L04: Random Forests

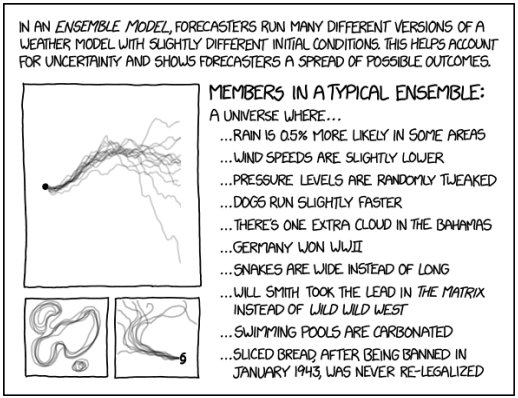
Deep Dive: Theory, Implementation, and Applications

Methods and Algorithms

Spring 2026

Outline

The Ensemble Approach



XKCD #1885 by Randall Munroe (CC BY-NC 2.5)

Learning Objectives

After this session, you will be able to:

1. **Derive** the variance reduction formula and analyze the role of tree correlation in ensemble performance
2. **Evaluate** Random Forest vs. gradient boosting using bias-variance tradeoff analysis
3. **Analyze** feature importance using MDI, permutation importance, and SHAP values with statistical inference
4. **Critique** ensemble methods for regulatory compliance in financial applications (ECOA, GDPR)

Bloom's Level 4–5: Analyze, Evaluate, Critique

From Rules to Trees

Decision trees encode if-then-else rules as a hierarchical structure:

- Each **internal node** splits data on a feature threshold: $x_j \leq s$
- Each **leaf** contains a prediction (class label or numeric value)
- A path from root to leaf represents one decision rule

Key Questions:

- How to choose the **best split** at each node?
- When to **stop splitting**?
- How to **prevent overfitting**?

Decision trees: the fundamental building block of Random Forests and gradient boosting

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 at the node.

Properties:

- $G = 0$: pure node (all samples belong to one class)
- $G = 0.5$: maximum impurity for binary classification
- General maximum: $G_{\max} = 1 - 1/K$ for K classes
- **Lower Gini = better split**

Gini impurity: the probability of misclassifying a randomly chosen sample

Splitting Criteria: Information Gain

Entropy measures disorder at a node:

$$H = -\sum_{k=1}^{K} p_k \log_2(p_k) \qquad \text{Convention: } 0 \cdot \log_2(0) = 0$$

Information Gain from a split:

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

Gini vs. Entropy:

- Gini: faster to compute, tends to isolate the most frequent class
- Entropy: produces more balanced trees, slightly slower (log computation)
- In practice: similar performance—choice rarely matters

Both criteria aim to create the purest possible child nodes

Regression Trees: MSE Criterion

For regression, the splitting criterion uses Mean Squared Error:

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

Split quality is measured by the reduction in MSE:

$$\Delta\text{MSE} = \text{MSE}(\text{parent}) - \frac{n_L}{n}\text{MSE}(\text{left}) - \frac{n_R}{n}\text{MSE}(\text{right})$$

Key properties:

- Leaf prediction = mean of samples in the leaf: $\hat{y} = \bar{y}_{\text{leaf}}$
- Equivalent to minimizing within-node variance
- Alternative: Mean Absolute Error (more robust to outliers)

Trees handle both classification and regression via appropriate impurity measures

CART: Recursive Partitioning Algorithm

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: Overfitting and Pruning

The Overfitting Problem:

- Fully grown trees can achieve 100% training accuracy
- But generalize poorly—they memorize noise in the data

Pre-pruning (stop early):

- `max_depth`, `min_samples_split`, `min_samples_leaf`
- Fast but may stop too early (miss useful splits deeper in the tree)

Post-pruning (grow then cut):

- Cost-complexity pruning: minimize $\sum_{m=1}^{|T|} R_m + \alpha |T|$
- α controls the tradeoff: larger α = simpler tree
- Select α via cross-validation

Ensembles largely eliminate the need for careful pruning of individual trees

Why Ensembles?

The Problem with Single Trees:

- **High variance**: small changes in data produce very different trees
- **Overfitting**: deep trees memorize noise
- **Instability**: a single outlier can change the entire tree structure

The Ensemble Solution:

- Train **multiple diverse models** on different data subsets
- **Combine predictions** (vote or average)
- Reduce variance while maintaining low bias

"Wisdom of crowds": the aggregate judgment of many weak learners can outperform any single strong learner.

Condorcet's Jury Theorem (1785): majority vote accuracy increases with number of independent voters

Bootstrap Sampling

Bootstrap: sample n observations *with replacement* from original data.

Key Properties:

- Each bootstrap sample has the same size n as the original
- Expected unique observations: $\approx 63.2\%$ (as $n \to \infty$, converges to $1 - 1/e$)
- Remaining $\sim 37\%$: **out-of-bag (OOB)** samples—free validation set

Why Bootstrap Creates Diversity:

- Each tree sees a different subset of the data
- Trees disagree on noisy observations $\to$ errors cancel out
- OOB samples enable internal error estimation

Efron (1979). Bootstrap methods: another look at the jackknife. Annals of Statistics, 7(1), 1–26.

Bagging: Mathematical Foundation

Variance Reduction by Averaging
For B predictions, each with variance σ^2 and pairwise correlation ρ:

Full derivation:

$$\text{Var}\left(\frac{1}{B}\sum_{b=1}^{B}\hat{f}_b\right) = \frac{1}{B^2}\left[B\sigma^2 + B(B-1)\rho\sigma^2\right]$$

$$= \frac{\sigma^2}{B} + \frac{B-1}{B}\rho\sigma^2 = \rho\sigma^2 + \frac{(1-\rho)\sigma^2}{B}$$

Key insight:

- If independent ($\rho = 0$): $\text{Var} = \sigma^2/B$ — perfect variance reduction
- If perfectly correlated ($\rho = 1$): $\text{Var} = \sigma^2$ — no benefit at all
- **Reduce ρ between trees to maximize ensemble benefit**

This formula is the theoretical foundation for why Random Forests add feature randomization

Why Bagging Reduces Variance (Proof Sketch)

Setup: B trees with predictions $\hat{f}_1, \ldots, \hat{f}_B$, each with $\text{Var}(\hat{f}_b) = \sigma^2$ and $\text{Cov}(\hat{f}_i, \hat{f}_j) = \rho\sigma^2$ for $i \neq j$.

Step 1: Variance of the average:

$$\text{Var}(\bar{f}) = \text{Var}\left(\frac{1}{B}\sum_{b=1}^{B}\hat{f}_b\right) = \frac{1}{B^2}\sum_i\sum_j \text{Cov}(\hat{f}_i, \hat{f}_j)$$

Step 2: Separate diagonal and off-diagonal terms:

$$= \frac{1}{B^2}\left[\underbrace{B \cdot \sigma^2}_{\text{diagonal}} + \underbrace{B(B-1) \cdot \rho\sigma^2}_{\text{off-diagonal}}\right]$$

Step 3: As $B \to \infty$:

$$\text{Var}(\bar{f}) \to \rho\sigma^2$$

Conclusion: Correlation ρ is the **limiting factor**—even infinite trees cannot reduce variance below $\rho\sigma^2$.

This motivates feature randomization: the key innovation from bagging to Random Forests

Random Forest: Two Sources of Randomness

Source 1: Bootstrap Sampling (inherited from bagging)

- Each tree trained on a different bootstrap sample
- Creates data-level diversity

Source 2: Feature Randomization (the key RF innovation)

- At each split, consider only a random subset of m features
- Classification default: $m = \sqrt{p}$
- Regression default: $m = p/3$
- **Decorrelates trees** beyond what bootstrap alone achieves

Historical origin:

- Ho (1995): random subspace method (feature sampling only)
- Breiman (2001): combined with bagging $\rightarrow$ Random Forests

---

Feature randomization is what separates Random Forests from simple bagged trees

Random Forest Algorithm

Require: Training data $\{(x_i, y_i)\}_{i=1}^{n}$, number of trees B, features per split m

for $b = 1$ to B **do**
  Draw bootstrap sample $\mathcal{D}_b$ of size n (with replacement)
  Grow tree T_b on $\mathcal{D}_b$:
    At each node, select m features randomly (without replacement)
    Find best split among m features using impurity criterion
    Split until stopping criterion (or node is pure)
end for
Prediction:
  Classification: $\hat{y} = \text{majority vote}\{T_b(x)\}_{b=1}^{B}$
  Regression: $\hat{y} = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$

Defaults: $m = \sqrt{p}$ (classification), $m = p/3$ (regression), $B = 100$–500

Breiman (2001). Random Forests. Machine Learning, 45(1), 5–32.

Hyperparameters: Number of Trees

`n_estimators` (number of trees B):

- More trees = lower variance; **Random Forests never overfit** by adding trees
- Diminishing returns: most gains occur before $B = 100$–500
- Cost: training and prediction time scale linearly with B

Practical Guidelines:

- Start with $B = 100$; increase if OOB error is still decreasing
- For production: balance accuracy vs. latency requirements
- Monitor OOB convergence to find the "sweet spot"

Unlike most hyperparameters, more trees cannot hurt accuracy—only cost and latency

Hyperparameters: Tree Complexity

`max_depth`: Maximum tree depth

- Deeper trees capture more complex patterns but increase variance
- Default: unlimited (grow full trees)—works well due to bagging

`min_samples_split`: Minimum samples required to split a node

- Higher values produce simpler, more regularized trees
- Default: 2 (allow splitting until leaves are pure)

`min_samples_leaf`: Minimum samples in each leaf node

- Higher values create smoother predictions
- Default: 1 (fully grown trees)

Full trees (default) often work well because bagging handles variance reduction

Hyperparameters: Feature Randomization

`max_features`: Number of features considered at each split.

- Lower m = more decorrelated trees, but higher individual tree bias
- Higher m = stronger individual trees, but higher correlation

Default Values:

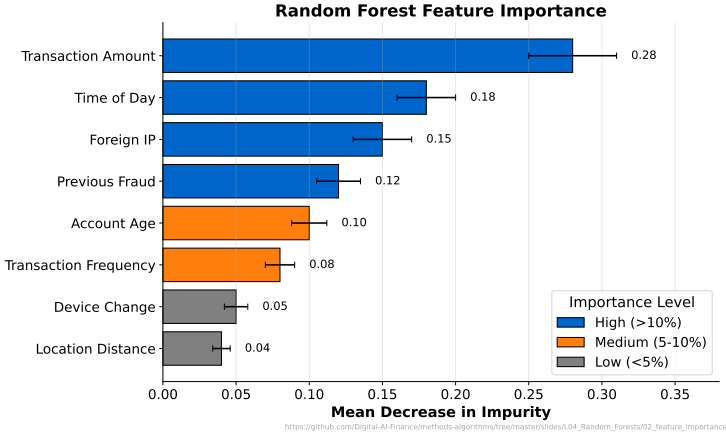
- Classification: $m = \sqrt{p}$ (e.g., 100 features $\rightarrow$ 10 per split)
- Regression: $m = p/3$ (e.g., 30 features $\rightarrow$ 10 per split)

Tuning Strategy:

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

Feature randomization is the key differentiator: it controls ρ in the variance formula

Feature Importance



Mean Decrease in Impurity: total impurity reduction from all splits on each feature

Mean Decrease in Impurity (MDI)

Formula:

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

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

Advantages:

- Fast to compute—comes free from the training process
- Available via `rf.feature_importances_` in scikit-learn

Limitations:

- **Bias toward high-cardinality features** (more possible splits)
- Inflated importance for correlated features

MDI is a useful first pass but should be validated with permutation importance

Permutation Importance

Idea: Randomly shuffle feature j, measure how much accuracy drops.

1. Compute baseline score S on validation data
2. Permute feature j's values (break feature-target association)
3. Recompute score S_j^π on permuted data
4. Importance $= S - S_j^\pi$ (larger drop = more important)

Advantages over MDI:

- Model-agnostic—works with any classifier
- Less biased toward high-cardinality features
- Uses validation data, so reflects true predictive contribution

Limitation: Slower—requires re-evaluation for each feature.

Breiman (2001) proposed permutation importance as the preferred method

Statistical Inference for Feature Importance

Problem: Is a feature's importance score statistically significant?

Permutation Testing (Altmann et al., 2010):

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

Bootstrap Confidence Intervals:

- Retrain RF multiple times, collect 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—point estimates of importance can be misleading

SHAP Values: Game-Theoretic Feature Importance

SHapley Additive exPlanations (Lundberg & Lee, 2017):

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

Axiomatic Properties (unique among attribution methods):

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

TreeSHAP: Exact computation in $O(TLD^2)$ vs. brute force $O(2^p)$

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.

Bias-Variance Decomposition

Expected Prediction Error:

$$\mathbb{E}\Big[(y - \hat{f}(x))^2\Big] = \underbrace{\text{Bias}^2(\hat{f})}_{\text{systematic error}} + \underbrace{\text{Var}(\hat{f})}_{\text{sensitivity to data}} + \underbrace{\sigma_\varepsilon^2}_{\text{irreducible}}$$

Single Decision Tree:

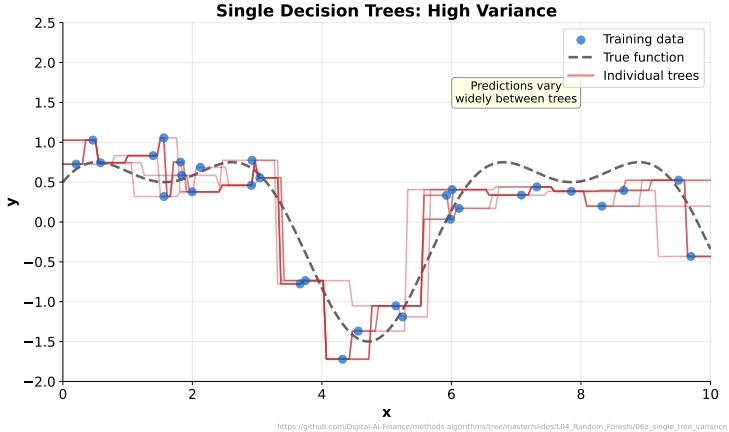
- Low bias (can fit arbitrarily complex patterns)
- **High variance** (small data changes $\rightarrow$ very different trees)

Random Forest:

- Bias: approximately the same as a single tree
- Variance: reduced by factor $\approx \rho + (1 - \rho)/B$
- Net effect: **substantially lower total error**

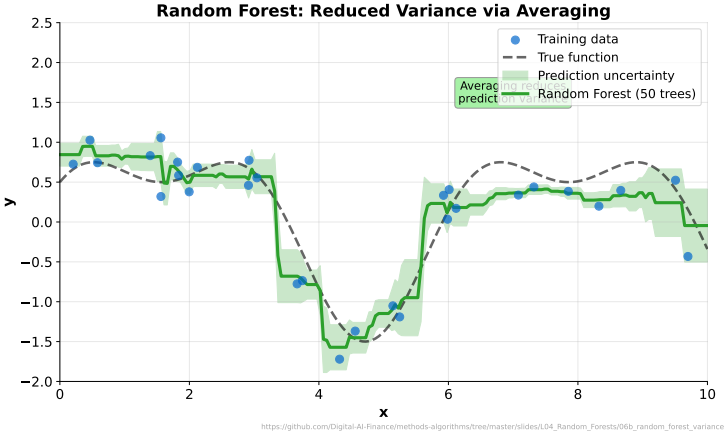
Ensembles exploit the bias-variance tradeoff: reduce variance without increasing bias

Single Trees: High Variance



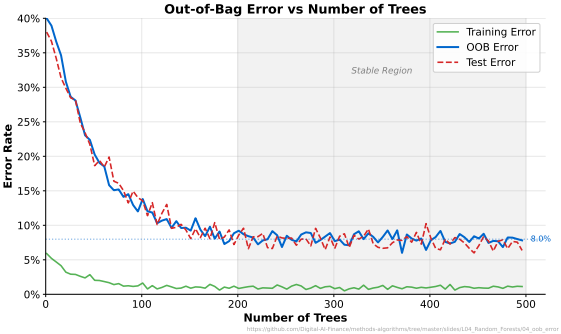
Individual trees overfit to their bootstrap samples, producing erratic predictions

Random Forest: Variance Reduction



Averaging decorrelated trees dramatically reduces prediction variance

Out-of-Bag Error Estimation



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

OOB Error: How It Works

For each observation i:

1. Identify trees where observation i was **out-of-bag** (not in the bootstrap sample)
2. Aggregate predictions from only those OOB trees
3. Compare aggregated prediction to true label y_i

Properties:

- Each sample is OOB for $\sim 36.8\%$ of trees (never used in their training)
- No separate validation set needed—uses all data for both training and evaluation
- Approximates the **0.632 bootstrap estimator** (Efron & Tibshirani, 1997)

Practical Use:

- Enable via `oob_score=True` in scikit-learn
- Monitor convergence: plot OOB error vs. number of trees

OOB error is approximately unbiased and avoids the cost of k-fold cross-validation

From Bagging to Boosting

Two Complementary Strategies:

Property	Bagging (RF)	Boosting
Training	Parallel	Sequential
Target	Reduce variance	Reduce bias
Base learners	Full trees	Shallow trees (stumps)
Combination	Average / vote	Weighted sum
Overfit risk	Low	Higher (tune carefully)

Core idea of boosting: Train each new model on the *residuals* (errors) of the previous ensemble—sequentially correcting mistakes.

Bagging and boosting are complementary: one reduces variance, the other reduces bias

AdaBoost Algorithm

Require: Training data $\{(x_i, y_i)\}_{i=1}^{N}$ with $y_i \in \{-1, +1\}$

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

Key insight: Misclassified samples get higher weights $\rightarrow$ next learner focuses on hard cases.

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

Gradient Boosting: 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 $m = 1, \ldots, M$:
   - 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 $\{(x_i, r_{im})\}$
   - Line search: $\gamma_m = \arg\min_\gamma \sum_i L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i))$
   - Update: $F_m(x) = F_{m-1}(x) + \nu \cdot \gamma_m \, h_m(x)$

Shrinkage: Learning rate $\nu \in (0, 1]$ regularizes by slowing learning.
For squared loss: $r_{im} = y_i - F_{m-1}(x_i)$ (literal residuals!)

Friedman (2001). Greedy function approximation. Annals of Statistics, 29(5), 1189–1232.

XGBoost, LightGBM, CatBoost: Industrial Boosting

XGBoost (Chen & Guestrin, 2016)—regularized objective with Taylor expansion:

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

where $g_i = \partial L / \partial \hat{y}^{(t-1)}$, $\quad h_i = \partial^2 L / \partial (\hat{y}^{(t-1)})^2$

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:

- Home Credit Default Risk (Kaggle 2018): top 50 all used gradient boosting
- Built-in monotonicity constraints for regulatory compliance

Fraud detection—LightGBM fast enough for real-time (<5ms):

- Handles extreme class imbalance via `scale_pos_weight`
- Histogram binning enables deployment on streaming data

Criterion	Random Forest	Boosting
Bias-variance	Low variance	Low bias
Overfitting risk	Low	Higher (tune carefully)
Interpretability	Feature importance	SHAP values
Kaggle/industry	Baseline	State-of-the-art

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

CRITICAL: Class Imbalance in Fraud Detection

The Reality of Fraud Data:

- Fraud is typically $<1\%$ of transactions (100:1 imbalance)
- A model predicting "not fraud" for ALL transactions achieves 99% accuracy!
- **Accuracy is the WRONG metric** for imbalanced classification

Correct Evaluation Metrics:

- **Precision**: Of flagged transactions, how many are actually fraud?
- **Recall**: Of actual fraud cases, how many did we catch?
- **F1 Score**: Harmonic mean $= 2 \cdot \frac{\text{Prec} \cdot \text{Rec}}{\text{Prec} + \text{Rec}}$
- **AUC-PR**: Area under Precision-Recall curve (preferred over ROC-AUC)

NEVER use accuracy for imbalanced classification—it is dangerously misleading

Handling Class Imbalance

1. **Class Weighting** (scikit-learn):
   - `class_weight='balanced'`: auto-adjust by inverse class frequency
   - `class_weight='balanced_subsample'`: per-tree balancing in RF
2. **Resampling Techniques:**
   - **SMOTE**: Synthetic Minority Oversampling TEchnique—generates synthetic minority samples
   - **Undersampling**: Reduce majority class (risk: losing information)
   - **Combination**: SMOTE + Tomek links for cleaner boundaries
3. **Threshold and Cost-based Tuning:**
   - Default threshold 0.5 is rarely optimal for imbalanced data
   - Choose threshold based on business costs: FN cost $\gg$ FP cost for fraud

Always check class distribution BEFORE training any classifier!

Credit Risk: Feature Importance in Practice

Worked Example—Top features in credit scoring:

1. **Payment history** (35%)—most predictive of default
2. **Debt-to-income ratio** (25%)—capacity to repay
3. **Credit utilization** (20%)—current leverage
4. **Length of credit history** (10%)—track record

From Importance to Business Decisions:

- SHAP waterfall plots show *why* each applicant was approved/denied
- Regulatory requirement: provide **adverse action reasons** (ECOA)
- Top 3–4 SHAP features become the reason codes on denial letters

Feature importance bridges the gap between model predictions and actionable business decisions

Implementation: scikit-learn 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'` **is critical for imbalanced fraud detection**

Comparison: Random Forest vs. Alternatives

Aspect	RF	Single Tree	Logistic	KNN	XGBoost
Accuracy	High	Medium	Medium	Medium	High
Interpretability	Medium	High	High	Low	Low
Training Speed	Medium	Fast	Fast	Fast	Slow
Feature Import.	Yes	Yes	Coef.	No	Yes
Non-linear	Yes	Yes	No	Yes	Yes
Overfit Risk	Low	High	Low	Medium	Medium

When to choose RF: tabular data, mixed feature types, need feature importance, want robust out-of-the-box performance with minimal tuning.

RF trades some interpretability for significant accuracy and robustness gains

Hands-on Exercises

Exercise 1 (Foundations): Train a single decision tree on credit data. Visualize the tree structure. Compute Gini impurity manually for the root split.

Exercise 2 (Random Forest): Build a Random Forest with 200 trees. Compare OOB error to 5-fold CV. Analyze feature importance using both MDI and permutation methods.

Exercise 3 (Advanced): Tune `max_features` and `n_estimators` via grid search. Apply to imbalanced fraud detection with `class_weight='balanced'`. Generate SHAP summary plots.

Exercises progress from Bloom's Level 2 (understand) to Level 5 (evaluate)

Summary: Random Forests Deep Dive

Core Concepts:

- Ensemble of decision trees with **bootstrap sampling + feature randomization**
- Variance reduction: $\text{Var}(\bar{f}) = \rho\sigma^2 + (1-\rho)\sigma^2/B$
- OOB error provides "free" cross-validation
- Boosting complements bagging: sequential bias reduction vs. parallel variance reduction

Practical Takeaways:

- RF is the best "first model" for tabular data—robust and minimal tuning
- SHAP values enable regulatory-compliant explanations (ECOA, GDPR)
- For imbalanced data: use `class_weight='balanced'`, evaluate with AUC-PR
- XGBoost/LightGBM often outperform RF with careful tuning

Next session: PCA and t-SNE for dimensionality reduction

Closing Thought

"Stir the pile of decision trees until the validation loss converges."

— Adapted from XKCD #1838 "Machine Learning"

Appendix

Advanced Topics

Topics: Gini derivation, bagging proof, bootstrap 63.2% rule, consistency, XGBoost derivation, AdaBoost-exponential loss, SHAP axioms, complexity analysis

Appendix: Gini Impurity from First Principles

Setup: A node contains samples from K classes with proportions $p_1, \ldots, p_K$.

Derivation (expected misclassification probability):

1. Pick a random sample: class k with probability p_k
2. Classify it using the class distribution: assign class j with probability p_j
3. Misclassification probability for a class-k sample: $1 - p_k$

Expected misclassification rate:

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

Connection to variance: For binary case $(p, 1-p)$:

$$G = 2p(1-p) = 2\,\mathrm{Var}(\mathrm{Bernoulli}(p))$$

Gini impurity is proportional to the variance of the class indicator variable.

Gini impurity has a natural interpretation as expected misclassification under random labeling

Appendix: Bagging Variance Reduction—Full Proof

Given: B predictions $\hat{f}_1, \ldots, \hat{f}_B$ with $\text{Var}(\hat{f}_b) = \sigma^2$, $\text{Corr}(\hat{f}_i, \hat{f}_j) = \rho$ for $i \neq j$.

Step 1: $\text{Var}(\bar{f}) = \text{Var}\left(\frac{1}{B}\sum_{b=1}^{B} \hat{f}_b\right) = \frac{1}{B^2}\sum_{i=1}^{B}\sum_{j=1}^{B} \text{Cov}(\hat{f}_i, \hat{f}_j)$

Step 2: Separate terms:

$$= \frac{1}{B^2}\left[\sum_{i=1}^{B} \text{Var}(\hat{f}_i) + \sum_{i \neq j} \text{Cov}(\hat{f}_i, \hat{f}_j)\right] = \frac{1}{B^2}\left[B\sigma^2 + B(B-1)\rho\sigma^2\right]$$

Step 3: Simplify:

$$= \frac{\sigma^2}{B} + \frac{B-1}{B}\rho\sigma^2 = \frac{\sigma^2}{B}(1-\rho) + \rho\sigma^2$$

Step 4: Limiting behavior as $B \to \infty$: $\quad \text{Var}(\bar{f}) \to \rho\sigma^2$

Conclusion: Even with infinitely many trees, variance cannot go below $\rho\sigma^2$. This is why RF adds feature randomization: to reduce ρ.

The irreducible floor $\rho\sigma^2$ is the fundamental reason Random Forests randomize feature subsets

Appendix: Bootstrap—The 63.2% Rule

Claim: In a bootstrap sample of size n, approximately 63.2% of observations are unique.

Proof:

1. Probability that observation i is *not* selected in one draw: $1 - \frac{1}{n}$
2. Probability not selected in any of n draws: $\left(1 - \frac{1}{n}\right)^n$
3. Taking the limit as $n \to \infty$:

$$\lim_{n\to\infty} \left(1 - \frac{1}{n}\right)^n = e^{-1} \approx 0.368$$

Taylor expansion: $\ln\left(1 - \frac{1}{n}\right) = -\frac{1}{n} - \frac{1}{2n^2} - \cdots \approx -\frac{1}{n}$

Therefore: $\left(1 - \frac{1}{n}\right)^n \approx e^{n\cdot(-1/n)} = e^{-1}$

$P(\text{selected at least once}) = 1 - e^{-1} \approx 0.632$

Practical implication: Each tree's OOB set contains $\sim$37% of the training data.

For $n = 1000$: expected unique samples $= 632$, OOB samples $= 368$

Appendix: Random Forest Consistency

Question: Does a Random Forest converge to the true function as $n \to \infty$?

Breiman (2001): Strong Law of Large Numbers for Forests

- As $B \to \infty$, the RF generalization error converges a.s.
- The forest does not overfit by adding trees: $\text{PE}^* \leq \bar{\rho} \frac{s^2}{(E_\theta[s])^2}$
- where $\bar{\rho}$ = mean correlation, s = margin strength

Scornet, Biau & Vert (2015): Formal consistency proof

- Proved: Breiman's RF is consistent for additive regression models
- Key condition: $\min(n_{\min}, p) \to \infty$ and $k/p \to 0$
- Result: $\mathbb{E}[\hat{f}_n(x)] \to f(x)$ as $n \to \infty$

Wager & Athey (2018): Asymptotic normality and confidence intervals for RF predictions—enables statistical inference with forests.

Scornet et al. (2015). Consistency of random forests. Annals of Statistics, 43(4), 1716–1741.

Appendix: XGBoost Objective Derivation

Start: Loss at iteration t with second-order Taylor expansion:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{N} L\Big(y_i,\, \hat{y}_i^{(t-1)} + f_t(x_i)\Big) + \Omega(f_t)$$

$$\approx \sum_{i=1}^{N} \Big[L(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \tfrac{1}{2} h_i f_t^2(x_i)\Big] + \Omega(f_t)$$

where $g_i = \partial L / \partial \hat{y}^{(t-1)}$, $h_i = \partial^2 L / \partial (\hat{y}^{(t-1)})^2$.

Optimal leaf weights: For leaf j with samples I_j:

$$w_j^* = -\frac{G_j}{H_j + \lambda}, \quad G_j = \sum_{i \in I_j} g_i, \quad H_j = \sum_{i \in I_j} h_i$$

Optimal split gain:

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

The second-order approximation enables efficient exact greedy split finding

Appendix: AdaBoost as Exponential Loss Minimization

Claim: AdaBoost minimizes the exponential loss function:

$$L = \sum_{i=1}^{N} \exp(-y_i F(x_i)), \quad F(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$

Proof sketch (forward stagewise additive modeling):

1. At step t, minimize $L_t = \sum_i w_i^{(t)} \exp(-y_i \alpha_t h_t(x_i))$ where $w_i^{(t)} = \exp(-y_i F_{t-1}(x_i))$
2. Optimal h_t: minimizes weighted classification error $\epsilon_t = \sum_{i:h_t(x_i) \neq y_i} w_i$
3. Optimal α_t: $\frac{\partial L_t}{\partial \alpha_t} = 0 \Rightarrow \alpha_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$

Connection to logistic regression:

- Exponential loss $\rightarrow$ logistic loss: yields LogitBoost
- Both are proper scoring rules; logistic loss is more robust to outliers

Friedman, Hastie & Tibshirani (2000). Additive logistic regression. Annals of Statistics, 28(2), 337–407.

Appendix: SHAP Axioms and TreeSHAP Complexity

Shapley's Axioms (unique attribution satisfying all four):

1. **Efficiency**: $\sum_j \phi_j = f(x) - \mathbb{E}[f(X)]$ (attributions explain the full prediction)
2. **Symmetry**: If $f(S \cup \{i\}) = f(S \cup \{j\})$ for all S, then $\phi_i = \phi_j$
3. **Dummy**: If feature j never changes f, then $\phi_j = 0$
4. **Additivity**: For $f = f_1 + f_2$: $\phi_j(f) = \phi_j(f_1) + \phi_j(f_2)$

Complexity Comparison:

Method	Complexity	Exact?
Brute force Shapley	$O(2^p)$	Yes
KernelSHAP	$O(p \cdot M)$ (M samples)	Approximate
TreeSHAP	$O(T \cdot L \cdot D^2)$	Yes

where T = trees, L = max leaves, D = max depth, p = features.

Lundberg et al. (2020). From local explanations to global understanding with TreeSHAP. Nature MI, 2, 56–67.

Appendix: Computational Complexity

Training Complexity:

$$O(B \cdot n \cdot m \cdot \log n)$$

where B = trees, n = samples, m = features per split ($\sqrt{p}$ or $p/3$).

Prediction Complexity:

$$O(B \cdot \text{depth}) \quad \text{per sample}$$

Memory:

$$O(B \cdot \text{nodes per tree}) \approx O(B \cdot 2n) \quad \text{for full trees}$$

Parallelization:

- Trees are **embarrassingly parallel**—train on separate cores
- `n_jobs=-1` in scikit-learn uses all available CPU cores
- Prediction also parallelizable across trees (and across samples)

RF scales linearly with B and n—practical for datasets up to millions of samples

Appendix: References and Further Reading

Textbooks:

- James et al. (2021). *An Introduction to Statistical Learning*, Ch. 8: Tree-Based Methods
- Hastie et al. (2009). *Elements of Statistical Learning*, Ch. 15: Random Forests

Original Papers:

- Breiman (2001). Random Forests. *Machine Learning*, 45(1), 5–32.
- Friedman (2001). Greedy function approximation. *Annals of Statistics*, 29(5), 1189–1232.
- Chen & Guestrin (2016). XGBoost. *KDD*, 785–794.
- Lundberg & Lee (2017). SHAP. *NeurIPS*, 4765–4774.

Software:

- `scikit-learn`: `https://scikit-learn.org`
- `xgboost`: `https://xgboost.readthedocs.io`
- `shap`: `https://shap.readthedocs.io`

Recommended starting point: ISLR Chapter 8 for intuition, then Breiman (2001) for theory