L04: Random Forests

Deep Dive: Theory, Implementation, and Applications

Methods and Algorithms – MSc Data Science

Part 1: Decision Tree Foundations

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 impurity for binary classification
- Lower Gini = better split

Gini impurity: probability of misclassifying a random sample

Splitting Criteria: Information Gain

Entropy measures disorder:

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

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

Decision Tree Algorithm

Recursive Partitioning:

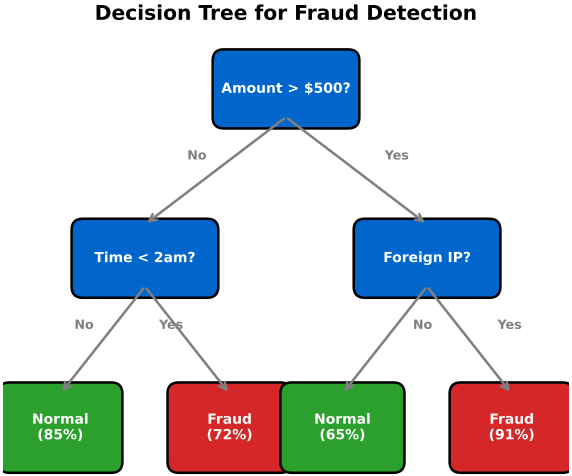
1. Start with all samples at root
2. For each feature and threshold:
   - Calculate impurity reduction
   - Select split with maximum reduction
3. Create child nodes with split samples
4. Recurse until stopping criterion met

Stopping Criteria:

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

Greedy algorithm: locally optimal splits at each step

Decision Tree: Fraud Detection Example



Each path through the tree represents a fraud detection rule

Part 2: Why Ensembles?

Problem with Single Trees:

- High variance: small data changes $\rightarrow$ 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:

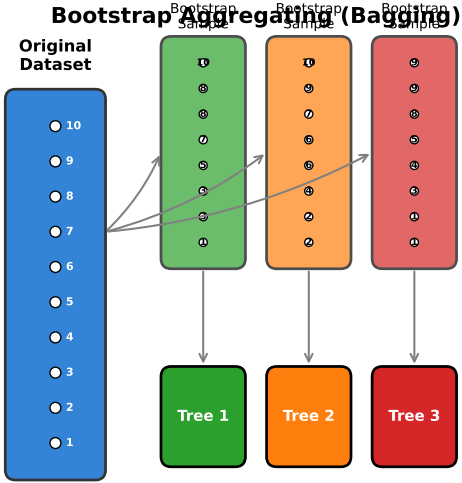
- Each sample: same size as original (n observations)
- Expected unique samples: $\approx 63.2\%$ (probability $1 - (1 - 1/n)^n$)
- Remaining $\sim 37\%$: out-of-bag (OOB) samples

Effect:

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

Bootstrap: key ingredient for reducing variance through aggregation

Bagging Visualization



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

Bootstrap Aggregating: train on random samples, aggregate predictions

Bagging: Mathematical Foundation

Variance Reduction by Averaging
For B independent predictions with variance σ^2:

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

With correlation ρ:

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

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: Breiman's key innovation over bagging

Random Forest Algorithm

Training:

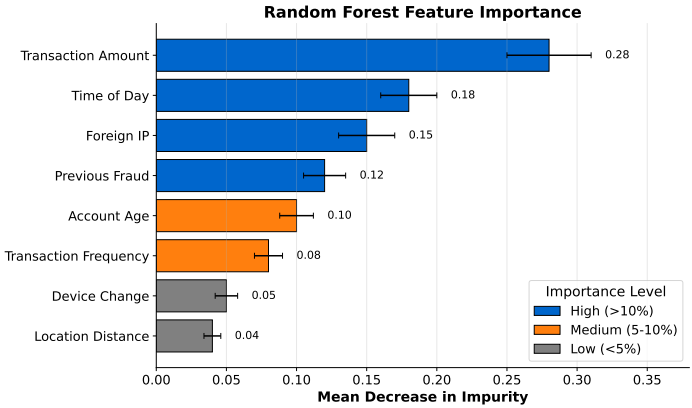
1. For $b = 1$ to B trees:
   - Draw bootstrap sample of size n
   - Grow tree:
     - At each node, select m features randomly
     - Find best split among m features
     - Split until stopping criterion

Prediction:

- Classification: majority vote across trees
- Regression: average predictions

Typical: 100-500 trees, but more trees never hurts (just slower)

Feature Importance



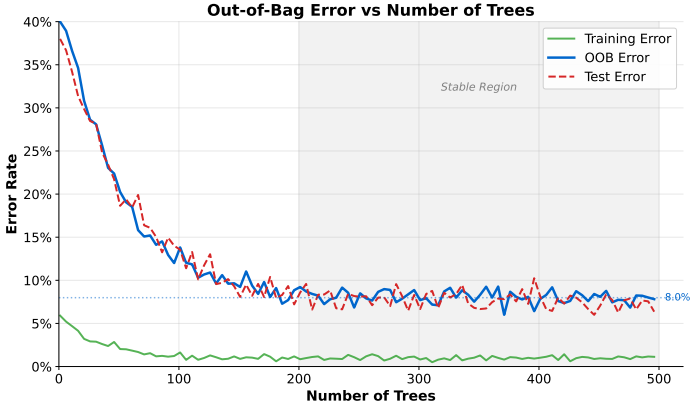
Mean Decrease in Impurity: sum of impurity reductions from splits on feature

Feature Importance Methods

1. **Mean Decrease in Impurity (MDI):**
   - Sum of Gini/entropy reductions from splits on feature
   - 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

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:

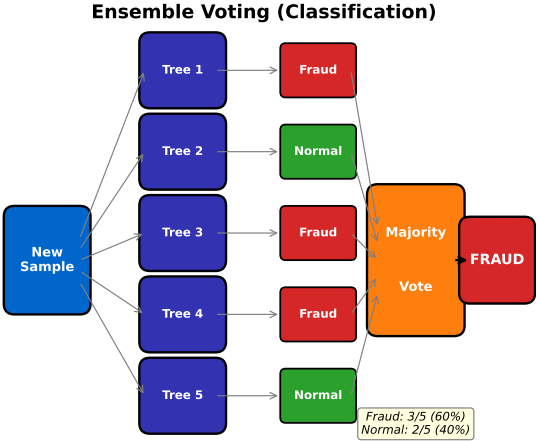
1. Identify trees where i was OOB (not in bootstrap sample)
2. Aggregate predictions from only those trees
3. Compare to true label

Benefits:

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

OOB error converges to leave-one-out cross-validation error

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:

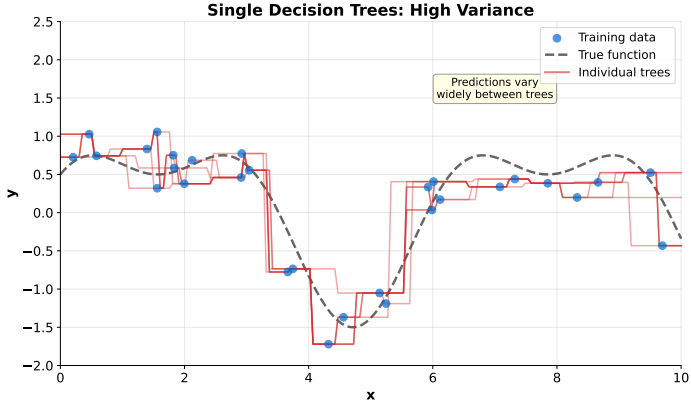
- Low bias (can fit complex patterns)
- High variance (sensitive to training data)

Random Forest:

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

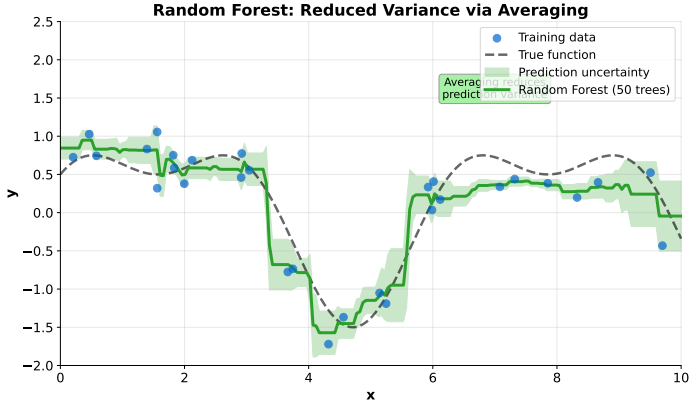
Ensembles reduce variance without increasing bias

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
- 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

Part 5: Practical Considerations

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:

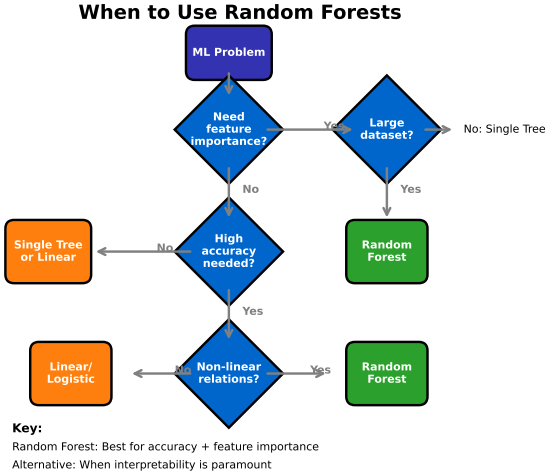
- Tabular data with mixed feature types
- Non-linear relationships expected
- Feature importance needed
- Out-of-the-box performance matters

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

Decision Framework



Start with Random Forest; switch if specific constraints require it

Implementation: scikit-learn

Classification:

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=100, max_features='sqrt',
                            oob_score=True, random_state=42)
rf.fit(X_train, y_train)
print(f"OOB Score: {rf.oob_score_:.3f}")
```

Key Parameters:

- `n_estimators`: number of trees
- `max_features`: features per split
- `oob_score`: compute OOB error
- `n_jobs`: parallel trees (-1 for all cores)

Set random_state for reproducibility

Summary: Random Forests

Core Concepts:

- Ensemble of decision trees with bootstrap + feature randomization
- Reduces variance while maintaining low bias
- OOB error provides free cross-validation

Practical Takeaways:

- Excellent default for tabular data
- Feature importance aids interpretation
- More trees never hurts (just slower)
- Hyperparameter tuning usually optional

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.

Documentation:

- scikit-learn: `sklearn.ensemble.RandomForestClassifier`

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