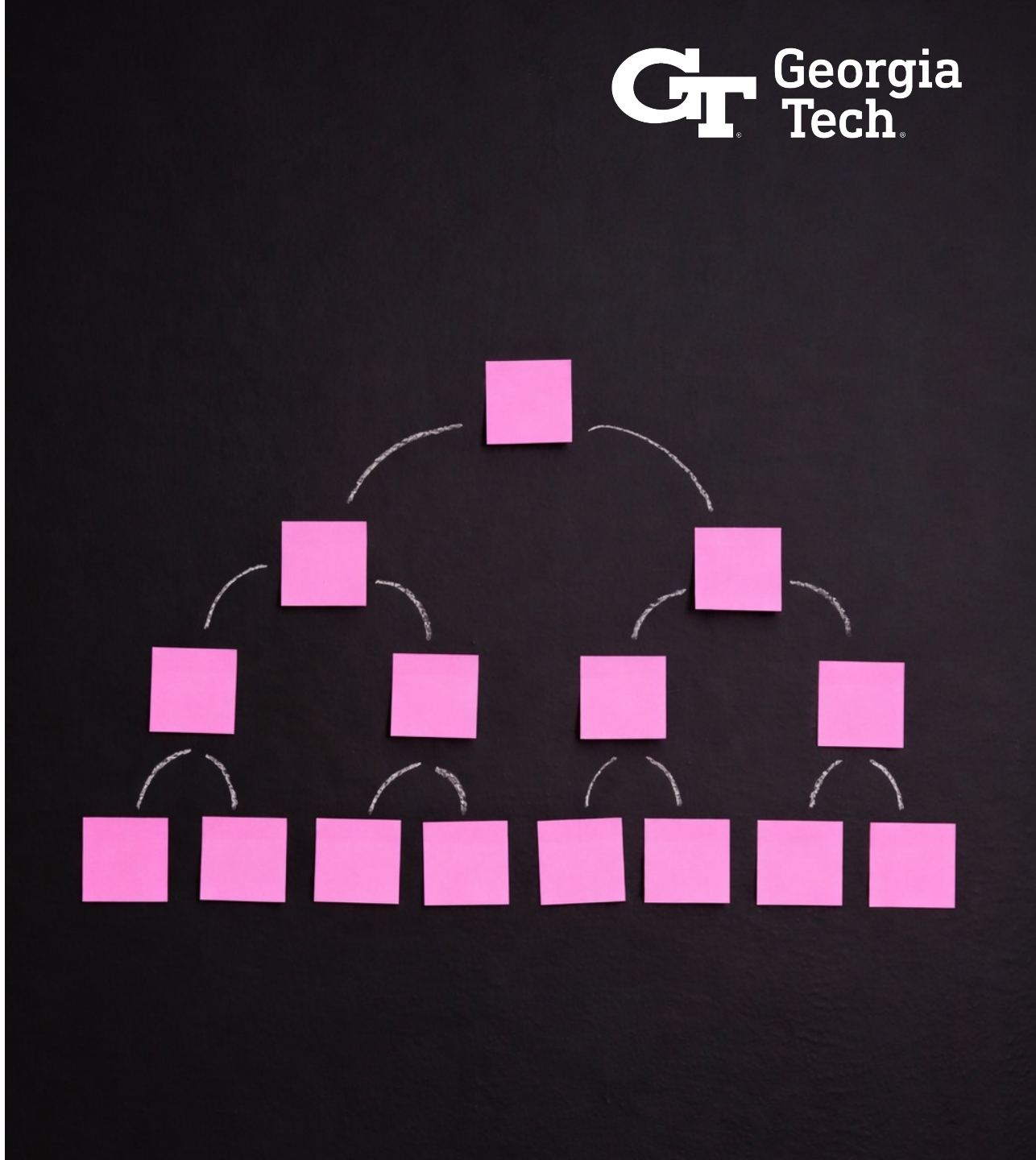
Decision Trees & Random Forests

Mohsen Moghaddam, Ph.D.

Gary C. Butler Family Associate Professor
H. Milton Stewart School of Industrial and Systems Engineering
George W. Woodruff School of Mechanical Engineering
Georgia Institute of Technology

SAIL
Symbiotic & Augmented Intelligence Laboratory

Spring 2026



Learning Outcomes

- Explain how decision trees model data using recursive if–then rules for classification and regression
- Describe how CART constructs trees using greedy splits and why trees are prone to overfitting
- Apply pruning concepts to control tree complexity and improve generalization
- Explain how bagging and random feature selection reduce variance in Random Forests
- Analyze the role of tree correlation and averaging in ensemble performance
- Interpret out-of-bag (OOB) error as an efficient alternative to cross-validation
- Evaluate feature importance using permutation and impurity-based measures
- Compare Random Forests and Boosted Trees in terms of bias–variance tradeoff and use cases

Decision Trees: Intuition & Mechanics

Example: Predicting House Prices

Goal: Use simple dataset to estimate how much a new house might sell for

→ **Features:** Size (square footage); Number of rooms; Location; Year built

Why this is interesting

We want a model that mimics **how a human real-estate expert might reason**, using clear "if–then" style rules

Why decision trees?

- Interpretability (if–then rules)
- Handle nonlinearities & interactions
- Minimal preprocessing

Finding Simple Patterns

We start by looking for strong, useful, and easy-to-understand trends:

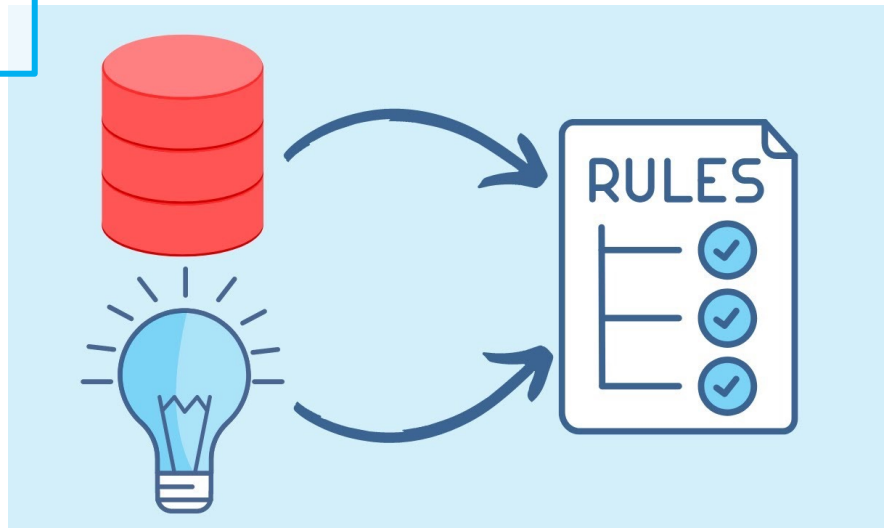
"Houses larger than 2,000 sq ft tend to sell for higher prices"

Then we can refine it further:

Each new rule makes our prediction a little sharper

"Among larger houses, those in better locations sell for even higher prices"

"Among smaller houses, newer ones cost more than older ones"

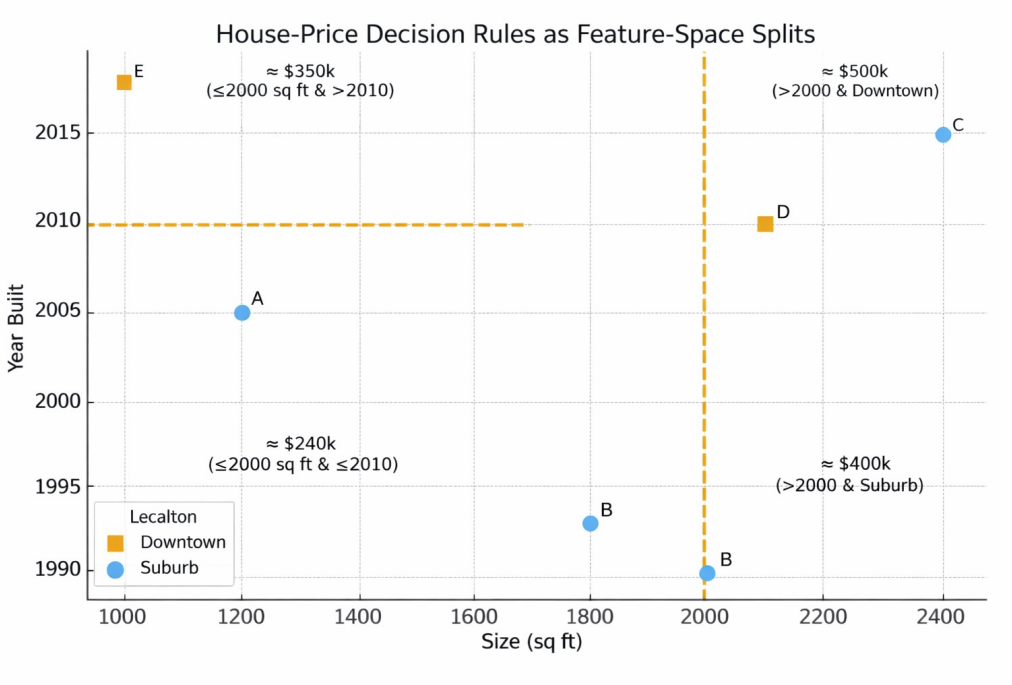


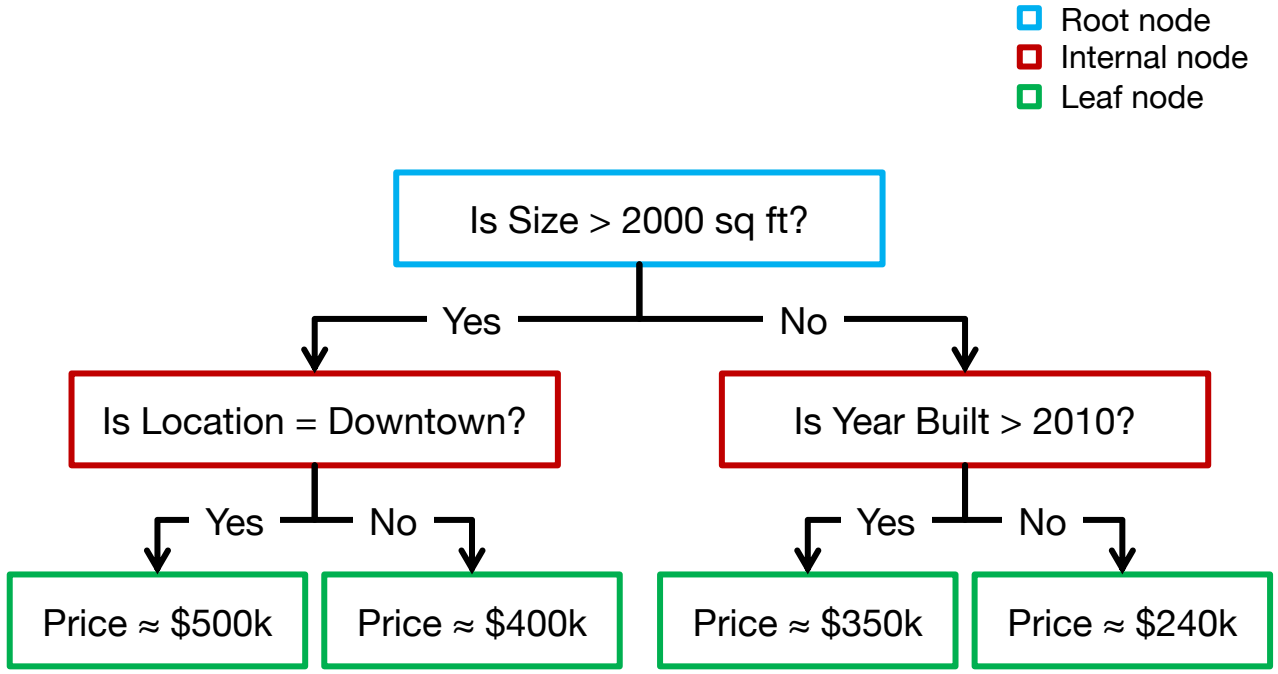
A tree learns these rules by choosing splits that improve prediction the most

Tree-Structured Decision Rule

A decision tree represents a sequence of **nested "if–then" rules**

- Each **Internal node** asks a question about one feature
- Each **leaf node** outputs a final prediction





Balance Details and Simplicity

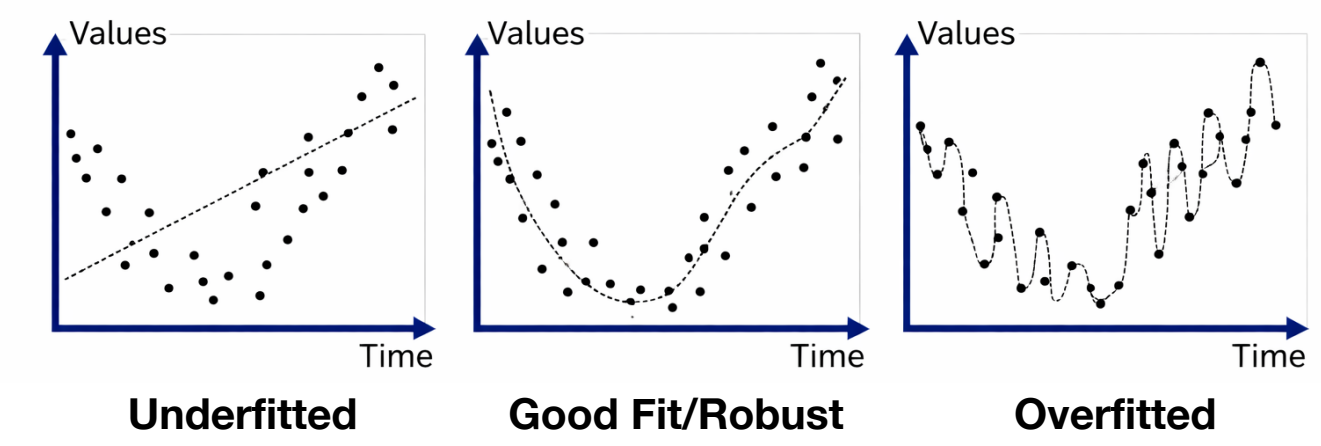
If we keep adding rules, we'll perfectly fit past data but make bad predictions for new houses

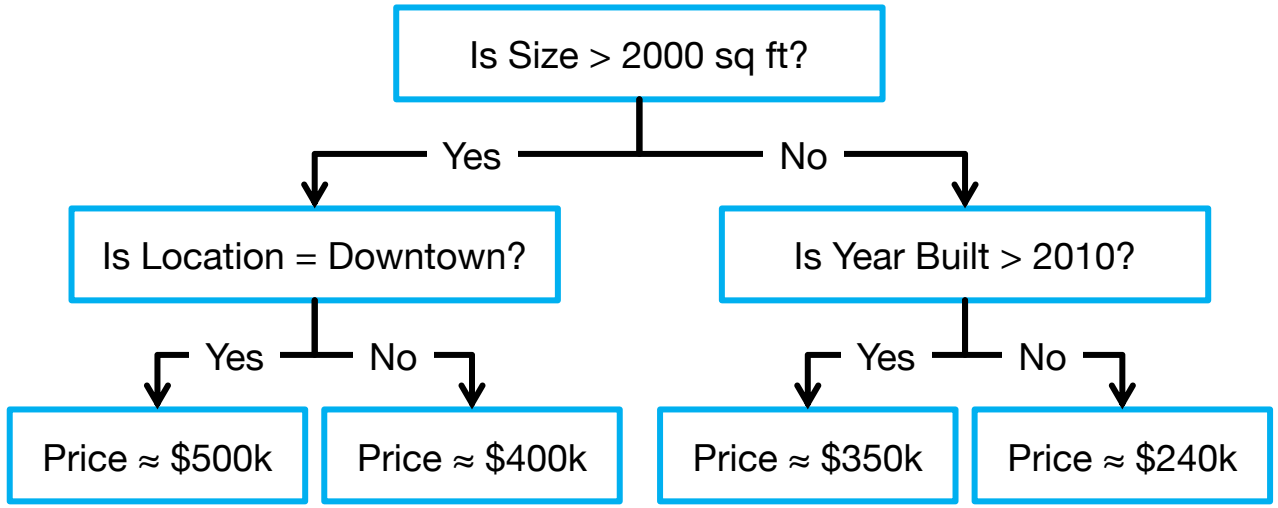
→ **overfitting**

- **Too deep:** low bias, high variance (overfit)
- **Too shallow:** high bias, low variance (underfit)

Simplify the rules, by removing unnecessary details, to capture only the stable, general patterns

→ **better generalization**





Designing a Decision Tree Algorithm

Design Considerations

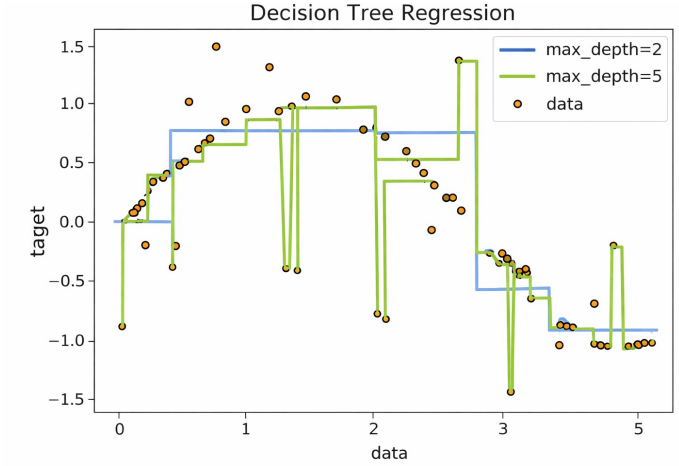
- Learn interpretable, rule-based models directly from data
- Automatically identify the most important factors influencing price
- **Balance fit and simplicity:** too few rules miss structure (underfitting), too many overfit noise (overfitting)
- Extend naturally to **ensemble methods** such as random forests, which average many rule sets to improve stability and performance
- Remain robust to noisy data and small perturbations in the training set

Single trees are unstable → ensembles average many trees to reduce variance

What Are Decision Trees?

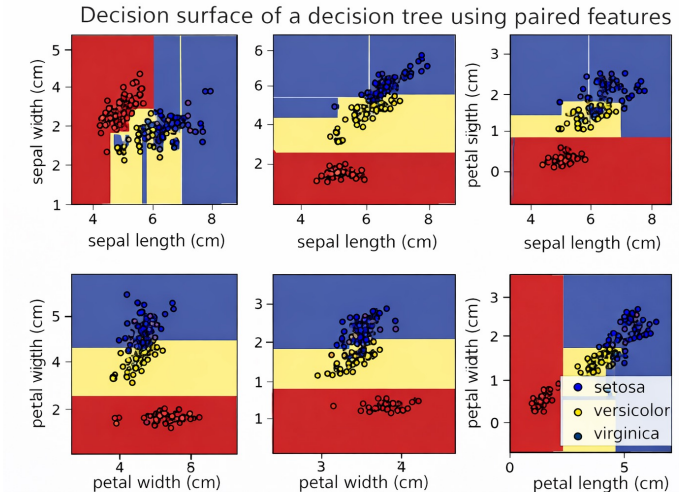
Decision Tree

Non-parametric supervised learning method for **classification** and **regression** that learns simple decision rules from data



Classification and Regression Decision Tree (CART)

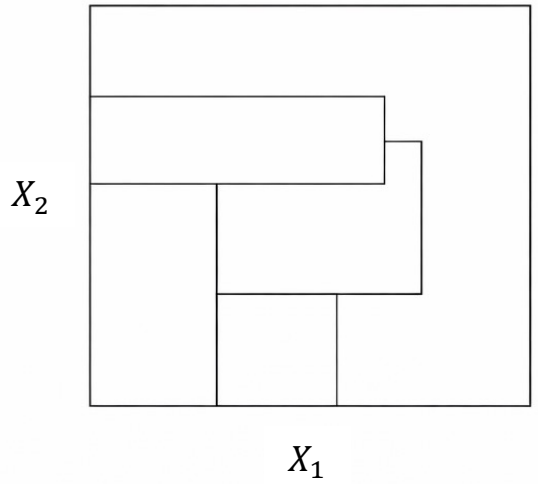
- A single framework handling both regression and classification by recursively learning simple rules
- Splits are chosen to **minimize impurity**, which makes it fast and interpretable but not globally optimal
- Can be sensitive to noise and prone to overfitting
- Grows the tree **greedily** (best split now ≠ best overall tree)

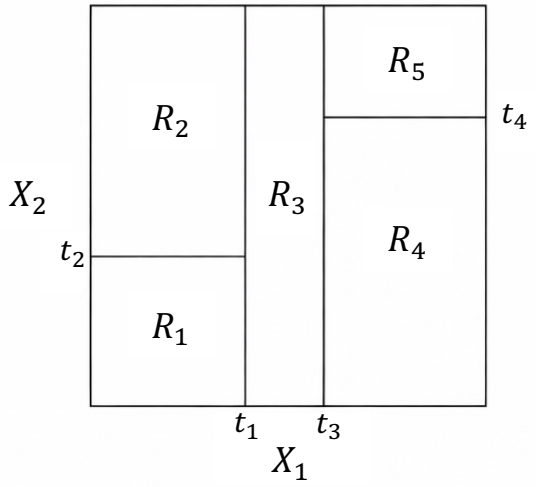


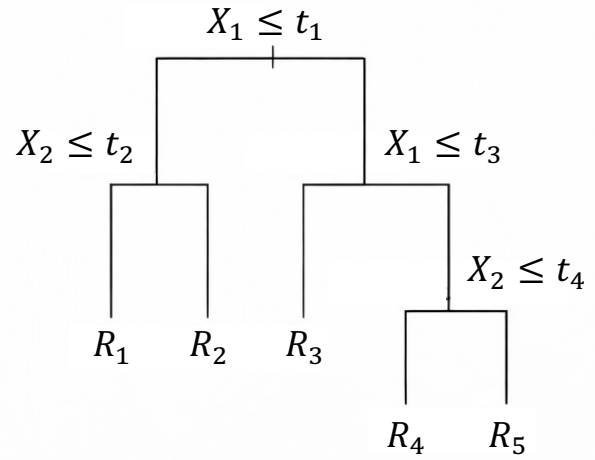
Why is a greedy split appealing computationally?

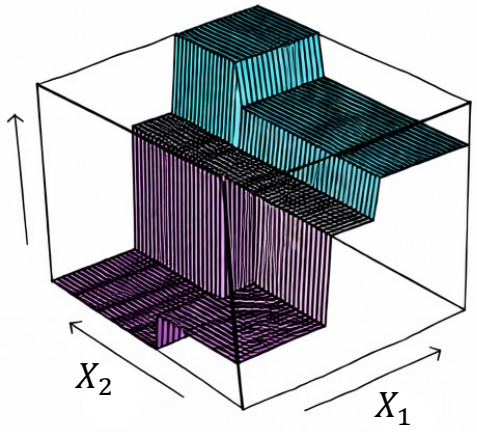
Classification and Regression Tree (CART)

- Partition the feature space into a set of **rectangles**
- Decision rule can be represented by a tree (binary tree most common)
- **Regression tree:** Fit a simple model (e.g., a constant) for each rectangle
- **Classification tree:** Majority vote for samples in the rectangle









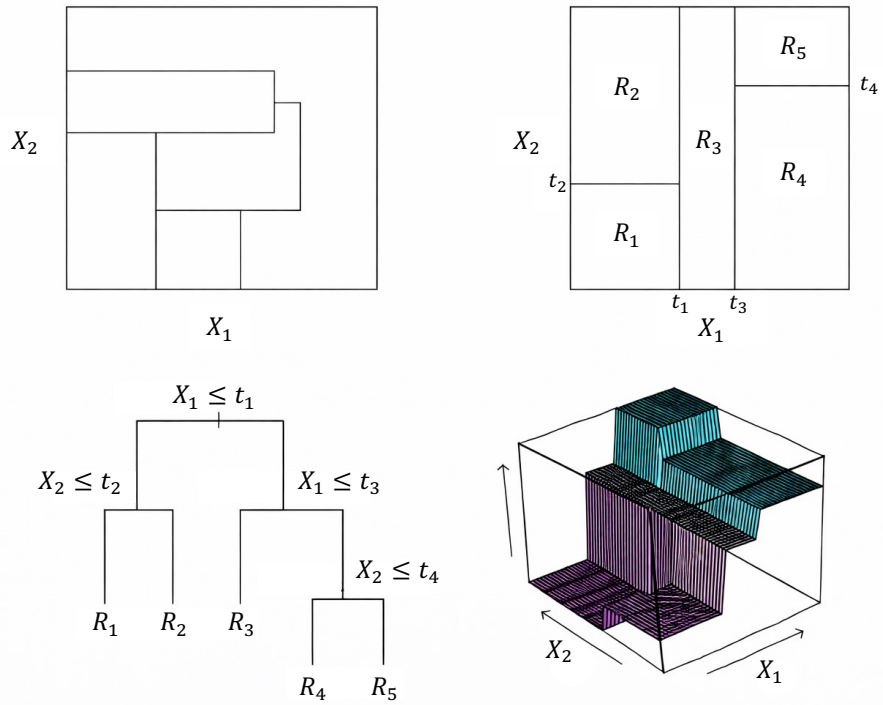
Why do we get rectangular shapes?

Regression Trees

Regression Tree: Simple Example

Output one of J constant values (one per region), depending on where the input falls

- First, split the space into two regions and model the response by the mean of Y in each region
- Choose the variable and split-point to achieve the best fit
- Then one or both regions are split into two more regions
- Continued, until some stopping rule is applied



$$\hat{f}(X) = \sum_{j=1}^{5} c_j \, \mathbb{I}\{(X_1, X_2) \in R_j\}$$

Mean of y for points in R_j

Region j

Leaf prediction =
Average of training y-values in that region

Boba Shop Example

Goal: Predict wait time (minutes) or classify as **short**, **medium**, **long**

Features

- hour_of_day (encode with sin/cos for cyclic time)
- day_of_week
- temp_F, rain_indicator
- nearby_class_ends_in_10min (0/1)—from campus timetable
- promo_active (0/1)
- mobile_orders_last_10min (count)

Trees handle mixed types naturally (e.g., categorical, numeric, binary)

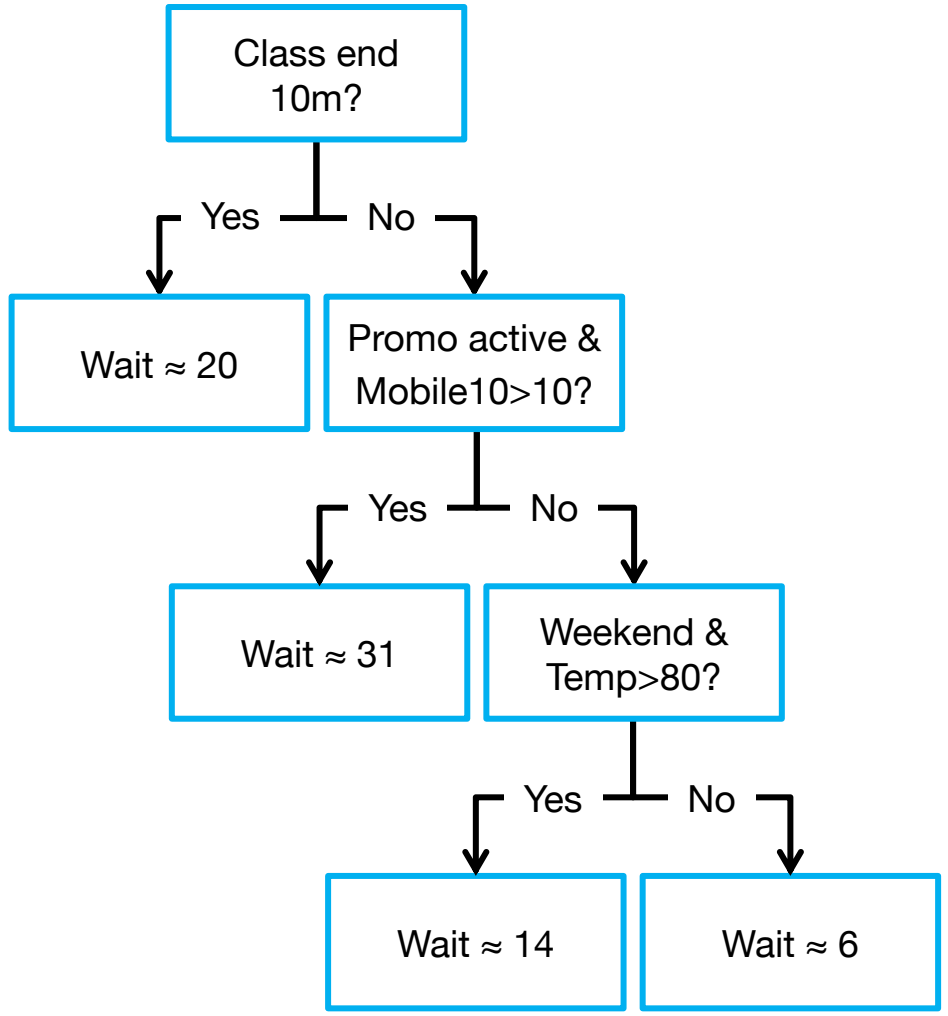
OBS	HOUR	DAY	TEMP	RAIN	CLASS_END	PROMO	MOBILE10	WAIT_MIN
1	11	Tue	78	0	1	0	7	18
2	15	Tue	83	0	0	1	12	26
3	12	Thu	65	1	1	0	5	22
4	19	Sat	88	0	0	1	20	35
5	10	Sun	72	0	0	0	2	6

Boba Shop Decision Rule

```
Input: class_end_10m, promo_active, mobile10,
day, temp_F

Output: wait_time

if class_end_10m = 1 then
    wait_time ≈ 18–22 min
else if promo_active = 1 and mobile10 > 10 then
    wait_time ≈ 26–35 min
else if day ∈ {Sat, Sun} and temp_F > 80 then
    wait_time ≈ 10–18 min
else
    wait_time ≈ 3–9 min // baseline short wait
```



Regression Tree: Fitting

Greedy approach: Constructing decision trees "top-down"

- Start with all training samples in a single region (the root node)
- At each step, choose the variable (greedy) and split point that best partition the data

Split on variable j at position s (two half-spaces):

$$R_1(j,s) = \{X | X_j \leq s\} \text{ and } R_2(j,s) = \{X | X_j > s\}$$

Solve for optimal splitting and "regression"

$$\min_{j,s} \left[\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2 \right]$$

Once (j,s) is decided, the optimal prediction is simply

$$\hat{c}_1 = \text{mean}(y_i | x_i \in R_1(j,s)),\ \hat{c}_2 = \text{mean}(y_i | x_i \in R_2(j,s))$$

Algorithm

Input: data $(x_i, y_i)_{i=1}^m$, features $X \in \mathbb{R}^p$

Output: best split (j^*, s^*), regions R_1, R_2, predictions $\hat{c}_1$, $\hat{c}_2$

for each variable $j \in \{1,2,\ldots,p\}$

    for each candidate cut-point s

        Define regions:

$$R_1(j,s) = \{X | X_j \leq s\}, R_2(j,s) = \{X | X_j > s\}$$

        Compute region predictions (means):

$$\hat{c}_1 = \text{mean}(y_i | x_i \in R_1(j,s)), \hat{c}_2 = \text{mean}(y_i | x_i \in R_2(j,s))$$

        Compute sum of squared errors:

$$\text{SSE}(j,s) = \Sigma_{i \in R1}(y_i - \hat{c}_1)^2 + \Sigma_{i \in R2}(y_2 - \hat{c}_2)^2$$

Choose $(j^*, s^*) = \arg\min_{j,s} \text{SSE}(j,s)$

Return: (j^*, s^*), R_1, R_2, $\hat{c}_1$, $\hat{c}_2$

Example: Predicting Exam Scores

Consider a small sample of six students who recently took an exam—for each student, two study-related features and final exam scores are recorded:

Student	X_1 = Study Hours/Day	X_2 = # Practice Tests	Y = Exam Score (100)
A	2	1	60
B	3	1	70
C	1	3	65
D	4	2	85
E	5	3	92
F	2	4	78

Goal: Use Study Hours and Practice Tests to predict Exam Score

Example: Predicting Exam Scores (cont.)

Step 1. Pick a variable

- We'll start with **Study Hours** (X_1)
- We'll test all possible cut-points (midpoints between sorted X_1 values):

$$s \in \{1.5, 2.5, 3.5, 4.5\}$$

Step 2. Pick a cut-point s and divide data

- Let's try $s = 3.5$
- **Left group:** R_1 students with $X_1 \leq 3.5 \rightarrow \mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{F}$
- **Right group:** R_2 students with $X_1 > 3.5 \rightarrow \mathrm{D}, \mathrm{E}$

Example: Predicting Exam Scores (cont.)

Step 3. Compute averages (predictions)

$$\hat{c}_1 = \text{mean}(\, y_i \mid i \in R_1\,) = \frac{60 + 70 + 65 + 78}{4} = 68.25 \quad \rightarrow \quad \text{Left group} \approx 68.3 \text{ points}$$

$$\hat{c}_2 = \text{mean}(\, y_i \mid i \in R_2\,) = \frac{85 + 92}{2} = 88.5 \quad \rightarrow \quad \text{Right group} \approx 88.5 \text{ points}$$

Step 4. Compute total squared error (SSE)

For R_1

$$(60 - 68.25)^2 + (70 - 68.25)^2 + (65 - 68.25)^2 + (78 - 68.25)^2 = 68.1 + 3.1 + 10.6 + 95.1 = 176.9$$

For R_2

$$(85 - 88.5)^2 + (92 - 88.5)^2 = 12.25 + 12.25 = 24.5$$

$$\text{Total SSE}(X_1, 3.5) = 176.9 + 24.5 = \mathbf{201.4}$$

Example: Predicting Exam Scores (cont.)

Step 5. Try all possible splits and pick the best

- We'll repeat Steps 2–4 for all cut-points on X_1 and X_2
- We'll compare all SSE values and choose the smallest

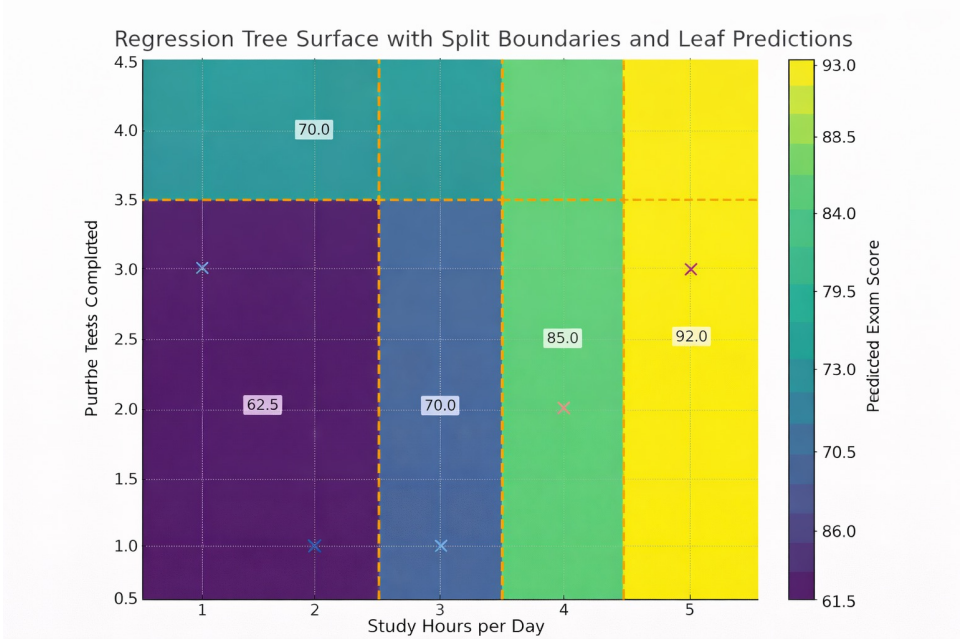
Variable	Cut s	SSE	Best?
Study Hours	1.5	417.0	
Study Hours	2.5	306.4	
Study Hours	**3.5**	**201.4**	✅ Best
Study Hours	4.5	280.0	
Practice Tests	1.5	344.8	
Practice Tests	2.5	234.5	
Practice Tests	3.5	220.6	

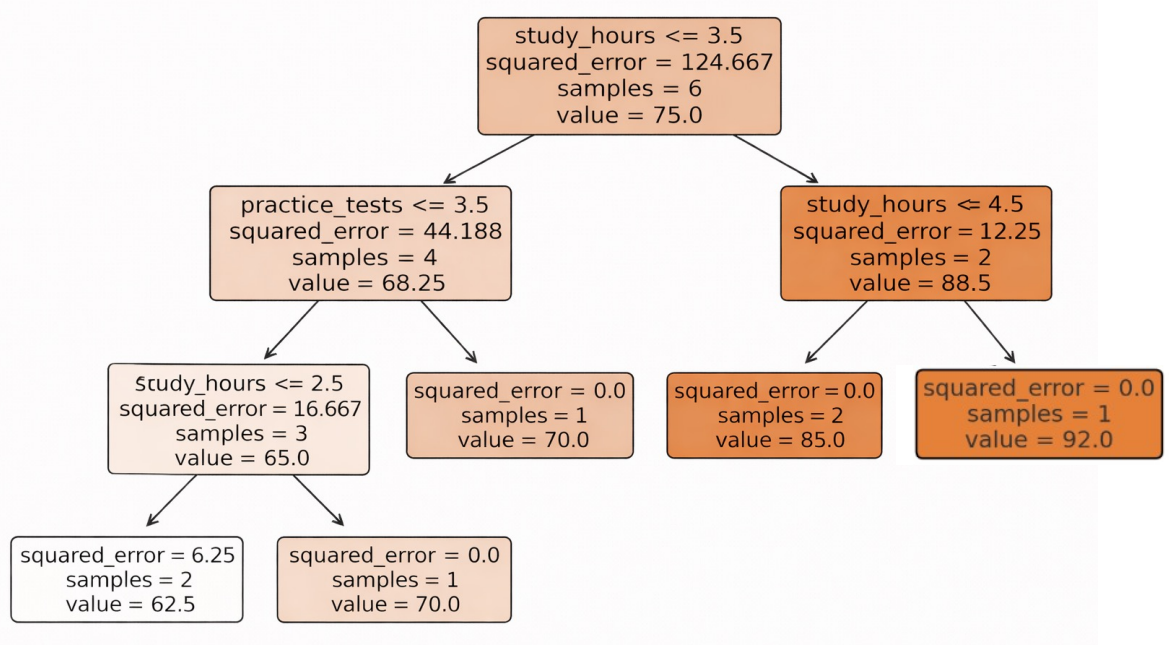
Regression Tree (Max Dept = 3)

The tree learns intuitive rules:

- If study_hours ≤ 2.5 and few practice tests → predict ≈ 62.5
- If study_hours ≤ 3.5 but many practice tests → predict ≈ 78
- If study_hours > 3.5 → split again by how far above 4.5 → 85 vs 92

Depth = complexity knob
(controls overfitting)





Tree Pruning

Data-Fit Complexity Tradeoff

Regularization parameter ($\alpha > 0$)

$$C_\alpha(S) = \sum_{j=1}^{|J|} \sum_{x_i \in R_j} \left(y_i - \hat{c}_j\right)^2 + \alpha |J|$$

Measure of data-fitting error for the j^{th} node

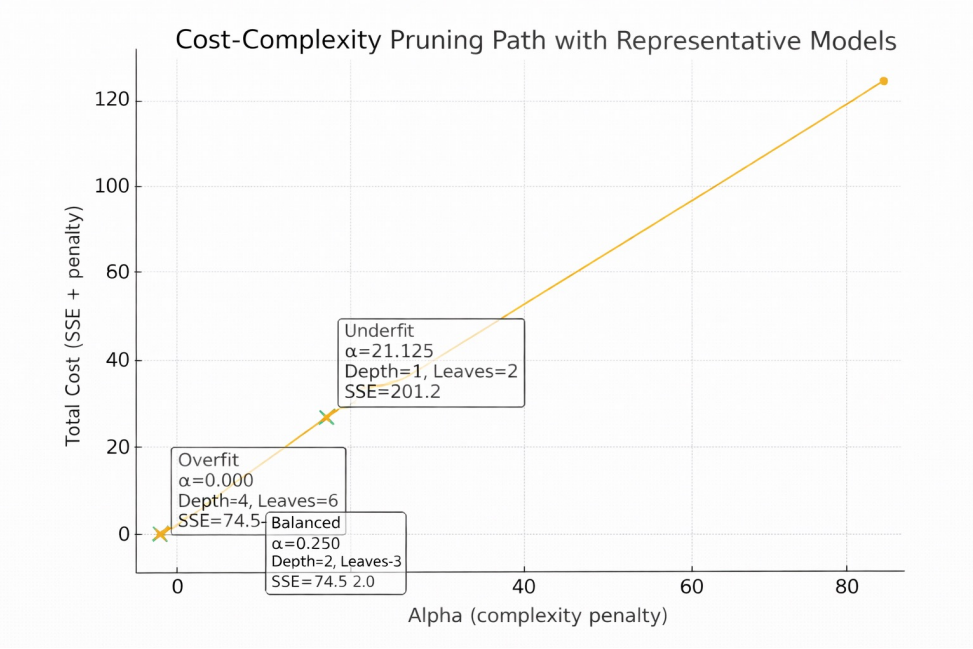
Number of terminal nodes in tree J

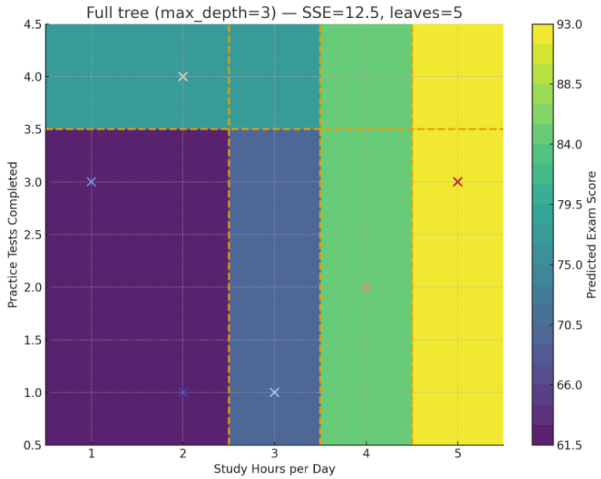
- Start with a large tree (fully grown)
- Iteratively remove branches (prune subtrees at internal nodes)
- At each step, compute SSE of the pruned tree & penalty $\alpha|J|$ (number of leaves)
- Choose the subtree that gives the lowest value of $C_\alpha(S)$

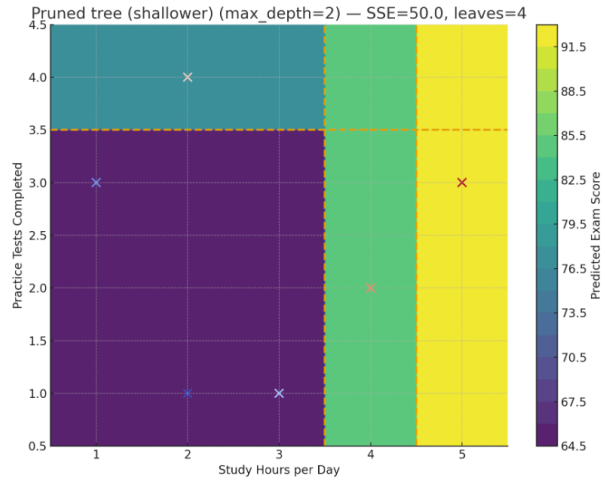
Why not just stop growing the tree earlier instead of pruning?

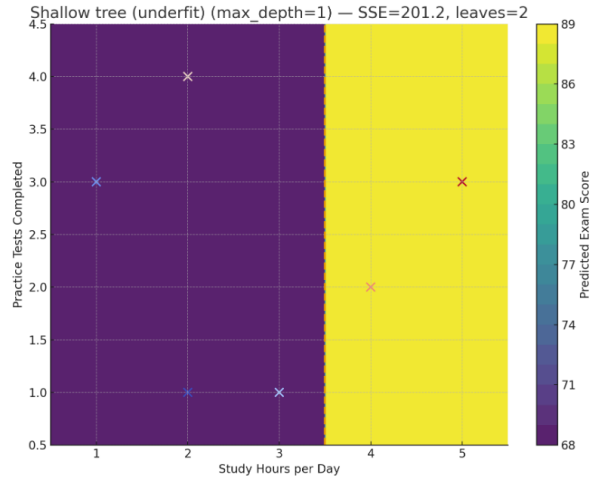
Continued Example

Pruning reduces overfitting by collapsing neighboring leaves that don’t meaningfully improve prediction accuracy









Full: $5.0 + 5\times6 = 35.0$

Depth-2: $35.2 + 5\times4 = 55.2$

Depth-1: $201.4 + 5\times2 = 211.4$

$$\alpha = 5$$

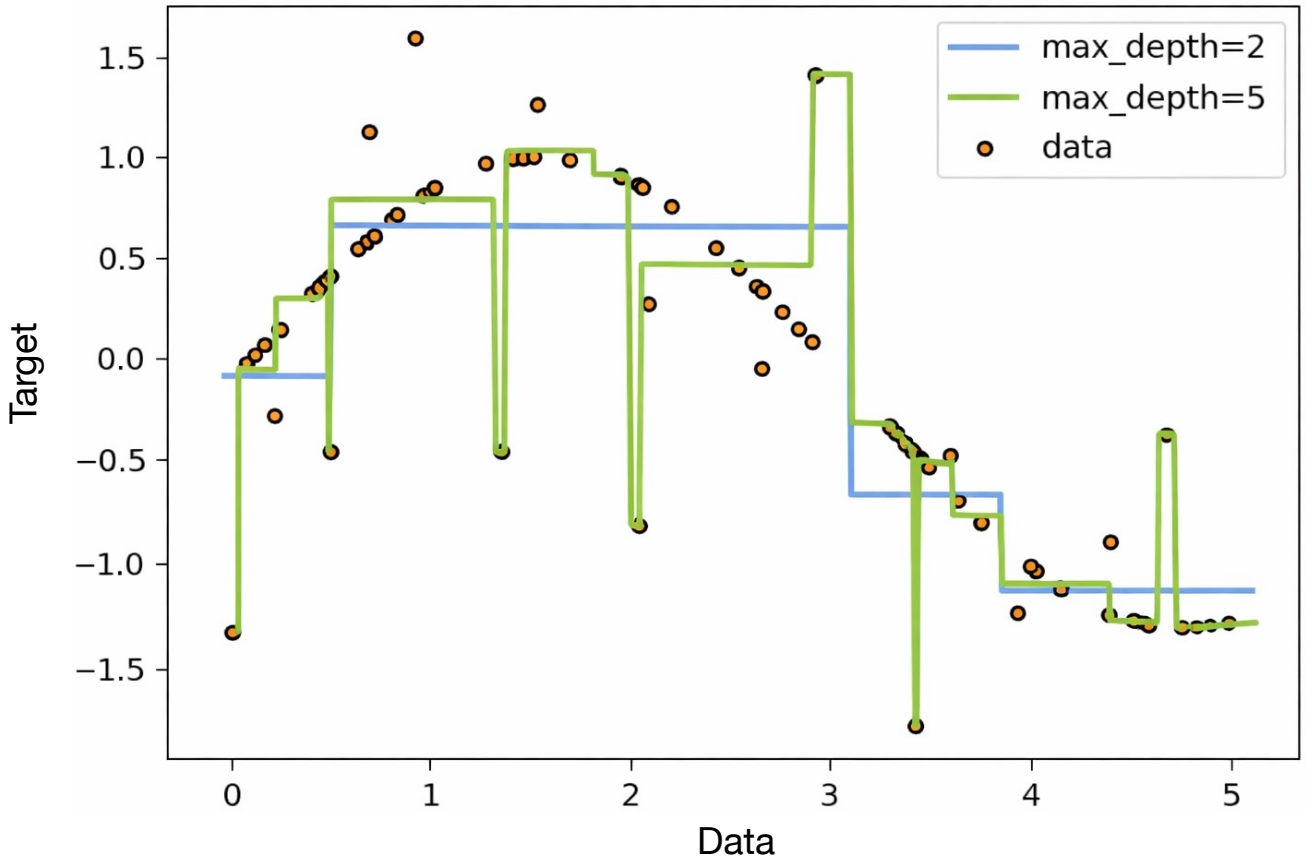
$$C_\alpha(S) = \mathrm{SSE} + \alpha|J|$$

Demo: Decision Tree Regression

When the maximum depth of the tree is set too high, the decision trees learn too fine details of the training data and learn from the noise

→ **overfitting**

https://scikit-learn.org/stable/auto_examples/tree/plot_tree_regression.html#sphx-glr-auto-examples-tree-plot-tree-regression-py



High depth → jagged fit → overfitting (noise-fitting)

Classification Trees

Classification Tree

Classification outcome takes value $1,2,\ldots,K$, which needs a different criterion for splitting node and pruning tree:

$$\hat{p}_{jk} = \frac{1}{N_j} \sum_{x \in R_j} \mathbb{I}(y_i = k)$$

$$k(j) = \arg\max_k \hat{p}_{jk}$$

$\hat{p}_{jk}$: estimated probability of class
N_j: number of samples in region R_j
$k(j)$: predicted class at node
k: a class label

Tree pruning:

$$C_\alpha(J) = \sum_{j=1}^{|J|} N_j Q_j(J) + \alpha |J|$$

$Q_j(J)$: average error per sample
$N_j Q_j(J)$: total error of the node

Leaf prediction = majority class; class probability = fraction in leaf

Different Measures of Error

Misclassification Error: ***Who wins?***

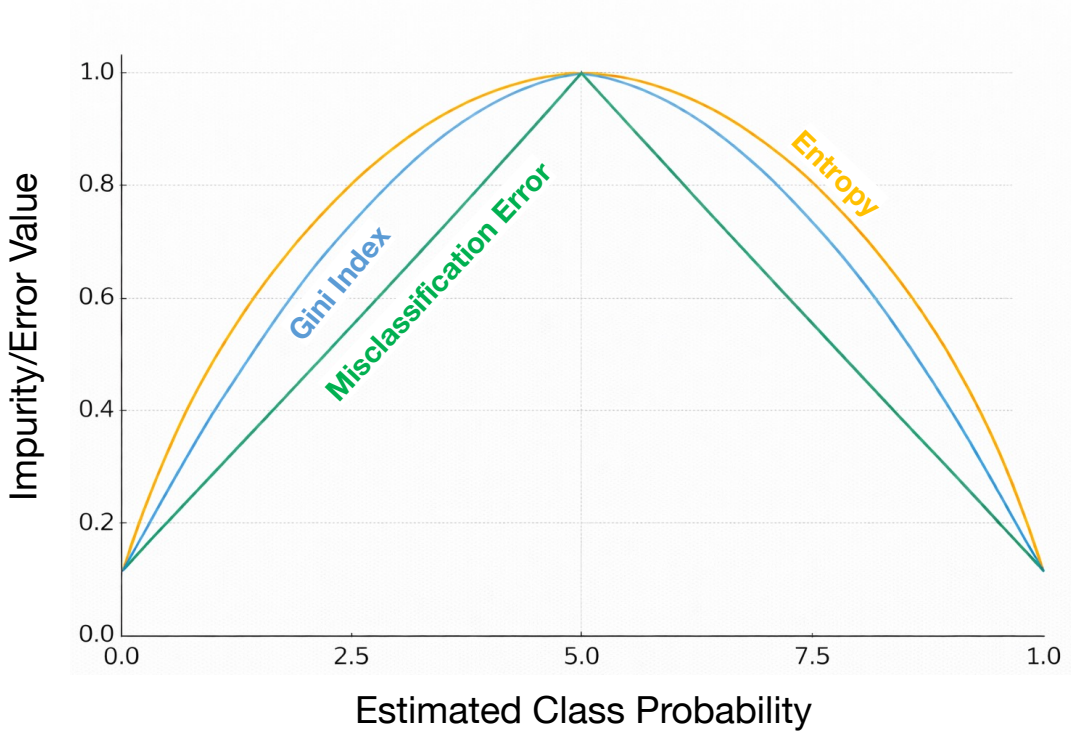
$$Q_j(J) = \frac{1}{N_j} \sum_{i \in R_j} \mathbb{I}(y_i \neq k(j)) = 1 - \hat{p}_{jk(j)}$$

Gini Index: ***How confident is the win?***

$$Q_j(J) = \sum_{k=1}^{K} \sum_{\substack{k'=1 \\ k \neq k'}}^{K} \hat{p}_{jk} \hat{p}_{jk'} = \sum_{k=1}^{K} \hat{p}_{jk}(1 - \hat{p}_{jk})$$

Cross-Entropy: ***How confident is the win?***

$$Q_j(J) = -\sum_{k=1}^{K} \hat{p}_{jk} \log \hat{p}_{jk}$$



→ Gini index and cross-entropy are more sensitive to change in $\hat{p}_{mk}$

Why not always use misclassification error if it is the simplest?

Example: Email Spam

- Information from 4601 email messages, 57 features
- Predict whether the email was “spam”
- Many features + noise → single tree unstable → ensembles
- Learn rules such as

```
if (%george < 0.6) & (%you > 1.5)  then spam
                                   else email
```

```
if (0.2×%you – 0.3×%george) > 0    then spam
                                   else email
```

- Using CART for classification tree

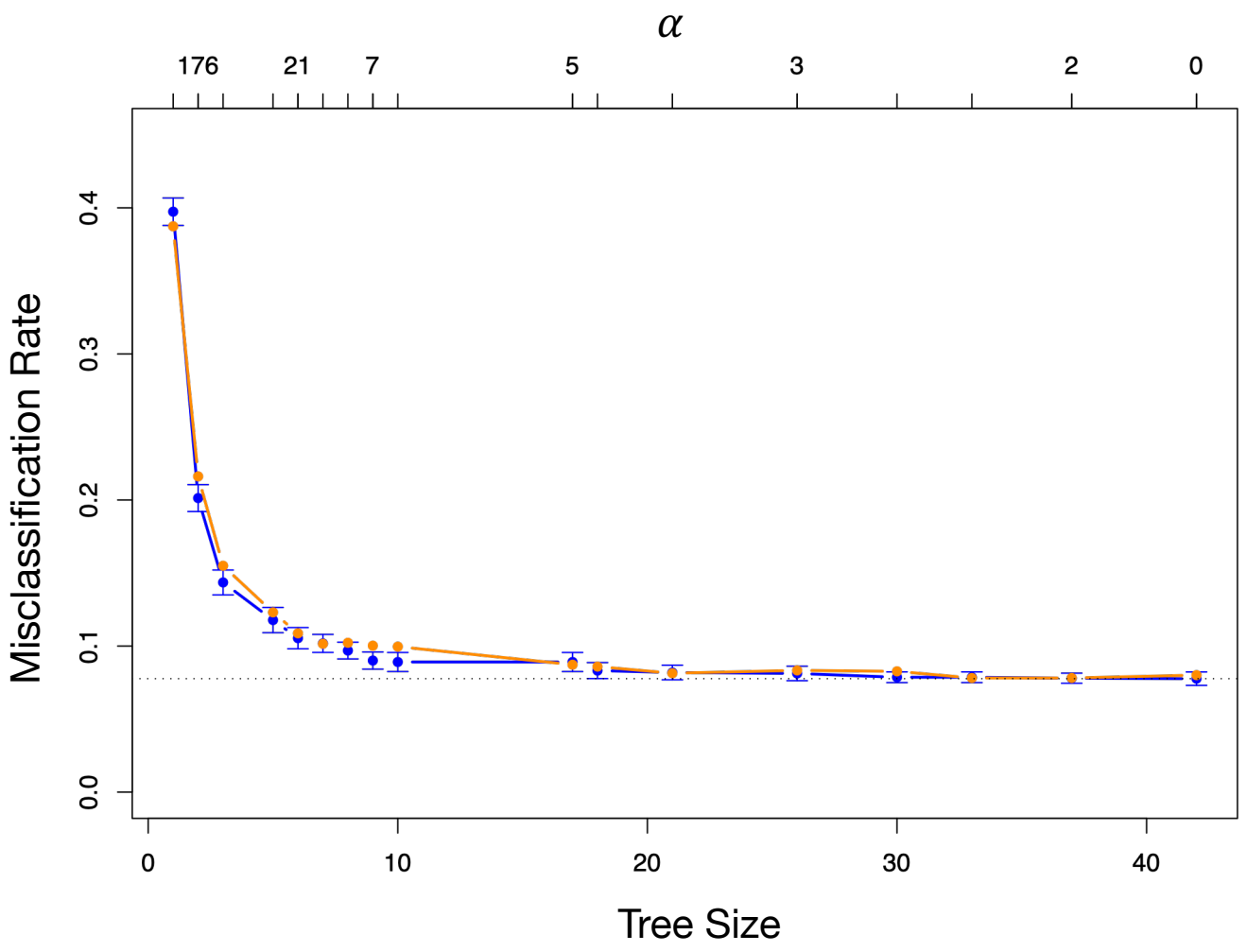
https://archive.ics.uci.edu/ml/datasets/spambase

Class	george	you	your	hp	free	hpl	!	our	re	edu	remove
spam	0.00	2.26	1.38	0.02	0.52	0.01	0.51	0.51	0.13	0.01	0.28
email	1.27	1.27	0.44	0.90	0.07	0.43	0.11	0.18	0.42	0.29	0.01

Example: Email Spam (cont.)

- Cross-validation chooses α that gives a 17-node tree
- Confusion matrix shows correct classifications vs. false positives and false negatives on test data

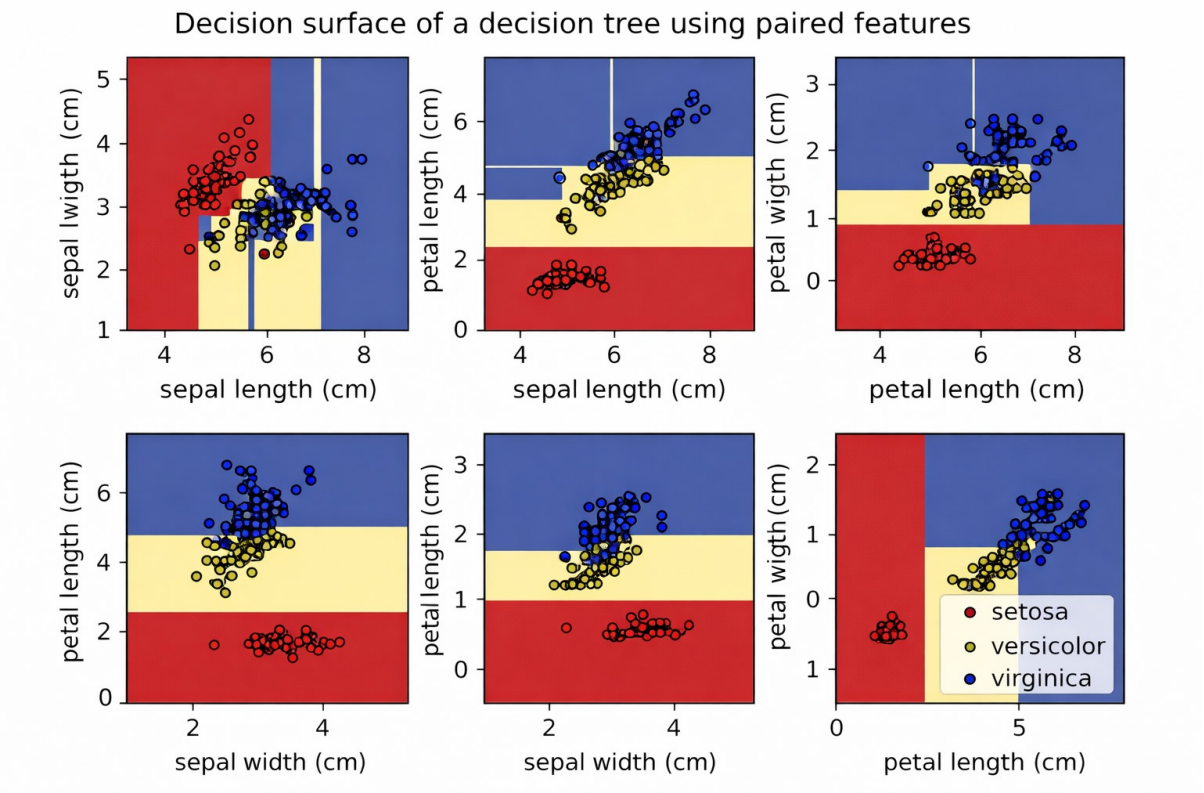
	Predicted	
True	email	spam
email	57.3%	4.0%
spam	5.3%	33.4%

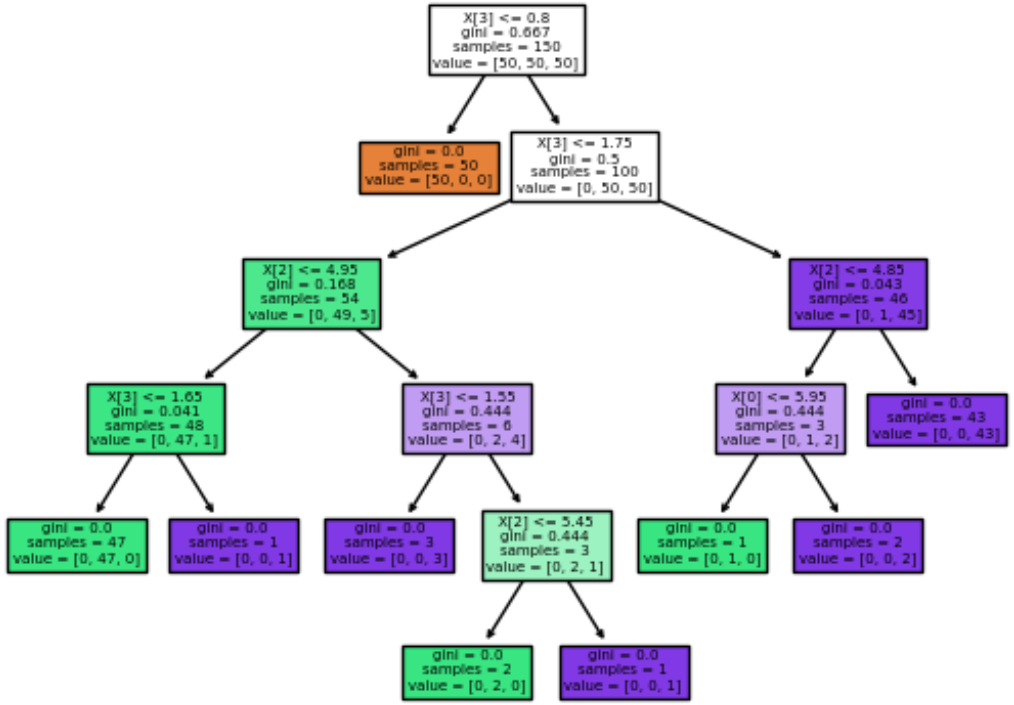


Overall error rate = 9.3%

Demo: CART for Fisher's Iris

A CART classification tree automatically selects features and axis-aligned split thresholds to partition the Iris dataset





https://scikit-learn.org/stable/auto_examples/tree/plot_iris_dtc.html

Random Forests: Motivation & Construction

Pros and Cons of Decision Trees

Pros of Decision Trees

- Capture complex, nonlinear patterns
- Automatically model feature interactions
- Interpretable (if–then rules)
- Handles mixed feature types

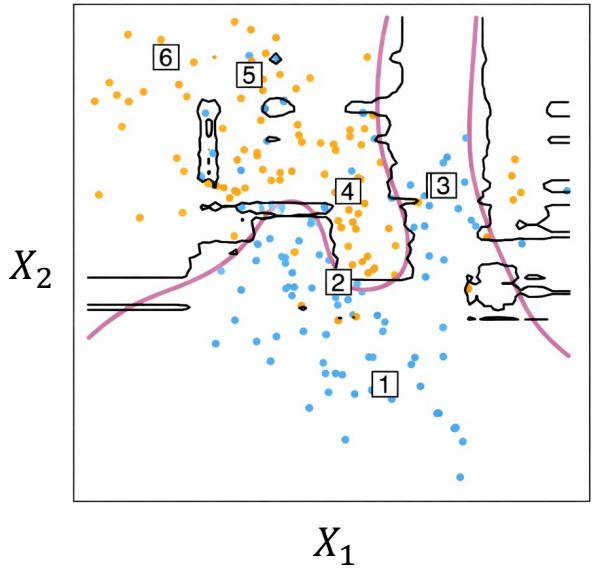
Cons of Decision Trees

- High variance and unstable
- Prone to overfitting
- Sensitive to noise

Random Forests for Classification & Regression (Bagging)

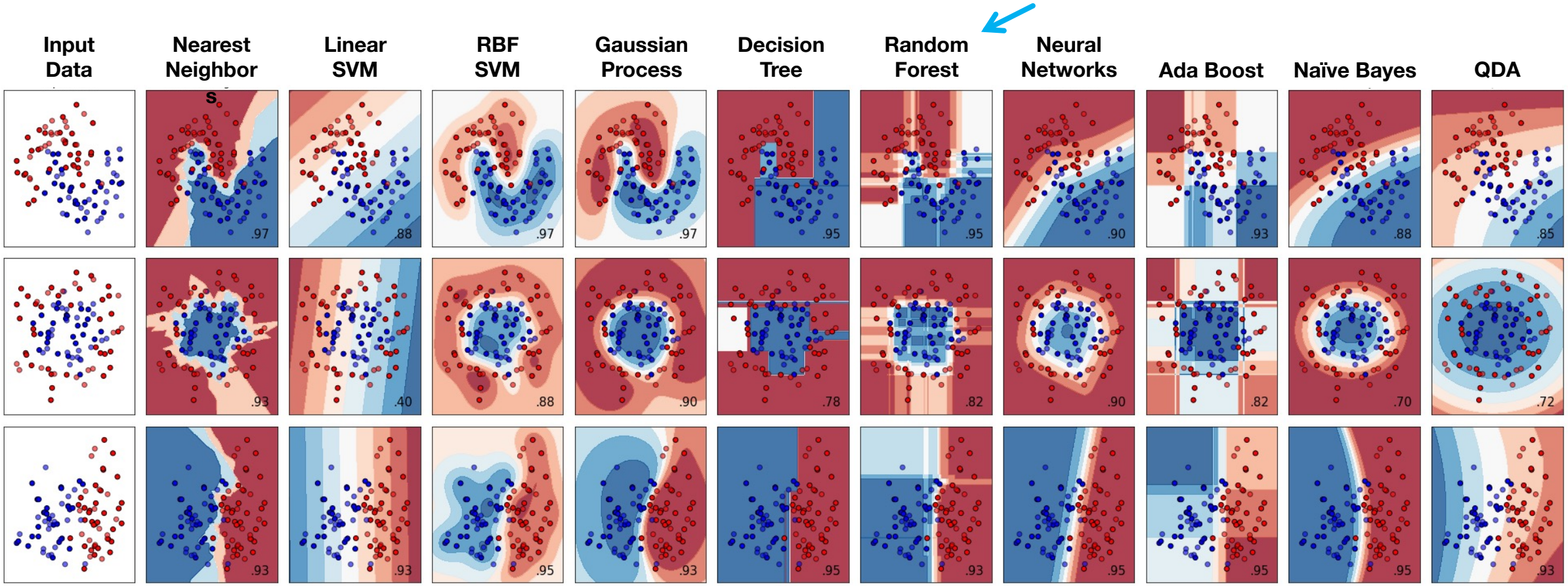
- Bootstrap samples from training data
- Grow a tree for each batch of bootstrap samples
- **Regression:** Average trees' predictions
- **Classification:** Take majority vote across trees

Random Forest Classifier



Classifiers Comparison

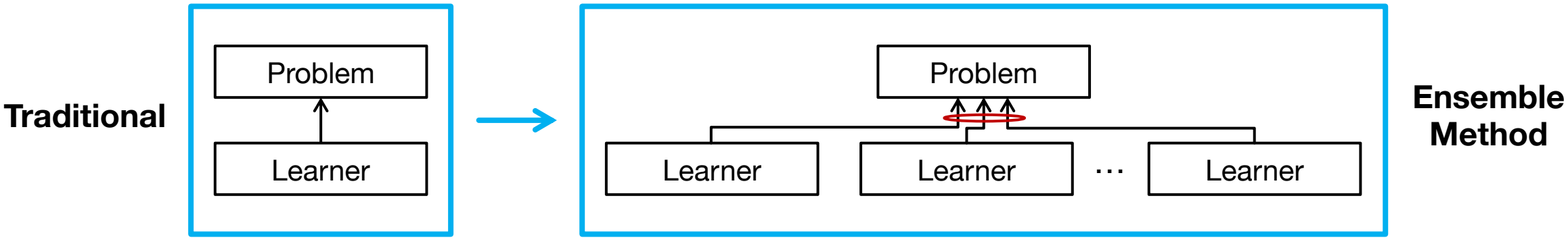
No free lunch: performance depends on data + bias–variance tradeoff.



Combination of Methods

"Can a set of weak learners create a single strong learner?" — Kearns & Valiant, 1988–89

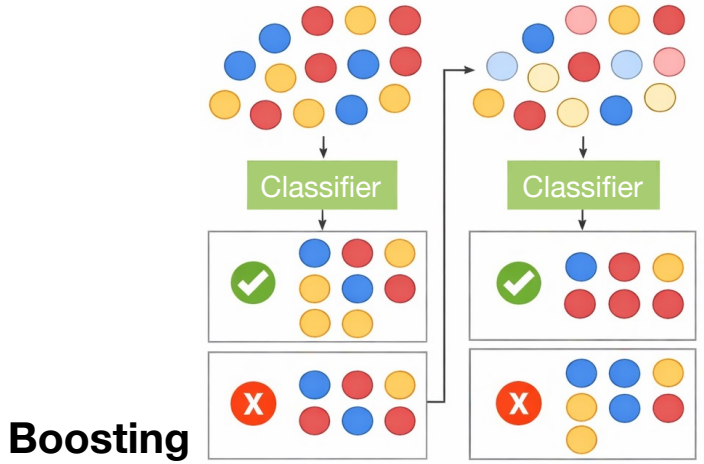
- In general machine learning tasks, there is no algorithm that is always the most accurate
- Different learners use different algorithms, hyperparameters, representations, training sets, and subproblems
- **Weak learner:** Any classifier that performs slightly better than random guessing
- **Key idea:** Instead of searching for one "best" model, construct a strong model by sequentially correcting mistakes made by simpler models



Two Main Ensemble Approaches

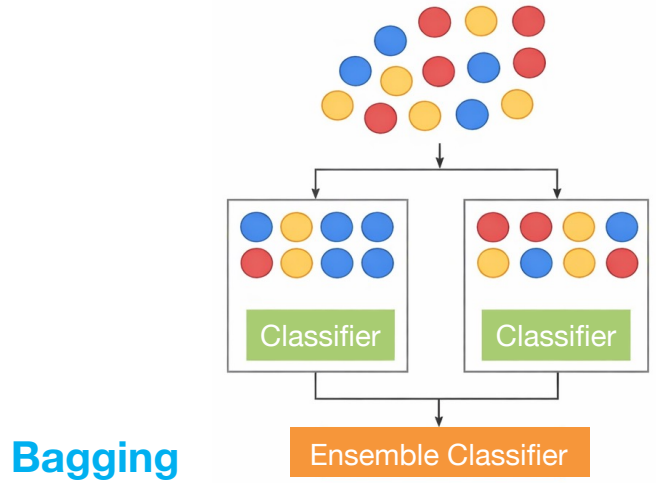
Boosting

- Run weak learner on weighted example set
- Combine weak learners **linearly**
- Require knowledge on the weak learner's performance
- Primarily reduces bias and sometimes variance



Bagging

- Run weak learners on bootstrap replicates of the training set
- Average weak leaners
- Primarily reduces bias



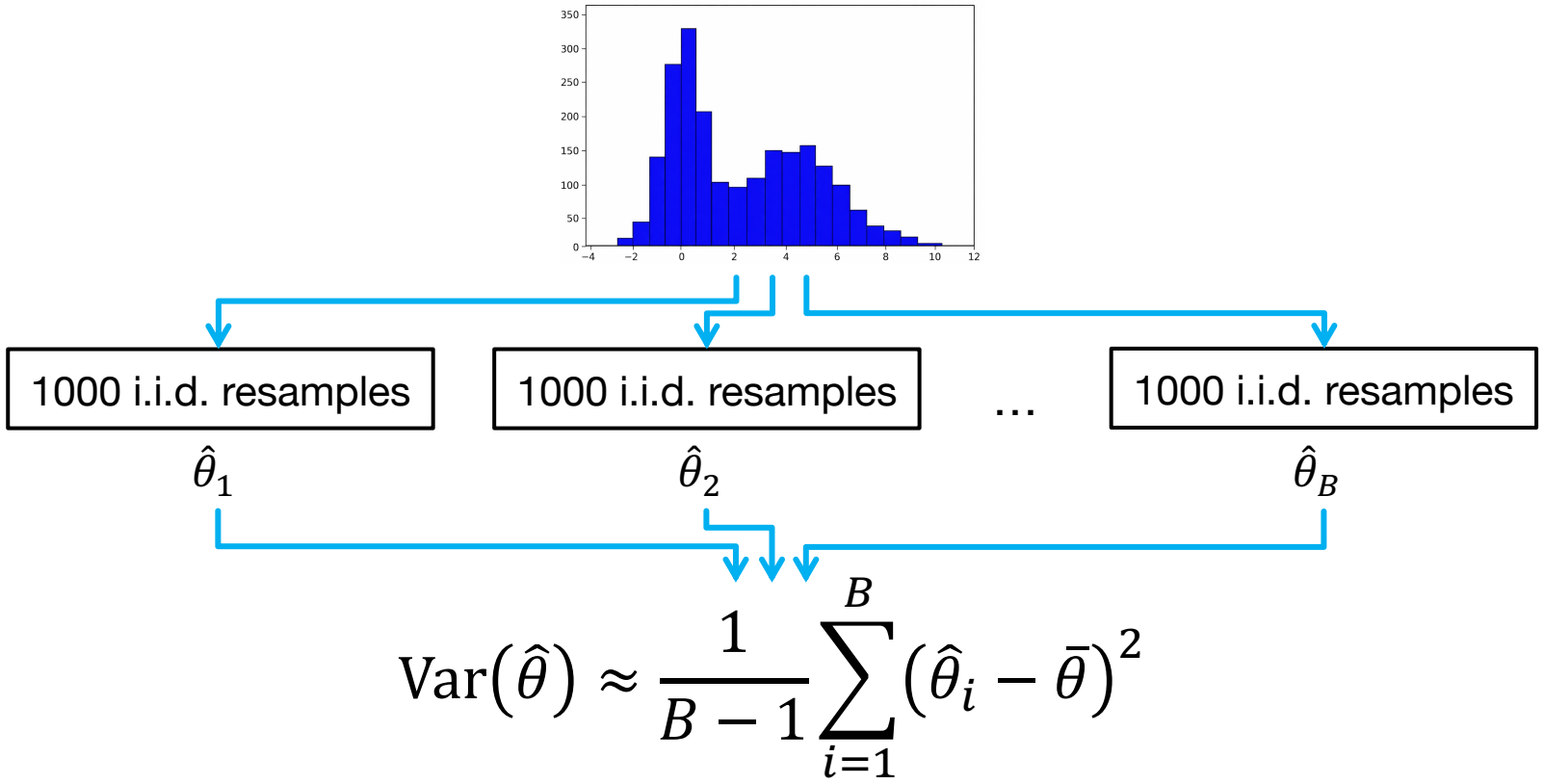
Bootstrap

- **Idea:** in statistics, we learn about the characteristics of the population by taking samples
- As the sample represents the population, analogous characteristics of the sample should give us information about the population characteristics
- Bootstrapping learns about the sample characteristics by taking resamples and using the information to infer the population
- **Resample:** We retake samples with replacement from the original samples
- Provides a powerful tool to calculate the standard error of an estimator, construct confidence intervals, and many other uses

Bootstrap sample = N draws with replacement from N training points

Example: Estimate Variance of Estimator

To decide the variance of an estimator $\hat{\theta}$, use data to generate B batches of new samples and evaluate $\hat{\theta}$ for each resampled batch



$$\mathrm{Var}(\hat{\theta}) \approx \frac{1}{B-1}\sum_{i=1}^{B}\left(\hat{\theta}_i - \bar{\theta}\right)^2$$

Example: Restaurant Rating

You want to predict whether a new restaurant will get a good rating (≥4 stars) on Yelp

From past restaurants, you have:

- Price level ($, $$, $$$)
- Wait time
- Service quality (survey)
- Food quality (survey)
- Neighborhood
- Parking availability

Approach: Random forest voting

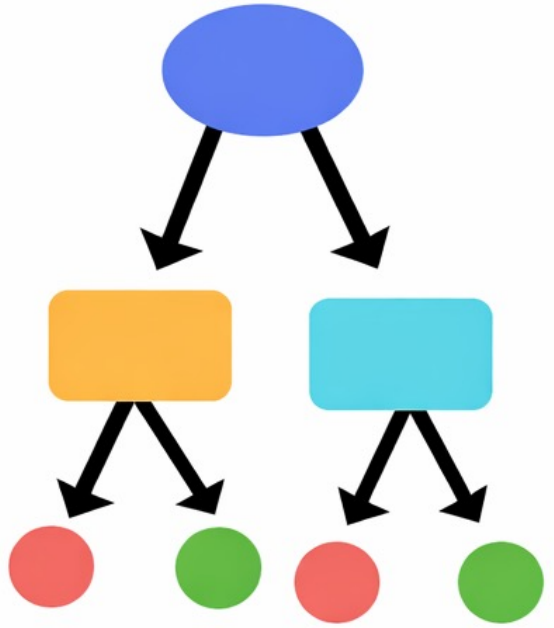
Example: Restaurant Rating (cont.)

A single decision tree might say:

```
if (food quality ≥ 7) & (service ≥ 6)  then good rating
                                       else bad rating
```

But this tree:

- Overfits unusual cases
- Changes drastically with small data changes
- Depends too much on one path of splits



One path can dominate → sensitive to small changes

Example: Restaurant Rating (cont.)

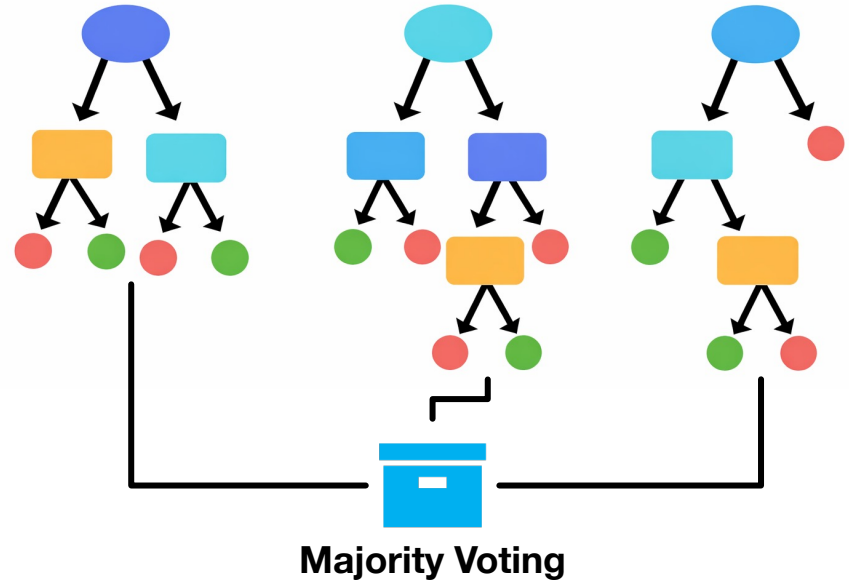
Instead of trusting one food critic, you ask **hundreds of different critics**, each:

- Looks at a random sample of restaurants
- Focuses on a random subset of features (e.g., prioritizing food quality, neighborhood, service, or price)
- Creates their own simple tree ("rule-of-thumb")

For a new restaurant:

- Each tree votes "good" or "not good"
- **The forest takes a majority vote** ☺

Why does bagging reduce overfitting?



Random Forest Algorithm

Input: training data, number of trees B, number of observations N, number of features p, number of randomly selected features ν

Output: ensemble of trees T_b, prediction rule

for $b = 1, \ldots, B$:

    Draw bootstrap sample Z_b of size N from training data

    Grow a random forest tree T_b for the bootstrapped data:

        for each node of the tree:

            Select ν variables at random from the p variables

            Pick the best variable/split among the ν

            Split the node into two child nodes

if Regression:

$$\hat{y}(x) = (1/B)\,\Sigma_{b=1}^{B} T_b(x)$$

if Classification:

$$\hat{y}(x) = \text{majority vote of } \{T_b(x), b = 1, \ldots, B\}$$

Return: ensemble of trees T_b, $b = 1, \ldots, B$, $\hat{y}(x)$

Random Forest: Mechanics & Theory

How to Tune Randomness

Effect of Averaging

- Average of B i.i.d. random variables, each with variance σ^2, has variance σ^2/B
- If the variables are dependent with positive correlation ρ, the variance of the average is $\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$

Correlation

The correlation between pairs of bagged tree limit the benefit of averaging

Candidates per Split

Before each split, select $\nu \leq p$ of features at random as candidates of splitting

- For classification, $\nu \approx \lfloor\sqrt{p}\rfloor$
- For regression, the default value $\nu \approx \lfloor p/3 \rfloor$

Effect of Averaging: i.i.d. Case

Example: Predicting house prices with many small, noisy models

- Suppose you build 100 tiny decision trees, each trained on a bootstrap sample, and each tree predicts:

  "The house price is around $500k ± some noise"

- If each tree has variance σ^2, and the trees' errors are independent, then averaging 100 predictions:
  - Cancels out noise
  - Variance reduces to $\sigma^2/100$ (e.g., like 100 students guessing the weight of an object—averaging vastly reduces the noise)

Averaging i.i.d. trees → variance drops fast

Effect of Averaging: Positive Correlated Case

Example: Trees all picking the same strong predictor

- Imagine one feature dominates the data: **House size** is the most predictive
  - Most trees use house size early in the split
  - The trees begin to look similar
  - Their prediction errors become positively correlated ($\rho > 0$)

- So, the variance becomes:

$$\rho\sigma^2 + \frac{(1-\rho)}{B}\sigma^2 \approx \rho\sigma^2 \longrightarrow$$ If B is large enough

  e.g., if all 100 students copy the same friend's guess, the average won't improve much

If Corr > 0, variance reduction is slower—gains plateau at a non-zero limit

Randomly Choosing Features at Splits

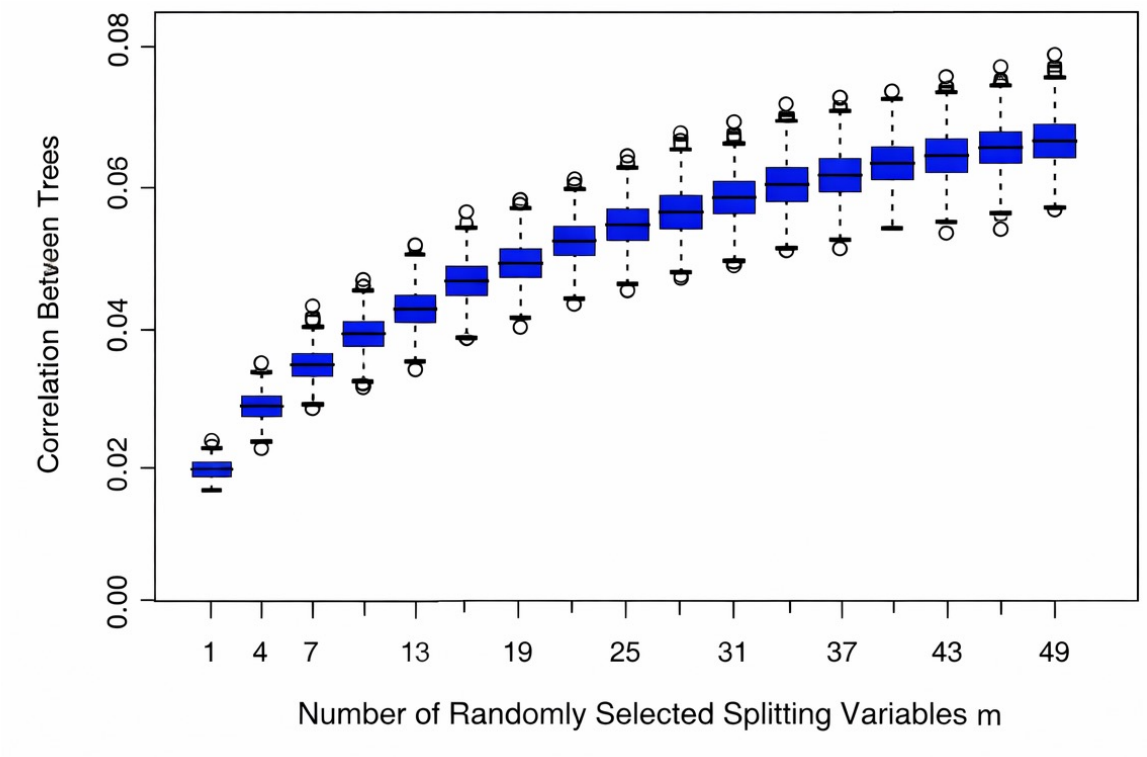
Why randomly choose features at each split? Force diversity between trees

- Suppose $p = 16$ total features
- For classification, choose: $v \approx [\sqrt{p}] = 4$ features per split
- Concrete scenario: Predicting if a patient has diabetes
- At each split, a tree might see only 4 random features like glucose level, BMI, age, blood pressure
- Another tree may see skin thickness, insulin level, BMI, Pregnancies
- Thus, each tree explores different relationships, increasing diversity → lowering correlation → more averaging benefit

Does every tree see every feature eventually?

Tree Correlation

Random feature selection reduces correlation especially when one feature is strong



Correlations between pairs of trees drawn by a random forest regression algorithm, as function of m

Why doesn't correlation drop to zero?

Out-of-Bag (OOB) Error

Out-of-Bag (OOB) Samples

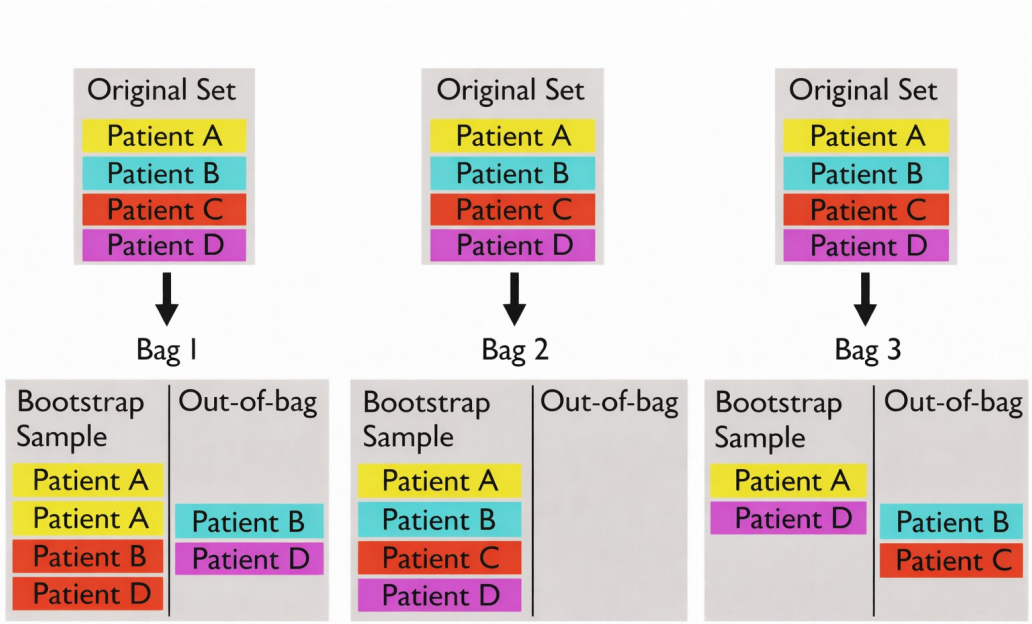
- For each observation $z^i = (x^i, y^i)$, construct its random forest by averaging only those trees corresponding to bootstrap samples in which z^i did not appear
- OOB error estimate is almost identical to that obtained by N-fold cross validation
- Help to decide the number of trees: Once OOB error stabilizes, the training can be terminated

OOB Error ≈ Cross Validation

- Each observatiob gets predicted by many trees that never saw it
- This is effectively like doing cross-validation automatically
- But without splitting the dataset and retraining repeatedly

Bagging Process

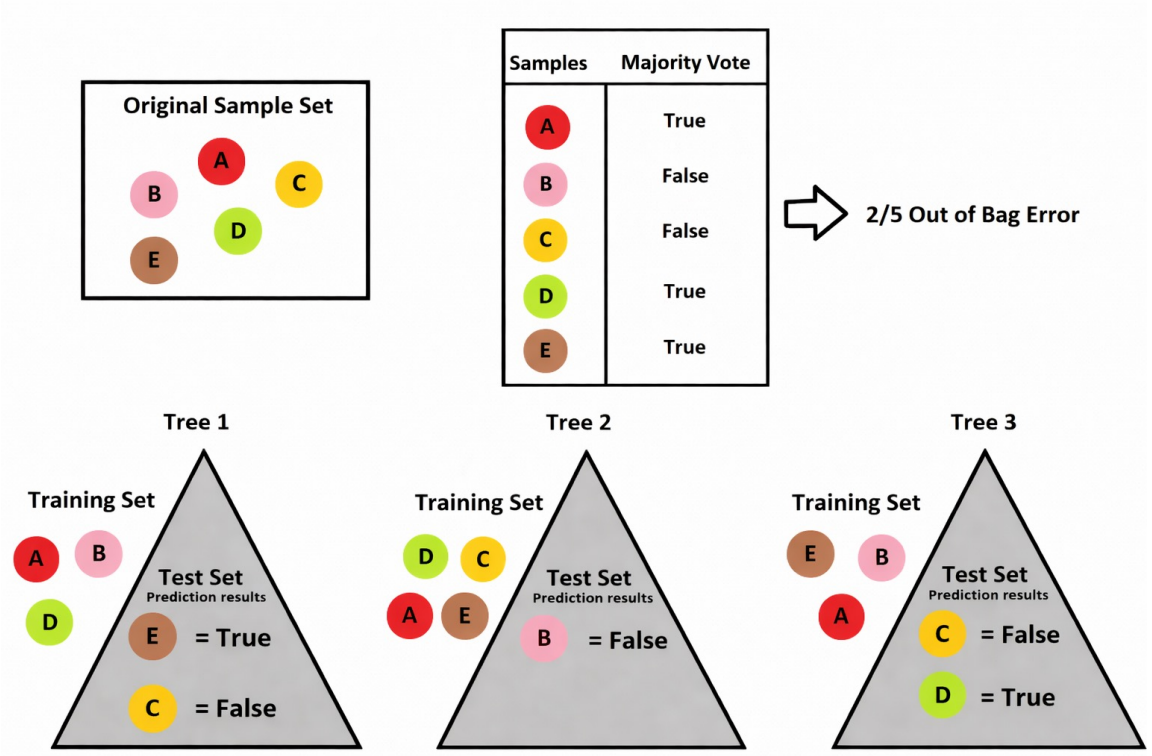
Visualizing the bagging process: Sampling 4 patients from the original set with replacement and showing the out-of-bag sets—only patients in the bootstrap sample would be used to train the model for that bag



Why don't we need to retrain models for OOB evaluation?

OOB Prediction Aggregation

OOB prediction aggregation (vote/average)



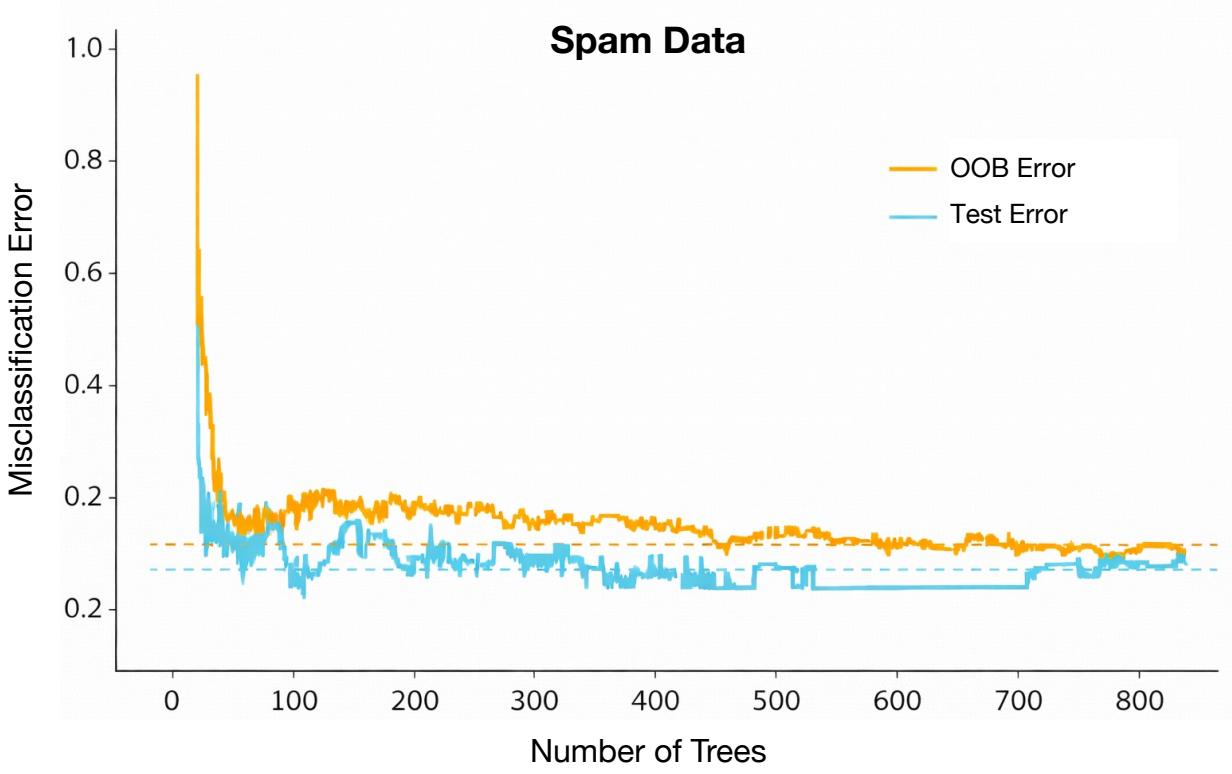
Why does averaging across many weak predictors help here?

OOB Error & The Number of Trees

OOB error helps choose the number of trees—stop adding trees when OOB curve plateaus (diminishing returns)

As you train more trees and compute OOB, you can often see the error decreasing and then flattening:

50 trees → 9.0% OOB error
100 trees → 6.1%
250 trees → 5.3%
400 trees → 5.1%
800 trees → 5.1%



Random Forest: Variable Importance

What Makes a Variable Important?

Definition: How much prediction accuracy drops when a feature's values are randomly permuted in OOB samples

- Use OOB samples to measure prediction strength of each variable
- When b^{th} tree is grown, measure prediction accuracy of OOB samples
- j^{th} variable randomly permuted in OOB sample, and accuracy is computed again
- Randomization effectively voids the effect of a variable (considering setting coefficient to 0 in linear regression model)
- Compute average decrease in accuracy as a measure of variable importance

Common types: (1) permutation importance; (2) impurity-based importance

Example: House Price Prediction

Suppose you train a random forest to predict house prices, using these features:

- X_1: Square footage
- X_2: Number of bedrooms
- X_3: Neighborhood quality score
- X_4: Age of house
- X_5: Distance to subway station

You want to know: Which features matter most?

- Correlated features can “share” importance
- High-cardinality features may be favored (impurity importance)
- Random forests use OOB samples to measure importance without building new models

Example: House Price Prediction (cont.)

Step 1: Baseline OOB accuracy

For each tree in the forest:

- Look at the OOB samples (houses it did not train on)
- Compute how accurately this tree predicts price on those samples

Say Tree #12 predicts OOB samples with 90% accuracy: This sets the **baseline performance**

X_1: Square footage
X_2: Number of bedrooms
X_3: Neighborhood quality score
X_4: Age of house
X_5: Distance to subway station

Why not use training accuracy as the baseline?

Example: House Price Prediction (cont.)

Step 2: Destroy one variable (permute it)

For variable X_3, random forest says:

> *"Let me shuffle this variable randomly among the OOB houses—keeping everything else the same—so the neighborhood information becomes meaningless."*

This is equivalent to:

❌ "Setting its coefficient to 0" in linear regression

❌ "Erasing the variable's signal"

After shuffling X_3, Tree #12 might now get 78% accuracy

OOB accuracy to drop: 90% → 78% (drop = 12%)

X_1: Square footage
X_2: Number of bedrooms
X_3: Neighborhood quality score
X_4: Age of house
X_5: Distance to subway station

Example: House Price Prediction (cont.)

Step 3: Interpret the drop

If permuting X_3 makes the prediction **much worse**, then:

- X_3 was very important
- The tree depended on the neighborhood score to make decisions
- Destroying X_3 removed real signal

If permuting X_1 (square footage) only drops accuracy by 2%, then: $\boldsymbol{X_3}$ **is more important than** $\boldsymbol{X_1}$

X_1: Square footage
X_2: Number of bedrooms
X_3: Neighborhood quality score
X_4: Age of house
X_5: Distance to subway station

Can a feature be useful but still have low importance?

Example: House Price Prediction (cont.)

The forest averages the accuracy drop across all trees:

Variable	Avg OOB Accuracy Drop	Interpretation
X_3	**12%**	The most important predictor
X_1	2%	Somewhat important
X_5	1%	Barely matters
X_2	0.3%	Barely matters
X_4	0%	Might be irrelevant

→ *We measure importance by seeing how much worse the model performs when we randomly scramble one variable—if scrambling hurts performance a lot, the variable was important*

X_1: Square footage
X_2: Number of bedrooms
X_3: Neighborhood quality score
X_4: Age of house
X_5: Distance to subway station

Why not trust a single tree's importance?

Other Examples

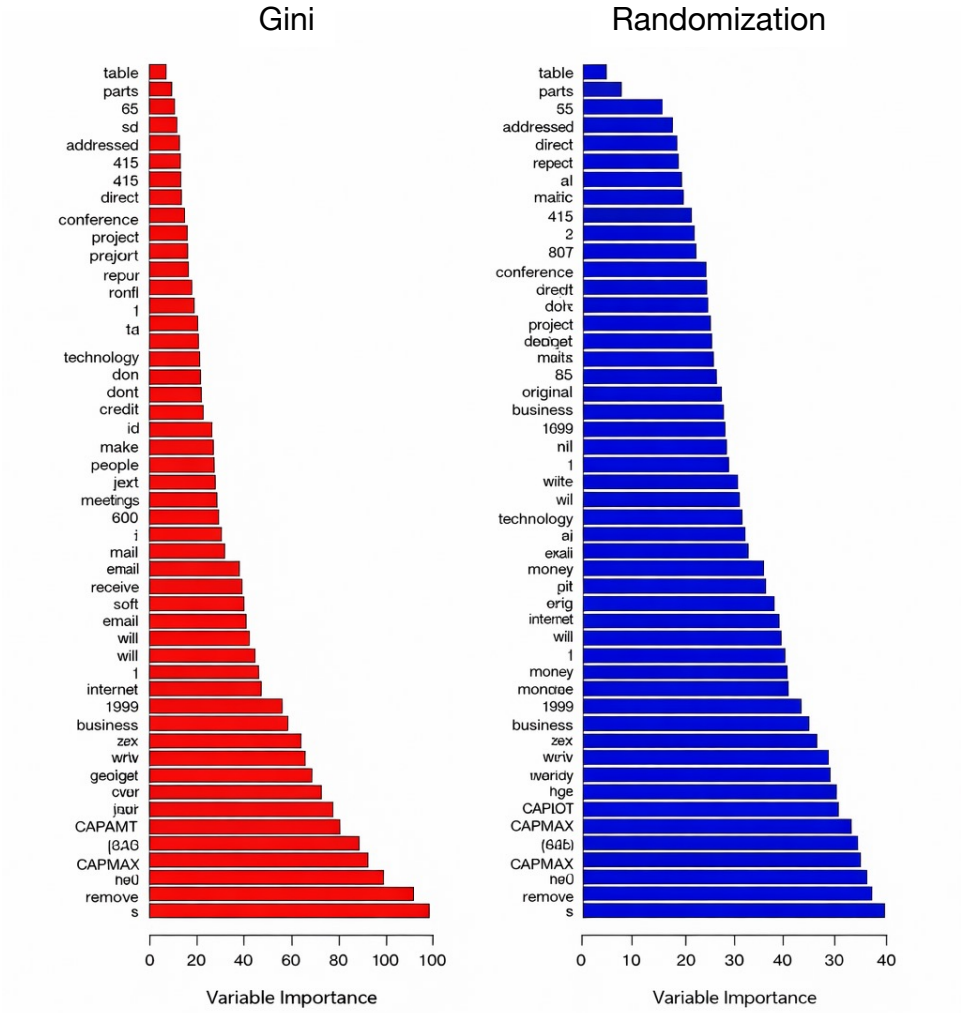
Correct income predictions based on OOB samples: 82% (≤50K: 91.13%; >50K: 57.03%)—**permutation importance is model-agnostic**; works for any predictor

	<=50K	>50K	MeanDecreaseAccuracy	Importance (MeanDecreaseGini)
occupation	0.022	0.063	0.032	115.53
age	-0.001	0.065	0.016	108.30
education_num	0.023	0.077	0.036	96.92
relationship	0.022	0.079	0.036	80.74
hrs_per_week	0.002	0.039	0.011	67.80
marital	0.024	0.066	0.034	61.21
workclass	0.007	-0.006	0.004	41.75
country	0.000	-0.006	-0.001	21.85

n = 2000 cases used in estimation;

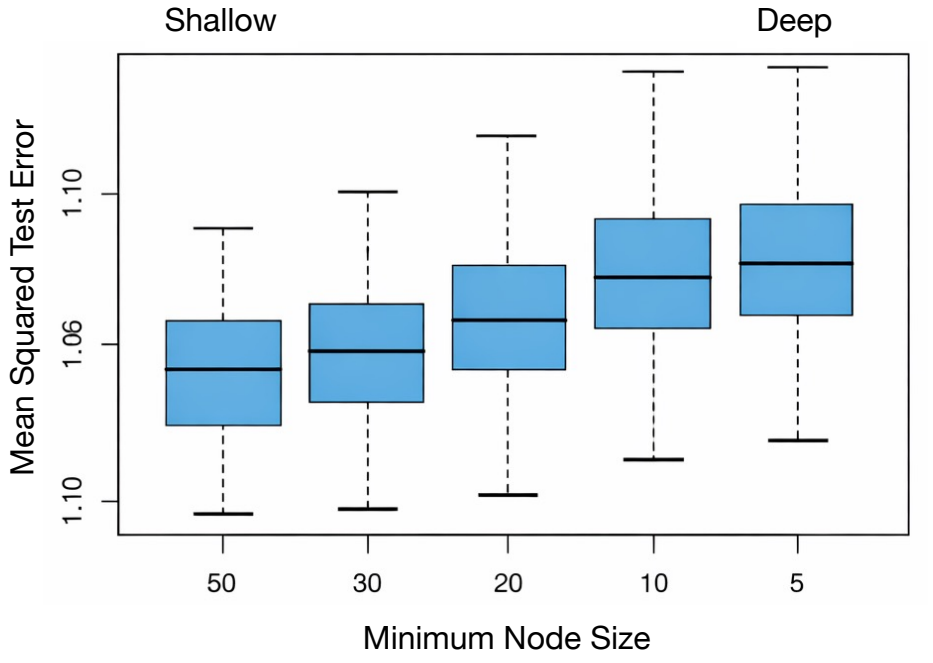
Other Examples

- Variable importance plots for a classification random forest grown on the **spam data**
- The left plot bases the importance on the Gini splitting index, as in gradient boosting
- The rankings compare well with the rankings produced by gradient boosting
- The right plot uses OOB randomization to compute variable importances and tends to spread the importances more uniformly

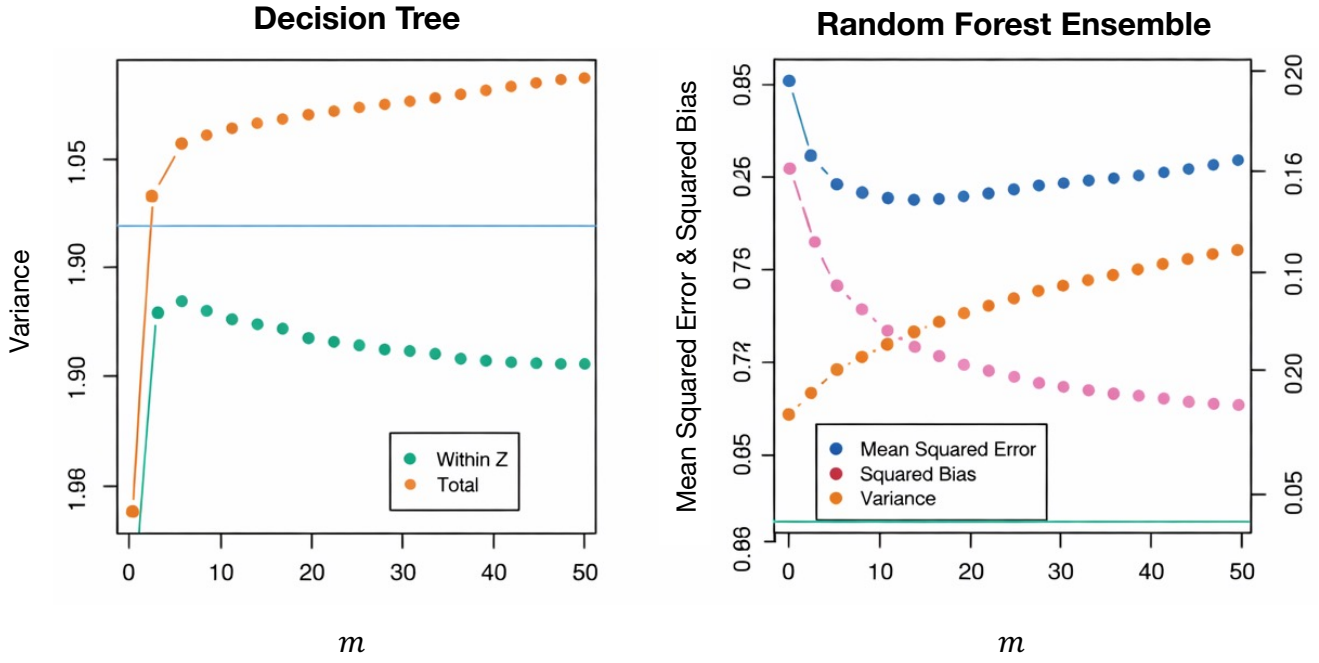


Bias–Variance Tradeoff

As trees become deeper (smaller minimum node size), random forest test error tends to increase, indicating **overfitting**:



Increasing the number of candidate features m raises single-tree variance but reduces ensemble variance, yielding a **bias–variance tradeoff** that determines random forest error:



Key Takeaways

What We Learned This Week

- Decision trees model data using recursive if–then rules and handle nonlinear interactions with minimal preprocessing
- Tree depth and pruning control the bias–variance tradeoff: deeper trees fit noise, shallower trees generalize better
- CART uses greedy splits to partition feature space into interpretable regions
- Random forests reduce variance by averaging many trees trained on bootstrap samples with random feature selection
- OOB error provides an efficient alternative to cross-validation for random forests
- Feature importance can be assessed via impurity reduction or permutation-based performance drops
- Boosting trees build models sequentially to correct errors and can outperform random forests with careful tuning