On the training set, fit a regression tree with `DecisionTreeRegressor(max_depth=3)`.
   - Compute and report the training and test MSE.
   - Plot the tree and interpret the top-level split and one terminal-node prediction.
3. Prune the tree via cost-complexity pruning:
   - Obtain the pruning path with `cost_complexity_pruning_path`.
   - Use 5-fold CV (`GridSearchCV`) to select the optimal `ccp_alpha`. Report its value and the number of leaves in the pruned tree.
4. Refit the tree with the chosen `ccp_alpha` on the full training data, and report the test MSE and RMSE.
5. (Optional) Plot the pruned tree and briefly describe how its splits relate to the most important predictors.