L04: Decision Trees

Full Lecture: From Questions to Predictions

Methods and Algorithms

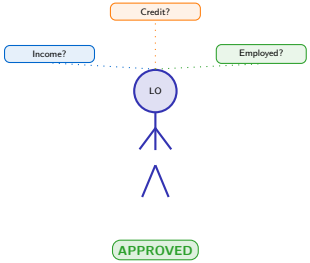
MSc Data Science

A Loan Officer Asks Three Questions and Decides in 30 Seconds

The Scene
A loan officer sits across from an applicant. Three questions flash through her mind:

- "What is your annual income?"
- "How long have you been employed?"
- "Any missed payments in the last 5 years?"

Within 30 seconds, she says: **Approved.**



Question

Can a machine learn which questions to ask — and in what order?

Decision trees formalize this human process: learn the best questions from data

What Will You Learn in This Lecture?

Learning Objectives

1. **Derive** Gini impurity and information gain from first principles
2. **Analyze** how recursive partitioning builds axis-aligned boundaries
3. **Evaluate** overfitting using pruning strategies and cross-validation
4. **Compare** decision trees with linear models for structured data

Prerequisites

Basic probability, linear/logistic regression concepts (L01–L02).

Why It Matters

DTs are the foundation for Random Forests, XGBoost, and LightGBM.

Bloom's levels: derive (4), analyze (4), evaluate (5), compare (5)

What Is a Decision Tree in Plain English?

A flowchart that learns from data

Think of the "20 Questions" game: each question narrows down the possibilities. A decision tree does the same — but **learns which questions to ask** by analyzing labeled data.

- No assumed functional form (non-parametric)
- Works with numerical and categorical features
- Each prediction has a human-readable path

Vocabulary

- **Root:** first question (most informative)
- **Internal node:** each decision point
- **Leaf:** final prediction
- **Split:** test that divides data
- **Depth:** levels from root to deepest leaf
- **Branch:** one path from root to leaf

Unlike linear models, decision trees can capture interactions and non-linearities automatically

Where Do Decision Trees Fit in Machine Learning?

Parametric vs Non-Parametric

	Parametric	Non-Parametric
Example	Linear/Logistic Reg.	Decision Trees
Assumption	Fixed form	Data-driven
# Parameters	Fixed	Grows with data
Interpretability	Coefficients	Decision paths

The ML Landscape So Far

- L01: Linear Regression → continuous prediction
- L02: Logistic Regression → probability + classification
- L03: KNN/K-Means → instance-based + clustering
- **L04: Decision Trees** → rule-based, non-linear

Bridge

Decision trees are the **building blocks for Random Forests** (next lecture).

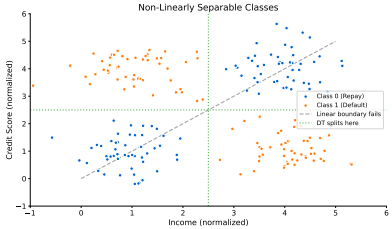
DTs bridge interpretability (like linear models) and flexibility (like KNN)

Why Can't a Straight Line Separate These Borrowers?

The Non-Linearity Problem

Some classification problems cannot be solved with a single linear boundary. The XOR-like pattern on the right shows two classes interleaved:

- A diagonal line misclassifies many points
- But two axis-aligned splits at $x = 2.5$ and $y = 2.5$ separate perfectly
- Decision trees excel at this



DTs create axis-aligned rectangles — any non-linear boundary can be approximated with enough splits

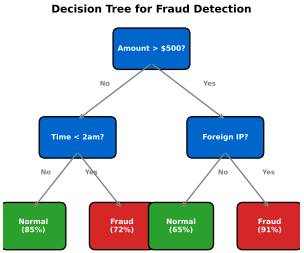
What Does a Trained Decision Tree Look Like?

Anatomy of a Trained Tree

- **Root** (top): the most informative split
- **Branches**: conditions that route data left or right
- **Leaves** (bottom): final class predictions
- Each path = one interpretable rule

Insight

The tree on the right was trained on fraud detection data — follow any path from root to leaf to read the decision rule.



sklearn's `export_text()` **prints the full tree as readable if/else rules**

How Does the Tree Choose the Best Split? (Gini Impurity)

Gini Impurity Formula

For a node with K classes and class proportions $p_1, p_2, \ldots, p_K$:

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

Worked example: 100 loan applicants, 60 repay, 40 default.

$$G = 1 - (0.6^2 + 0.4^2) = 1 - (0.36 + 0.16) = 1 - 0.52 = \mathbf{0.48}$$

Interpretation

$G = 0$ means pure node (all one class). $G = 0.5$ means maximum impurity for 2 classes.

Intuition

Gini = probability that a randomly chosen sample would be **misclassified** if labeled by the node's distribution.

Binary Gini: $G = 2p(1-p)$**, peaks at** $p = 0.5$

Which Split Reduces Impurity the Most?

Weighted Gini After Split

Parent: 100 applicants (60/40), $G = 0.48$.
After splitting on "Income ¿ $50k":

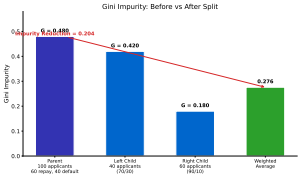
- Left child: 40 apps (10/30) $\rightarrow$ $G = 0.375$
- Right child: 60 apps (50/10) $\rightarrow$ $G = 0.278$

Weighted average:

$$G_{\text{weighted}} = \frac{40}{100} \times 0.375 + \frac{60}{100} \times 0.278 = \mathbf{0.317}$$

Impurity reduction:

$$\Delta G = 0.48 - 0.317 = \mathbf{0.163}$$



The tree greedily picks the feature and threshold that maximize ΔG at each node

What Does Entropy Add Beyond Gini?

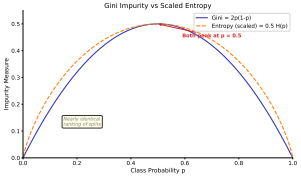
Entropy (Information Theory)

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

Same example: 60 repay, 40 default.

$$H = -(0.6 \log_2 0.6 + 0.4 \log_2 0.4) = \mathbf{0.971}\ \ \textbf{bits}$$

- $H = 0$ for pure node, $H = 1$ for 50/50 binary
- Entropy penalizes impurity slightly more than Gini
- In practice, both produce nearly identical trees



sklearn default is Gini; use `criterion='entropy'` to switch

Information Gain: The Decision Rule

Information Gain = Entropy Before − Weighted Entropy After

$$\text{IG}(D, \text{feature}) = H(D) - \sum_{v \in \text{values}} \frac{|D_v|}{|D|} \times H(D_v)$$

Worked example with 2 candidate features:

Split on Income ($>$ \$50k):

- Left: 40 apps, $H = 0.881$
- Right: 60 apps, $H = 0.469$
- IG $= 0.971 - (0.4 \times 0.881 + 0.6 \times 0.469)$
- IG $= 0.971 - 0.634 =$ **0.337**

Split on Employment ($>$ 2yr):

- Left: 50 apps, $H = 0.722$
- Right: 50 apps, $H = 0.722$
- IG $= 0.971 - (0.5 \times 0.722 + 0.5 \times 0.722)$
- IG $= 0.971 - 0.722 =$ **0.249**

Decision

Income wins (IG $= 0.337 > 0.249$) $\rightarrow$ split on Income first.

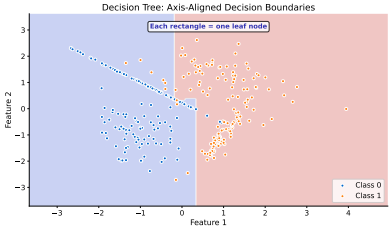
ID3 uses information gain; C4.5 uses gain ratio to correct for multi-valued features

How Does the Tree Partition Feature Space?

Axis-Aligned Rectangles

Each split creates a horizontal or vertical boundary in feature space. After depth 3:

- Up to $2^3 = 8$ rectangular regions
- Each rectangle = one leaf node
- More depth = finer rectangles = risk of overfit



Limitation

DTs can only make axis-aligned cuts — diagonal boundaries need many splits to approximate.

Each rectangle corresponds to a path: "if $x_1 > a$ AND $x_2 \leq b$ then class = ..."

The Recursive Partitioning Algorithm

GROW-TREE: How the Algorithm Works

Require: Dataset D, feature set F, max depth d
Ensure: A decision tree T

if all samples in D have same label OR $d = 0$ OR $|D| <$ min_samples **then**
    return leaf with majority class
end if
$f^*, t^* \leftarrow$ feature and threshold maximizing ΔG (or IG)
$D_L \leftarrow \{x \in D : x_{f*} \leq t^*\}$
$D_R \leftarrow \{x \in D : x_{f*} > t^*\}$
left $\leftarrow$ GROW-TREE(D_L, F, $d-1$)
right $\leftarrow$ GROW-TREE(D_R, F, $d-1$)
return node with split (f^*, t^*), left, right

Key Point

Greedy: each split is locally optimal. No backtracking — finding the globally optimal tree is NP-hard.

Time complexity: $O(n \cdot p \cdot d)$ **where** n **= samples,** p **= features,** d **= depth**

What Are the Stopping Rules?

When Does the Tree Stop Growing?

Parameter	Effect	Trade-off
`max_depth`	Limits tree levels	Too low → underfit; too high → overfit
`min_samples_leaf`	Minimum samples in a leaf	Higher = smoother; lower = noisier
`min_impurity_decrease`	Minimum ΔG to split	Prunes weak splits early
`max_leaf_nodes`	Caps number of leaves	Controls model complexity

Insight

These are pre-pruning strategies — they prevent overfitting before it happens.

Use cross-validation to select optimal values for each stopping parameter

Interpreting the Tree: What Does Each Path Mean?

Three Loan Applicants Through the Tree

Applicant	Path	Interpretable Rule	Decision
A	Income $>$ 50k $\rightarrow$ Credit $>$ 680	High income AND good credit	**Approve**
B	Income $\leq$ 50k $\rightarrow$ Employment $>$ 2yr	Low income BUT stable job	**Approve**
C	Income $\leq$ 50k $\rightarrow$ Employment $\leq$ 2yr	Low income AND short tenure	**Deny**

Key Advantage

Every prediction comes with a **reason** — critical for regulated industries.

Limitation

Interpretability fades as depth increases — a 20-level path is hard to explain.

ECOA requires lenders to explain adverse actions — DT paths provide natural explanations

Reading Feature Importance from a Single Tree

Mean Decrease in Impurity (MDI)

- Features used near the **root** contribute the most impurity reduction
- MDI sums ΔG across all nodes where a feature is used, weighted by samples
- Normalized so all importances sum to 1

Permutation Importance (alternative):

- Shuffle one feature, measure accuracy drop
- More reliable for correlated features
- Slower but unbiased

Comparison

	MDI	Permutation
Speed	Fast	Slow
Bias	High-card.	Unbiased
Scope	Train	Test

MDI overestimates importance of high-cardinality features — always cross-check with permutation

The Overfitting Problem: Why Deep Trees Memorize

Pre-Pruning vs Post-Pruning

Pre-Pruning (stop early):

- Set `max_depth`, `min_samples_leaf`
- Fast but may stop too soon
- Used by sklearn's `DecisionTreeClassifier`

Post-Pruning (grow then cut):

- Grow full tree, then remove branches
- Cost-complexity pruning (CCP):

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

where $R(T)$ = misclassification rate, $|T|$ = number of leaves, α = complexity penalty.

Insight

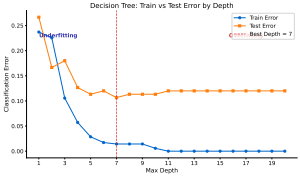
Increasing α prunes more aggressively — use CV to find optimal α.

sklearn: use `ccp_alpha` parameter and `cost_complexity_pruning_path()` to find candidates

Can You See the Overfitting?

Depth vs Error

- Train error drops monotonically as depth increases
- Test error drops, then **rises** — classic overfitting
- Optimal depth: where test error is minimized
- Use k-fold cross-validation to select depth



Insight

The gap between train and test error is a direct measure of overfitting.

sklearn: `GridSearchCV(param_grid={'max_depth': range(1,20)})` **automates this**

Regression Trees: Predicting Numbers, Not Categories

MSE Instead of Gini

For regression, each leaf predicts the **mean** of its samples. The split criterion becomes:

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

Worked example: Predict house price.

- Parent: 10 houses, mean price = \$300k, MSE = 5000
- Split on sqft > 1500:
- Left: 4 houses, mean = \$200k, MSE = 1200
- Right: 6 houses, mean = \$367k, MSE = 800
- Weighted MSE = $\frac{4}{10} \times 1200 + \frac{6}{10} \times 800 = 960$
- Reduction = $5000 - 960 = 4040$

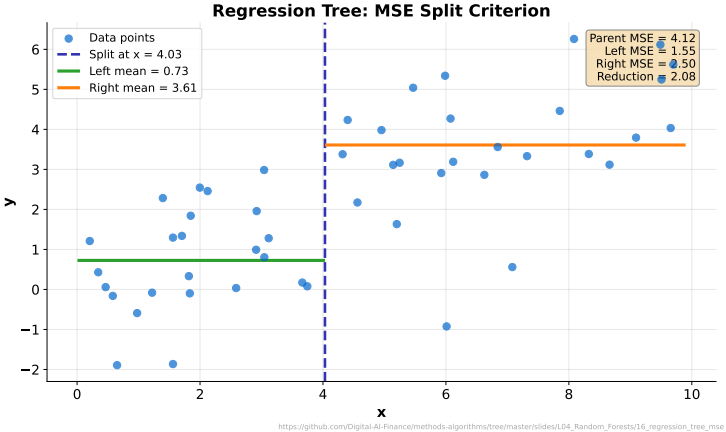
Key

Same algorithm, different criterion: minimize variance instead of impurity.

---

sklearn: `DecisionTreeRegressor(criterion='squared_error')`

Regression Trees: MSE Split Visualization



A depth-1 regression tree finds the optimal split that minimizes weighted MSE in child nodes

When to Use a Decision Tree — and When Not To

Pros

- Interpretable decision paths
- Handles mixed feature types
- No feature scaling needed
- Fast training and prediction

Cons

- High variance (unstable)
- Prone to overfitting
- Axis-aligned boundaries only
- Small changes in data → different tree

Bridge to Random Forests

The solution to high variance: **grow many trees and let them vote** → Random Forests (next lecture).

A single tree is rarely used in production — it is a building block for ensembles

DT vs Logistic Regression for Credit Scoring

Head-to-Head Comparison

Property	Decision Tree	Logistic Regression
Decision boundary	Axis-aligned rectangles	Linear hyperplane
Feature interactions	Captured automatically	Must engineer manually
Interpretability	Path-based rules	Odds ratios
Probability output	Leaf proportions (noisy)	Calibrated via sigmoid
Stability	Low (high variance)	High (low variance)
Missing values	Can handle natively	Requires imputation
Non-linearity	Built-in	Requires feature transforms

When to Choose Which

Regulated setting with linear relationships → logistic regression. Complex interactions → DT or RF.

Many teams use both: logistic regression for baseline, DT/RF for production

Hands-on: Build a Credit Scoring Tree

Exercise with sklearn

Step 1: Load and split

```
from sklearn.tree import (
    DecisionTreeClassifier,
    export_text)
from sklearn.model_selection import (
    train_test_split)

X_train, X_test, y_train, y_test = \
    train_test_split(X, y,
    test_size=0.3, random_state=42)
```

Step 2: Train and evaluate

```
clf = DecisionTreeClassifier(
    max_depth=4,
    min_samples_leaf=10,
    random_state=42)
clf.fit(X_train, y_train)

print(f"Test acc: "
  f"{clf.score(X_test, y_test):.3f}")
print(export_text(clf,
  feature_names=feat_names))
```

Tasks

(1) Vary `max_depth` from 1 to 15 and plot train/test accuracy. (2) Find optimal depth via CV.

Use `plot_tree(clf)` to visualize — or `export_graphviz` for publication-quality figures

Key Takeaways

Concepts

- **Gini impurity**: $G = 1 - \sum p_k^2$
- **Entropy**: $H = -\sum p_k \log_2 p_k$
- **Information gain**: parent entropy minus weighted child entropy
- **Recursive partitioning**: greedy, top-down
- **Pruning**: pre- and post- to control depth

Practical Skills

- Interpret paths as decision rules
- Choose depth via cross-validation
- Read feature importance (MDI + permutation)
- When to use DT vs logistic regression
- DTs $\rightarrow$ building blocks for RF

One-Sentence Summary

A decision tree learns the best sequence of yes/no questions from data, trading interpretability for variance.

Master decision trees and you have the foundation for all tree-based ensemble methods

What's Next: From One Tree to a Forest

The Problem with One Tree

- A single tree is **unstable**: small data changes → completely different tree
- High variance limits practical accuracy
- The fix: **grow 500 different trees and let them vote**

Random Forest Recipe

1. Bootstrap sample the data
2. At each split, use $\sqrt{p}$ random features
3. Grow many decorrelated trees
4. Aggregate by majority vote

Why It Works

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

Decorrelating trees ($\rho \downarrow$) reduces ensemble variance.

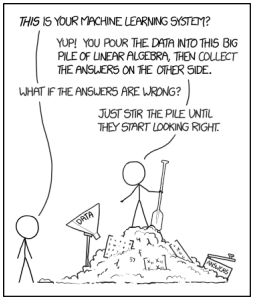
Next lecture: Random Forests — bagging, feature randomization, OOB error, feature importance

Closing Thought

"The tree that learns too well from the past is doomed to fail in the future."

Decision trees teach us the fundamental trade-off in machine learning: **complexity vs generalization**. Every additional split fits the training data better — but risks memorizing noise.

The solution? Don't trust one tree. Trust a forest.



XKCD #1838 by Randall Munroe (CC BY-NC 2.5)