Lecture 5: Decision Trees & Random Forests

Luu Minh Sao Khue

From splitting criteria to ensemble learning — understanding decision trees, entropy and Gini, pruning and bias–variance, and how Random Forests improve stability via bagging and feature randomness.

1. Learning Objectives

After this lecture, you will be able to:

- Explain how decision trees split data using impurity measures.
- Derive information gain using entropy and Gini index.
- Control tree complexity with depth and pruning.
- Understand bias–variance tradeoff in trees.
- Describe bagging, random subspace, and Random Forest algorithms.
- Implement and visualize trees and forests using `scikit-learn`.
- Compare single tree vs. Random Forest accuracy.

2. Intuition

Decision trees partition the feature space into rectangular regions by asking a sequence of simple questions: "Is feature $x_j \leq t$?". At each node, the algorithm selects the best split that most *purely* separates the target classes (for classification) or minimizes variance (for regression).

Trees are easy to interpret but unstable: a small change in data can produce a very different tree. Ensembles such as Random Forests overcome this by averaging many trees trained on bootstrapped data samples.

3. Decision Tree Fundamentals

3.1 Recursive Partitioning

At each internal node, we select a feature j and a threshold t that maximizes the reduction in impurity. For classification, let $\mathcal{D}$ denote samples at a node, and p_k the fraction of class k within $\mathcal{D}$.

The impurity $I(\mathcal{D})$ measures class heterogeneity.

3.2 Entropy

$$I_{\text{entropy}}(\mathcal{D}) = -\sum_{k=1}^{K} p_k \log_2 p_k,$$

where $p_k = \frac{n_k}{|\mathcal{D}|}$ and n_k is the number of samples of class k. Entropy is maximal when classes are equally mixed and minimal when pure.

3.3 Gini Index

$$I_{\text{gini}}(\mathcal{D}) = \sum_{k=1}^{K} p_k(1 - p_k) = 1 - \sum_{k=1}^{K} p_k^2.$$

Gini impurity is computationally simpler and often used in CART (Classification and Regression Trees).

3.4 Information Gain

For candidate split (j, t), divide data into left and right subsets:

$$\mathcal{D}_L = \{x_i \mid x_{ij} \leq t\}, \quad \mathcal{D}_R = \{x_i \mid x_{ij} > t\}.$$

The information gain is

$$\text{Gain}(j, t) = I(\mathcal{D}) - \frac{|\mathcal{D}_L|}{|\mathcal{D}|} I(\mathcal{D}_L) - \frac{|\mathcal{D}_R|}{|\mathcal{D}|} I(\mathcal{D}_R).$$

We choose (j, t) with the largest gain.

3.5 Regression Trees

For continuous targets, impurity is measured by variance:

$$I_{\text{var}}(\mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} (y_i - \bar{y}_{\mathcal{D}})^2.$$

Splits minimize weighted average variance of children.

3.6 Stopping Criteria

Growth stops when:

- Maximum depth $d_{\max}$ reached,
- Node size $< n_{\min}$,
- No split improves impurity,
- Pure node (only one class).

4. Pruning and Regularization

4.1 Depth Control and Pruning

Deep trees can perfectly fit training data but overfit.

- **Pre-pruning:** limit depth, minimum samples per node.
- **Post-pruning:** grow full tree, then prune back branches that reduce validation performance.

4.2 Cost-Complexity Pruning

CART defines a cost function

$$R_\alpha(T) = R(T) + \alpha|T|,$$

where $R(T)$ is training error, $|T|$ is number of terminal nodes, and α penalizes complexity. Optimal subtree minimizes $R_\alpha(T)$.

4.3 Bias–Variance in Trees

Tree Type	Bias	Variance
Shallow tree	High	Low
Deep tree	Low	High

Ensembles such as Random Forests reduce variance by averaging many high-variance trees.

5. Random Forests

5.1 Ensemble Concept

A Random Forest is an ensemble of B decision trees

$$\hat{f}_{\mathrm{RF}}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}_b(x),$$

where each $\hat{f}_b$ is trained on a bootstrap sample of the data and uses random subsets of features.

5.2 Bagging (Bootstrap Aggregating)

Given training set $\mathcal{D}$ of size n, draw B bootstrap samples $\mathcal{D}^{(b)}$ (sampling with replacement). Train one tree on each sample. Bagging reduces variance because:

$$\mathrm{Var}\left(\frac{1}{B}\sum_{b} \hat{f}_b(x)\right) \approx \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2,$$

where ρ is correlation between trees. Lower ρ and large B reduce overall variance.

5.3 Random Subspace Method

At each split, only a random subset of m features (out of p) is considered. This decorrelates trees further and increases diversity.

Typical values:

$$m = \begin{cases} \sqrt{p}, & \text{for classification} \\ p/3, & \text{for regression.} \end{cases}$$

5.4 Out-of-Bag (OOB) Error

Each tree is trained on about 63% of samples (expected unique in a bootstrap). Remaining 37% serve as OOB data for validation. The OOB score estimates generalization error without cross-validation.

5.5 Feature Importance

Each split contributes to impurity reduction:

$$\mathrm{Imp}(j) = \frac{1}{B} \sum_{b=1}^{B} \sum_{t \in T_b :\, v(t)=j} p(t)\, \Delta I(t),$$

where $p(t)$ is proportion of samples reaching node t and ΔI is impurity decrease. Features with higher total importance are more influential.

6. Python Implementation and Visualization

```python
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Data
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42)

# Single Decision Tree
tree = DecisionTreeClassifier(
    criterion="gini", max_depth=3, random_state=42)
tree.fit(X_train, y_train)

plt.figure(figsize=(10, 6))
plot_tree(tree, filled=True, feature_names=load_iris().feature_names)
plt.title("Decision Tree (max_depth=3)")
plt.show()

# Random Forest
```

```python
rf = RandomForestClassifier(
    n_estimators=100, random_state=42, oob_score=True)
rf.fit(X_train, y_train)

# Evaluation
y_pred_tree = tree.predict(X_test)
y_pred_rf = rf.predict(X_test)

print("Tree accuracy:", accuracy_score(y_test, y_pred_tree))
print("RF accuracy:", accuracy_score(y_test, y_pred_rf))
print("OOB score:", rf.oob_score_)

# Feature importance
for name, val in zip(load_iris().feature_names, rf.feature_importances_):
    print(f"{name:25s}: {val:.3f}")
```

7. Model Comparison

Typical outcomes:

- Single tree interpretable but high variance.
- Random Forest higher accuracy, more stable, less interpretable.
- Feature importance provides global insights.

Empirically, Random Forests often outperform single trees, especially on noisy or high-dimensional data.

8. Summary

Decision Trees partition data via recursive splits optimizing impurity (Gini, entropy, variance). Their interpretability is offset by high variance and sensitivity to noise.

Random Forests aggregate many randomized trees to achieve lower variance and better generalization through:

- **Bagging:** training on bootstrap samples.
- **Random feature selection:** less correlation between trees.
- **OOB estimation:** built-in validation.
- **Feature importance:** model interpretability.

Bias–variance tradeoff:

$$\textit{Single tree} = \textit{low bias, high variance.}$$
$$\textit{Random forest} = \textit{moderate bias, low variance.}$$

9. Exercises

1. Derive the information gain formula using entropy.
2. Show that Gini impurity is minimized when a node is pure.
3. Implement cost-complexity pruning and plot $R_\alpha(T)$.
4. Train trees with various depths and plot training vs test error.
5. Compare bias and variance of trees vs Random Forests empirically.
6. Compute and visualize feature importance on any dataset.
7. Measure and interpret the OOB score vs. cross-validation accuracy.