

## L04: Random Forests

Deep Dive: Theory, Implementation, and Applications

Methods and Algorithms

Spring 2026

# Outline

- 1 Decision Tree Foundations
- 2 Ensemble Methods
- 3 Random Forests Algorithm
- 4 Bias-Variance and Tuning
- 5 Practical Considerations
- 6 From Bagging to Boosting
- 7 Practice
- 8 Implementation
- 9 Summary
- 10 References

## From Rules to Trees

- Decision trees encode if-then-else rules
- Each node splits data based on a feature threshold
- Leaves contain predictions (class or value)

## Key Questions

- How to choose the best split?
- When to stop splitting?
- How to make predictions?

---

Decision trees: the building blocks of Random Forests

## Splitting Criteria: Gini Impurity

**Gini Impurity** measures class mixture at a node:

$$G = 1 - \sum_{k=1}^K p_k^2$$

where  $p_k$  is the proportion of class  $k$  samples.

**Properties:**

- $G = 0$ : pure node (all samples same class)
- $G = 0.5$ : maximum for binary; general:  $G_{max} = 1 - 1/K$  for  $K$  classes
- Lower Gini = better split

---

**Gini impurity:** probability of misclassifying a random sample

## Splitting Criteria: Information Gain

**Entropy** measures disorder:

$$H = - \sum_{k=1}^K p_k \log_2(p_k)$$

**Note:** By convention,  $0 \cdot \log_2(0) = 0$  (via L'Hôpital's rule)

**Information Gain:**

$$IG = H(\text{parent}) - \sum_j \frac{n_j}{n} H(\text{child}_j)$$

**Comparison:**

- Gini: faster to compute, tends to isolate most frequent class
- Entropy: more balanced trees, slightly slower
- In practice: similar performance

---

Both criteria aim to create pure child nodes

## Regression Trees: MSE Criterion

For regression, use Mean Squared Error:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \bar{y})^2$$

Split quality:

$$\text{Reduction} = \text{MSE}(\text{parent}) - \sum_j \frac{n_j}{n} \text{MSE}(\text{child}_j)$$

Leaf prediction: mean of samples in leaf

---

Trees can handle both classification and regression tasks

## Recursive Partitioning:

1. Start with all samples at root
2. For each feature and threshold:
  - Calculate impurity reduction
  - Select split with maximum reduction
3. Create child nodes with split samples
4. Recurse until stopping criterion met

## Stopping Criteria:

- Maximum depth reached
- Minimum samples per leaf
- No improvement in impurity

---

Greedy is necessary: finding the globally optimal tree is NP-complete

**Classification and Regression Trees** (Breiman et al., 1984):

**Require:** node, dataset  $\mathcal{D}$ , depth

- 1: **if** stopping criterion met **then**
- 2:   **return** leaf with prediction  $\hat{y} = \text{majority}(\mathcal{D})$  or  $\bar{y}$
- 3: **end if**
- 4: **for** each feature  $j$  and threshold  $s$  **do**
- 5:   Compute impurity reduction:  $\Delta I = I(\mathcal{D}) - \frac{|\mathcal{D}_L|}{|\mathcal{D}|} I(\mathcal{D}_L) - \frac{|\mathcal{D}_R|}{|\mathcal{D}|} I(\mathcal{D}_R)$
- 6: **end for**
- 7:  $(j^*, s^*) = \arg \max_{j,s} \Delta I$
- 8: Split  $\mathcal{D}$  into  $\mathcal{D}_L = \{x : x_j \leq s^*\}$  and  $\mathcal{D}_R = \{x : x_j > s^*\}$
- 9: Recurse on  $\mathcal{D}_L$  (left child, depth+1) and  $\mathcal{D}_R$  (right child, depth+1)

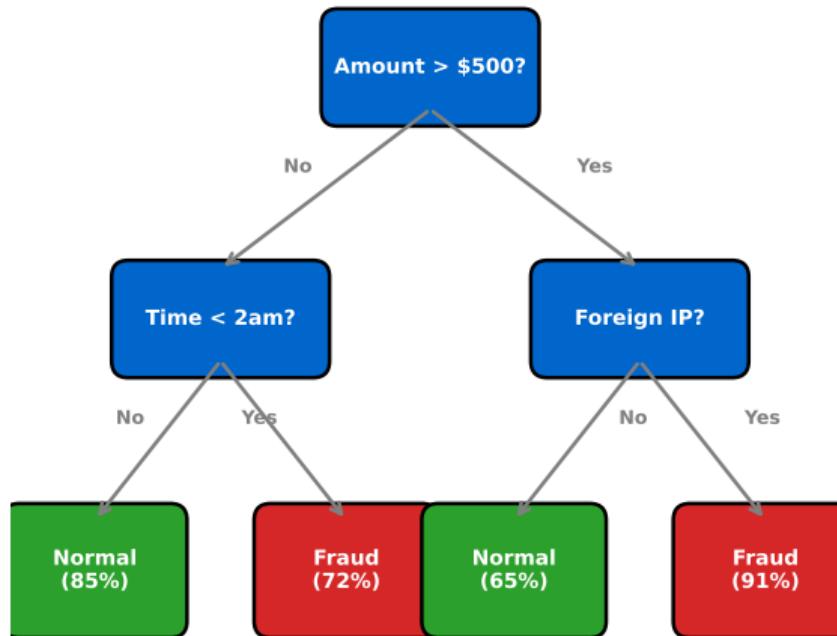
Impurity measures: Gini =  $1 - \sum_k p_k^2$ , Entropy =  $-\sum_k p_k \log_2 p_k$

---

Breiman et al. (1984). Classification and Regression Trees. Wadsworth.

# Decision Tree: Fraud Detection Example

## Decision Tree for Fraud Detection



[https://github.com/Digital-AI-Finance/methods-algorithms/tree/master/slides/L04\\_Random\\_Forests/01\\_decision\\_tree](https://github.com/Digital-AI-Finance/methods-algorithms/tree/master/slides/L04_Random_Forests/01_decision_tree)

Each path through the tree represents a fraud detection rule

## The Reality of Fraud Data

- Fraud is typically <1% of transactions (100:1 ratio)
- A model predicting “not fraud” for ALL transactions: 99% accuracy!
- **Accuracy is the WRONG metric** for imbalanced classification

## Correct Evaluation Metrics

- **Precision:** Of flagged transactions, how many are fraud?
- **Recall:** Of actual fraud, how many did we catch?
- **F1 Score:** Harmonic mean of precision and recall
- **AUC-PR:** Area under Precision-Recall curve (preferred over ROC)

---

NEVER use accuracy for imbalanced classification!

## 1. Class Weighting (scikit-learn)

- `class_weight='balanced'`: auto-adjust by class frequency
- `class_weight='balanced_subsample'`: per-tree balancing

## 2. Resampling Techniques

- **SMOTE**: Synthetic Minority Oversampling TEchnique
- **Undersampling**: Reduce majority class
- **Combination**: SMOTE + Tomek links

## 3. Threshold Tuning

- Default threshold 0.5 rarely optimal for imbalanced data
- Choose threshold based on business cost: FN cost >> FP cost for fraud

---

Always check class distribution BEFORE training any classifier!

## Part 2: Why Ensembles?

### Problem with Single Trees:

- High variance: small data changes → very different trees
- Prone to overfitting
- Unstable predictions

### Ensemble Solution:

- Train multiple diverse models
- Combine predictions
- Reduce variance while maintaining low bias

---

"Wisdom of crowds": aggregate many weak learners into strong learner

# Bootstrap Sampling

**Bootstrap:** sample with replacement from original data

**Properties:**

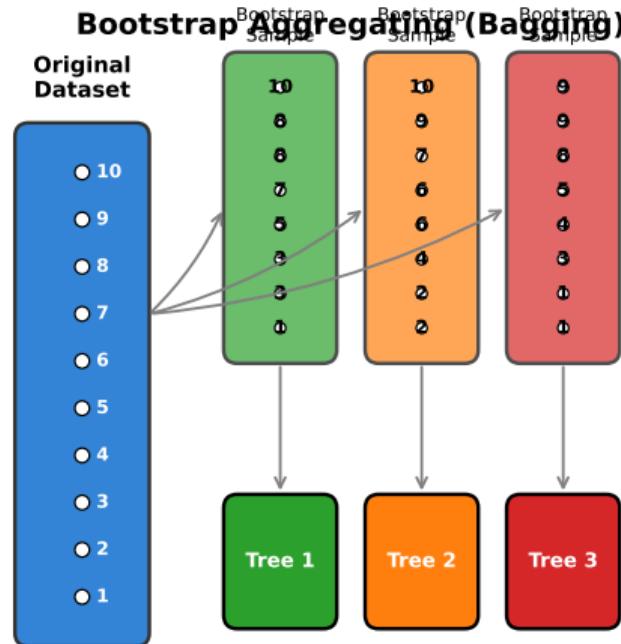
- Each sample: same size as original ( $n$  observations)
- Expected unique samples:  $\approx 63.2\%$  as  $n \rightarrow \infty$  (converges to  $1 - 1/e$ )
- Remaining  $\sim 37\%$ : out-of-bag (OOB) samples

**Effect:**

- Each tree sees different data subset
- Creates diversity among trees
- OOB samples provide validation

---

**Bootstrap:** key ingredient for reducing variance through aggregation



Each tree trained on ~63% unique samples (with replacement)

[https://github.com/Digital-AI-Finance/methods-algorithms/tree/master/slides/L04\\_Random\\_Forests/03\\_bootstrap](https://github.com/Digital-AI-Finance/methods-algorithms/tree/master/slides/L04_Random_Forests/03_bootstrap)

**Bootstrap Aggregating: train on random samples, aggregate predictions**

## Variance Reduction by Averaging

For  $B$  independent predictions with variance  $\sigma^2$ :

$$\text{Var} \left( \frac{1}{B} \sum_{b=1}^B \hat{f}_b(x) \right) = \frac{\sigma^2}{B}$$

With correlation  $\rho$ :

$$\text{Var} = \rho\sigma^2 + \frac{1 - \rho}{B}\sigma^2$$

Derivation: For correlated predictions with  $\text{Cov}(f_i, f_j) = \rho\sigma^2$ :

$$\text{Var}(\bar{f}) = \frac{1}{B^2} [B\sigma^2 + B(B-1)\rho\sigma^2] = \rho\sigma^2 + \frac{(1-\rho)\sigma^2}{B}$$

Key insight: Reduce correlation between trees to maximize variance reduction

---

Lower correlation between trees = greater ensemble benefit

## Part 3: Random Forests Algorithm

### Two Sources of Randomness:

1. **Bootstrap sampling**: each tree trained on random sample
2. **Feature randomization**: each split considers random subset

### Feature Subset Size (at each split):

- Classification:  $\sqrt{p}$  features (default)
- Regression:  $p/3$  features (default)
- Decorrelates trees more than bagging alone

---

Feature randomization: combines Ho's (1995) random subspace method with Breiman's bagging

# Random Forest Algorithm

## Training:

1. For  $b = 1$  to  $B$  trees:

- Draw bootstrap sample of size  $n$
- Grow tree:
  - At each node, select  $m$  features randomly (without replacement)
  - Find best split among  $m$  features
  - Split until stopping criterion

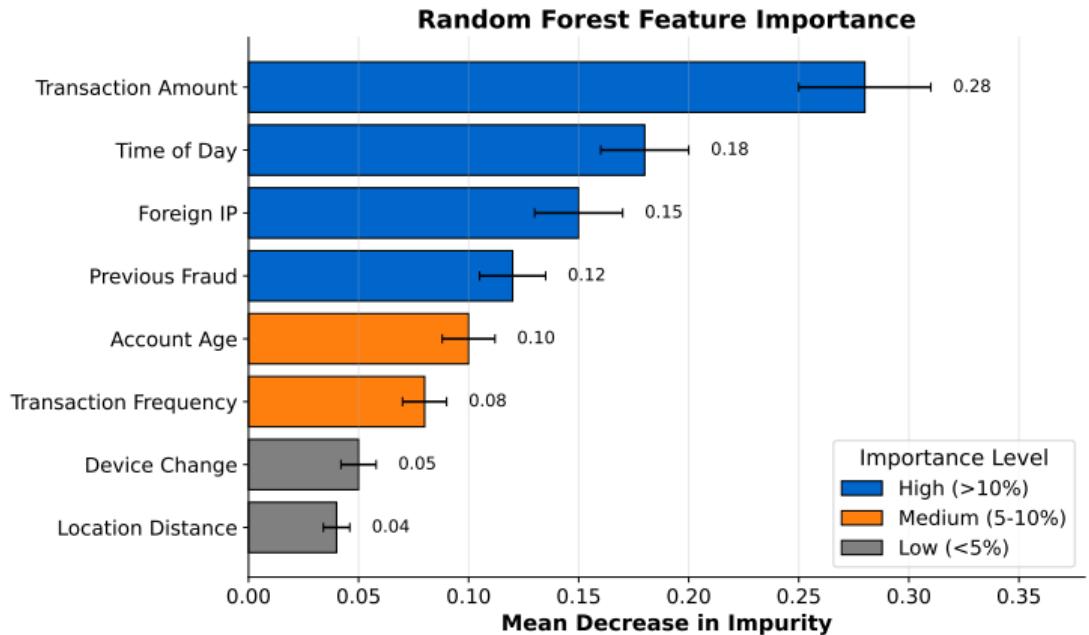
## Prediction:

- Classification: majority vote across trees
- Ties: broken randomly or by lowest class index
- Regression: average predictions

---

Typical: 100-500 trees; more trees improves accuracy (but increases memory/latency)

# Feature Importance



Mean Decrease in Impurity: sum of impurity reductions from splits on feature

## 1. Mean Decrease in Impurity (MDI):

$$\text{MDI}(j) = \frac{1}{T} \sum_{t=1}^T \sum_{\text{node } v \text{ splits on } j} p(v) \cdot \Delta I(v)$$

where  $p(v)$  = fraction of samples reaching node  $v$ ,  $T$  = number of trees.

- Fast to compute (comes free from training)
- Bias toward high-cardinality features

## 2. Permutation Importance:

- Permute feature values, measure accuracy drop
- More reliable, less biased
- Slower (requires re-evaluation)

---

Permutation importance preferred for final feature selection

**Problem:** How significant is a feature's importance score?

## Permutation Testing (Altmann et al., 2010)

1. Compute actual importance  $I_j$  for feature  $j$
2. Permute feature  $j$  labels, recompute importance ( $B$  times)
3. P-value =  $\frac{1 + \#(I_j^{\text{perm}} \geq I_j)}{1 + B}$

## Confidence Intervals

- Bootstrap RF training to get importance distribution
- Report 95% CI:  $[\hat{I}_j^{2.5\%}, \hat{I}_j^{97.5\%}]$
- Features with CI crossing zero may not be significant

---

Always report uncertainty in feature importance rankings

**SHapley Additive exPlanations** (Lundberg & Lee, 2017):

$$\phi_j = \sum_{S \subseteq \mathcal{F} \setminus \{j\}} \frac{|S|!(|\mathcal{F}| - |S| - 1)!}{|\mathcal{F}|!} [f(S \cup \{j\}) - f(S)]$$

**Properties** (unique among all feature attribution methods):

- **Efficiency:**  $\sum_{j=1}^p \phi_j = f(x) - \mathbb{E}[f(X)]$  (attributions sum to prediction)
- **Symmetry:** Equal contributors get equal attribution
- **Dummy:** Irrelevant features get  $\phi_j = 0$
- **Additivity:** Consistent across model ensembles

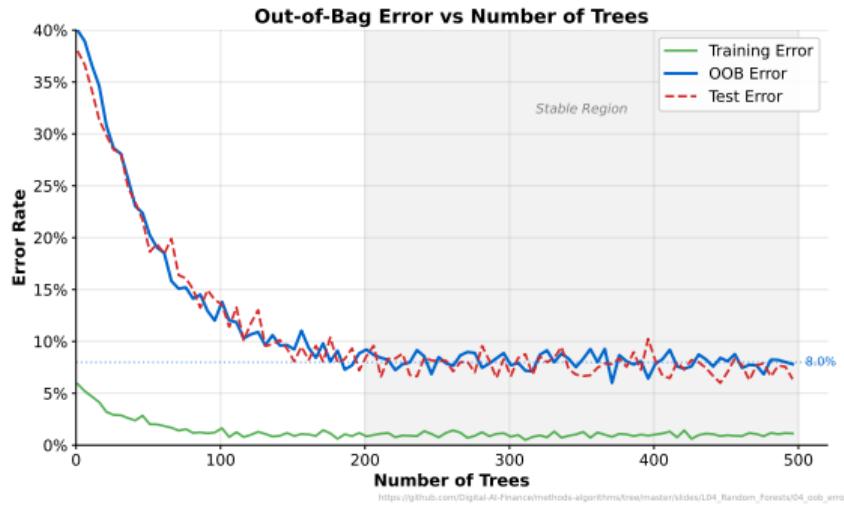
**TreeSHAP:**  $O(TLD^2)$  exact algorithm for tree ensembles vs.  $O(2^p)$  brute force

**Finance:** SHAP provides the *reason codes* required by ECOA/GDPR for adverse action notices in automated lending decisions.

---

Lundberg & Lee (2017). A unified approach to interpreting model predictions. NeurIPS, 4765–4774.

# Out-of-Bag Error



**OOB error:** free cross-validation using samples not in bootstrap

# Out-of-Bag Error: How It Works

For each observation  $i$ :

1. Identify trees where  $i$  was OOB (not in bootstrap sample)
2. Aggregate predictions from only those trees
3. Compare to true label

Benefits:

- No separate validation set needed
- Uses  $\sim 37\%$  of trees per sample
- Approximately unbiased estimate of generalization error

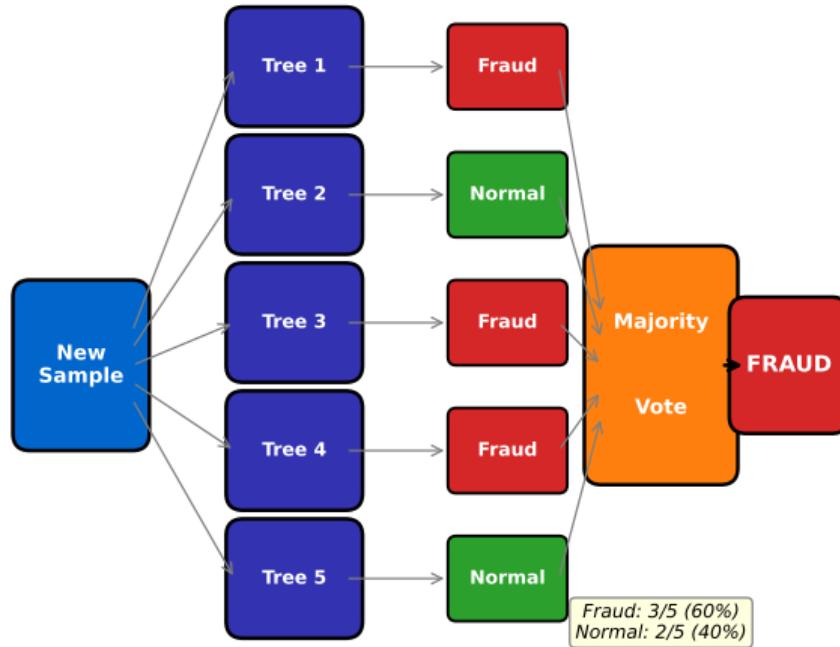
Why OOB Works:

- Each sample is OOB for  $\sim 36.8\%$  of trees (never used in their training)
- Prediction uses only trees that did not see that sample
- Approximates the **0.632 bootstrap estimator**, not leave-one-out CV

---

OOB error approximates the **0.632 bootstrap estimator** (Efron & Tibshirani, 1997)

## Ensemble Voting (Classification)



[https://github.com/Digital-AI-Finance/methods-algorithms/tree/master/slides/L04\\_Random\\_Forests/05\\_ensemble\\_voting](https://github.com/Digital-AI-Finance/methods-algorithms/tree/master/slides/L04_Random_Forests/05_ensemble_voting)

Classification: majority vote. Regression: average prediction

## Part 4: Bias-Variance Decomposition

### Expected Prediction Error:

$$\mathbb{E}[(y - \hat{f}(x))^2] = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

### Single Tree:

- Low bias (can fit complex patterns)
- High variance (sensitive to training data)

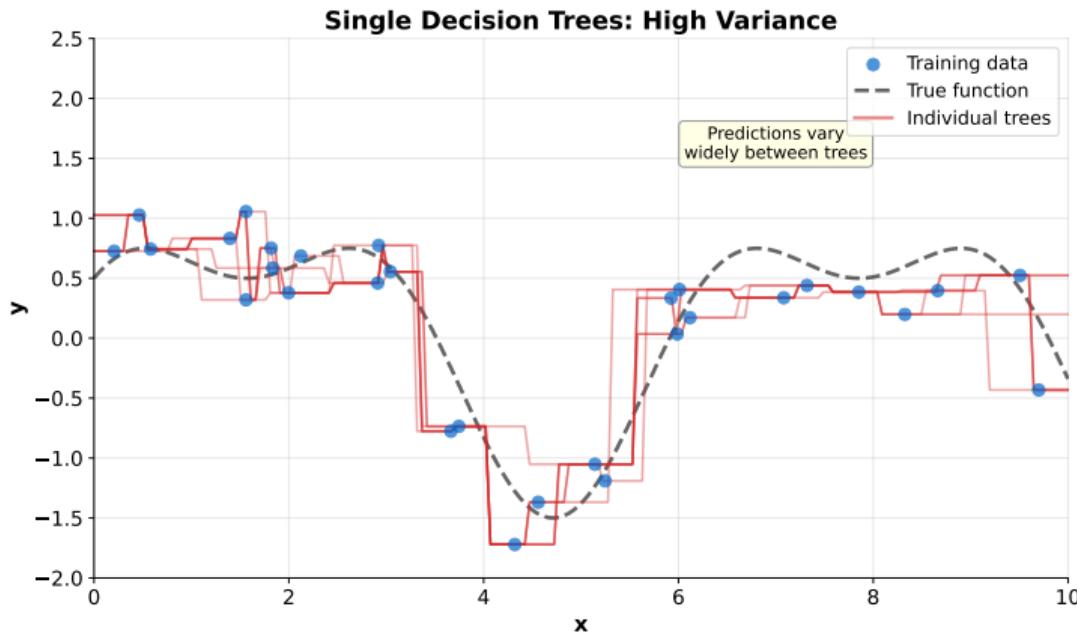
### Random Forest:

- Bias: similar to single tree
- Variance: reduced by  $\approx$  factor of  $1/B$  (with decorrelation)

---

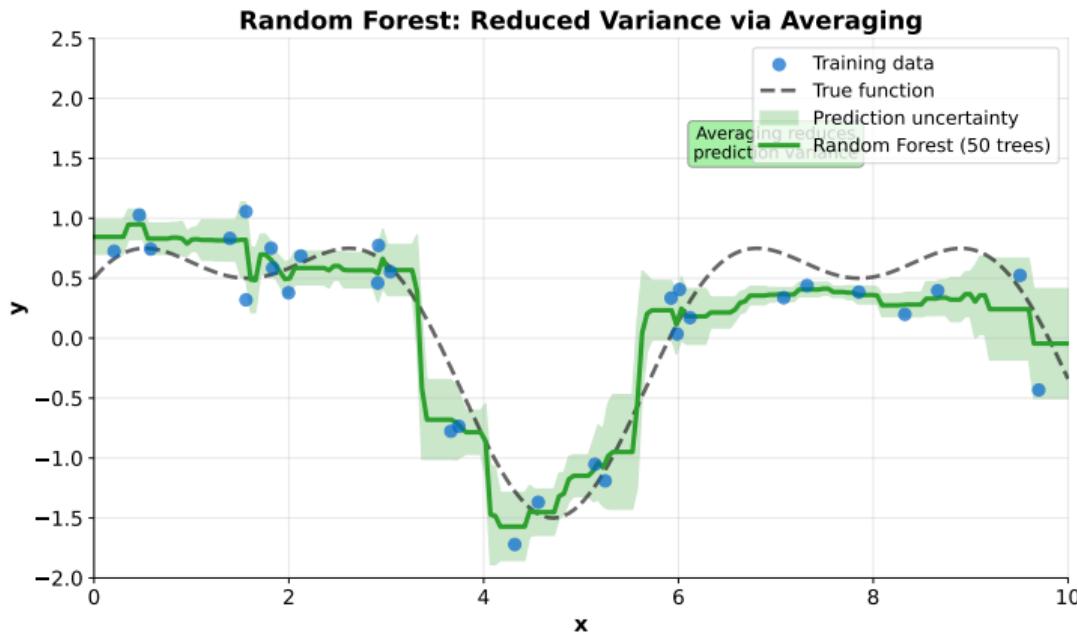
Ensembles reduce variance without increasing bias

# Single Trees: High Variance



Individual trees overfit to their bootstrap samples, producing erratic predictions

# Random Forest: Variance Reduction



Averaging decorrelated trees dramatically reduces prediction variance

## Hyperparameters: Number of Trees

### **n\_estimators** (number of trees):

- More trees = lower variance, never overfits (but higher resource usage)
- Diminishing returns after 100-500 trees
- Cost: linear increase in training/prediction time

### **Guidelines:**

- Start with 100, increase if OOB error still decreasing
- For production: balance accuracy vs. latency
- More trees always better (if time permits)

---

Unlike most hyperparameters, more trees cannot hurt accuracy

## Hyperparameters: Tree Complexity

**max\_depth:** Maximum tree depth

- Deeper = more complex patterns, higher variance
- Default: unlimited (grow full trees)

**min\_samples\_split:** Minimum samples to split

- Higher = simpler trees, more regularization
- Default: 2 (full trees)

**min\_samples\_leaf:** Minimum samples in leaf

- Higher = smoother predictions
- Default: 1 (full trees)

---

Full trees (default) often work well due to bagging's variance reduction

## Hyperparameters: Feature Randomization

**max\_features:** Features considered at each split

- Lower = more decorrelated trees, higher bias
- Higher = less decorrelated, lower bias

**Defaults:**

- Classification:  $\sqrt{p}$  (e.g., 10 features  $\rightarrow 3$ )
- Regression:  $p/3$  (e.g., 30 features  $\rightarrow 10$ )

**Tuning:**

- Try:  $\sqrt{p}$ ,  $\log_2(p)$ ,  $p/3$
- Cross-validate to find optimal

---

Feature randomization is key differentiator from bagged trees

### Advantages:

- Excellent accuracy out-of-the-box
- Handles mixed feature types
- Built-in feature importance
- Robust to outliers and missing values
- Parallelizable (trees independent)

### Limitations:

- Less interpretable than single tree
- Memory intensive (stores all trees)
- Slower prediction than linear models
- Cannot extrapolate beyond training range

---

Random Forests: excellent default choice for tabular data

## Comparison: Random Forest vs. Others

Aspect	RF	Single Tree	Logistic	KNN
Accuracy	High	Medium	Medium	Medium
Interpretability	Medium	High	High	Low
Training Speed	Medium	Fast	Fast	Fast
Feature Importance	Yes	Yes	Yes	No
Non-linear	Yes	Yes	No	Yes

---

RF trades some interpretability for significant accuracy gains

# When to Use Random Forests

## Use When:

- Tabular data with mixed feature types
- Non-linear relationships expected
- Feature importance needed
- Out-of-the-box performance matters

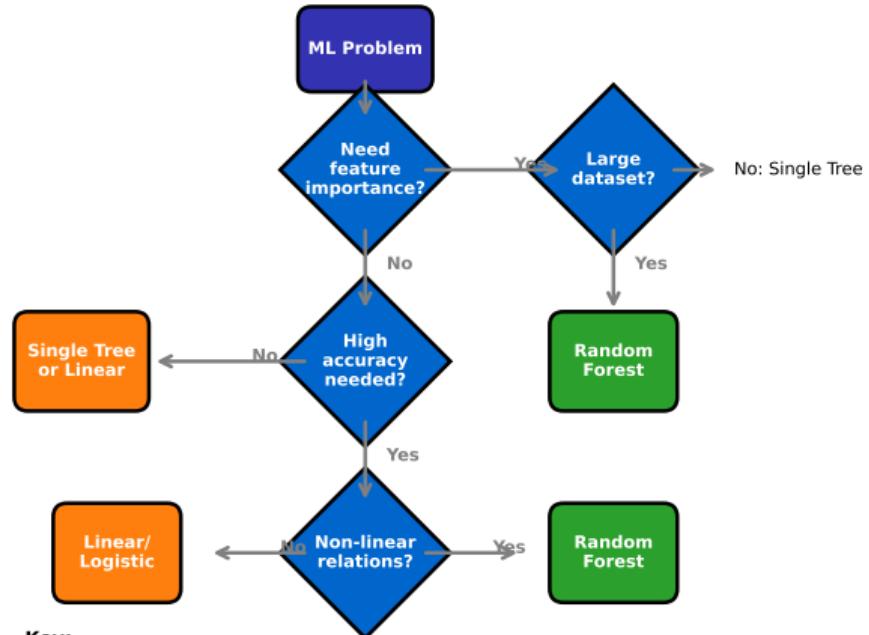
## Consider Alternatives When:

- Need fully interpretable model (use single tree)
- Very high-dimensional sparse data (use linear models)
- Extrapolation required (use parametric models)
- Need fastest prediction (use linear/shallow tree)

---

**Random Forest:** often the first model to try on tabular data

## When to Use Random Forests



**Key:**

Random Forest: Best for accuracy + feature importance

Alternative: When interpretability is paramount

[https://github.com/Digital-AI-Finance/methods-algorithms/tree/master/slides/L04\\_Random\\_Forests/07\\_decision\\_flowchart](https://github.com/Digital-AI-Finance/methods-algorithms/tree/master/slides/L04_Random_Forests/07_decision_flowchart)

**Start with Random Forest; switch if specific constraints require it**

- Bagging reduces **variance** by averaging independent models
- **Boosting** reduces **bias** by sequentially correcting errors
- Core idea: train each model on the *residuals* of the previous

**AdaBoost** (Freund & Schapire, 1997):

1. Initialize weights  $w_i = 1/N$  for all samples
2. For  $t = 1, \dots, T$ :
  - Fit weak learner  $h_t$  to weighted data
  - Compute weighted error:  $\epsilon_t = \sum_{i:h_t(x_i) \neq y_i} w_i$
  - Learner weight:  $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
  - Update:  $w_i \leftarrow w_i \cdot \exp(-\alpha_t y_i h_t(x_i))$ , then normalize
3. Final:  $H(x) = \text{sign}\left(\sum_{t=1}^T \alpha_t h_t(x)\right)$

---

Freund & Schapire (1997). A decision-theoretic generalization of on-line learning. JCSS, 55(1), 119–139.

# Gradient Boosting: A Functional Gradient Descent

**Key insight:** Boosting as gradient descent in function space (Friedman, 2001)

1. Initialize  $F_0(x) = \arg \min_c \sum_{i=1}^N L(y_i, c)$
2. For  $m = 1, \dots, M$ :
  - Compute pseudo-residuals:  $r_{im} = - \left[ \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \right]_{F=F_{m-1}}$
  - Fit base learner  $h_m$  to pseudo-residuals  $\{(x_i, r_{im})\}$
  - Line search:  $\gamma_m = \arg \min_\gamma \sum_{i=1}^N L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i))$
  - Update:  $F_m(x) = F_{m-1}(x) + \nu \cdot \gamma_m h_m(x)$

Learning rate  $\nu \in (0, 1]$  provides regularization via **shrinkage**

For squared loss:  $r_{im} = y_i - F_{m-1}(x_i)$  (literal residuals!)

---

Friedman (2001). Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5), 1189–1232.

**XGBoost** (Chen & Guestrin, 2016) — regularized objective:

$$\mathcal{L}^{(t)} = \sum_{i=1}^N [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t)$$

where  $g_i = \partial_{\hat{y}^{(t-1)}} L(y_i, \hat{y}^{(t-1)})$ ,  $h_i = \partial_{\hat{y}^{(t-1)}}^2 L(y_i, \hat{y}^{(t-1)})$

Regularization:  $\Omega(f) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2$

**Optimal split gain:**

$$\text{Gain} = \frac{1}{2} \left[ \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right] - \gamma$$

**LightGBM:** Leaf-wise growth, histogram binning (Ke et al., 2017)

**CatBoost:** Ordered boosting, native categoricals (Prokhorenkova et al., 2018)

---

Chen & Guestrin (2016). XGBoost: A scalable tree boosting system. KDD, 785–794.

# Boosting in Finance: The Dominant Paradigm

**Credit scoring** — XGBoost/LightGBM dominate competitions (Kaggle):

- Home Credit Default Risk (2018): Top 50 all used gradient boosting
- Handles missing values natively (learned split direction)
- Built-in monotonicity constraints for regulatory compliance

**Fraud detection:**

- LightGBM: Fast enough for real-time scoring (<5ms per transaction)
- Handles extreme class imbalance via `scale_pos_weight`

**Random Forests vs. Boosting — When to choose?**

Criterion	RF	Boosting
Bias-variance	Low variance	Low bias
Overfitting risk	Low	Higher (tune carefully)
Interpretability	Feature importance	SHAP values
Speed (inference)	Parallelizable	Sequential
Kaggle/industry	Baseline	State-of-the-art

See Bentéjac et al. (2021) for comprehensive RF vs. boosting comparison.

### Open the Colab Notebook

- Exercise 1: Train a decision tree on credit data
- Exercise 2: Build a random forest and analyze feature importance
- Exercise 3: Tune hyperparameters with cross-validation

Link: <https://colab.research.google.com/> See course materials for Colab notebook

## Implementation: Complete Example

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score

# Initialize with key hyperparameters
rf = RandomForestClassifier(
    n_estimators=200,          # Number of trees
    max_features='sqrt',      # Features per split
    max_depth=None,           # Full trees (default)
    min_samples_leaf=1,        # Minimum samples in leaf
    class_weight='balanced',   # Handle imbalanced data
    oob_score=True,            # Enable OOB error
    n_jobs=-1,                # Use all CPU cores
    random_state=42            # Reproducibility
)

# Train and evaluate
rf.fit(X_train, y_train)
print(f"OOB Score: {rf.oob_score_:.3f}")
print(f"CV Score: {cross_val_score(rf, X, y, cv=5).mean():.3f}")
```

---

**Key:** `class_weight='balanced'` for imbalanced fraud detection

## Core Concepts:

- Ensemble of decision trees with bootstrap + feature randomization
- Reduces variance while maintaining low bias
- OOB error provides free cross-validation
- **Breiman's insight:** Low tree correlation + high tree strength = best generalization

## Practical Takeaways:

- Excellent default for tabular data
- Feature importance aids interpretation
- More trees = higher accuracy (but more memory/latency)
- Hyperparameter tuning often optional, but recommended for imbalanced data

---

Next: PCA and t-SNE for dimensionality reduction

# References

## Textbooks:

- James et al. (2021). *ISLR*, Chapter 8: Tree-Based Methods
- Hastie et al. (2009). *ESL*, Chapter 15: Random Forests

## Original Papers:

- Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1), 5-32.
- Breiman, L. (1996). Bagging Predictors. *Machine Learning*, 24(2), 123-140.
- Friedman, J.H. (2001). Greedy function approximation. *Annals of Statistics*, 29(5).
- Chen & Guestrin (2016). XGBoost. *KDD*, 785–794.
- Lundberg & Lee (2017). SHAP. *NeurIPS*, 4765–4774.

---

Breiman's 2001 paper: one of the most cited in ML