CS 457/557: Machine Learning

Lecture 05-*:
Decision Trees

University of Wisconsin
LA CROSSE

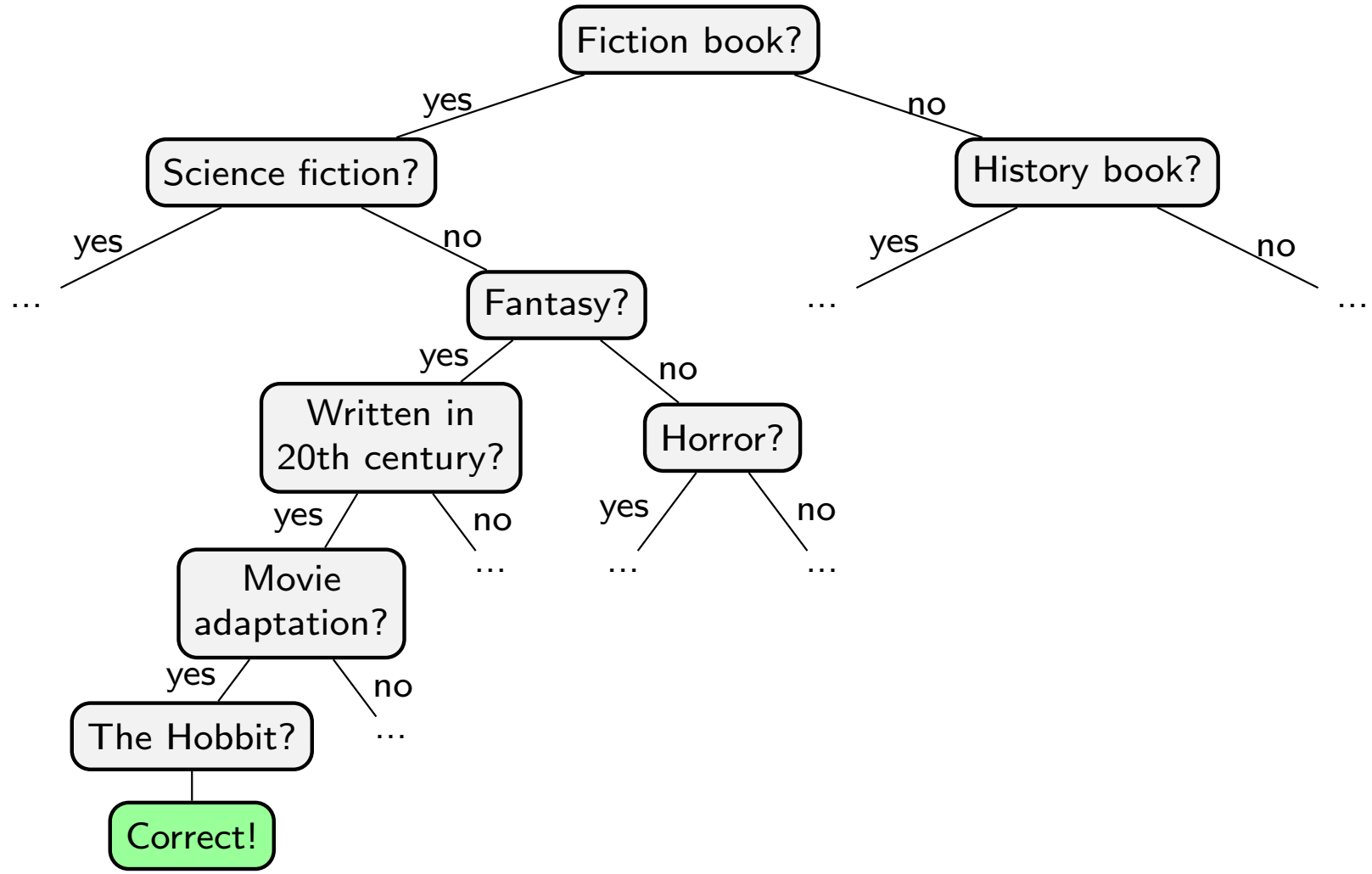
Prof. Jason Sauppe
jsauppe@uwlax.edu
https://cs.uwlax.edu/~jsauppe/

Fall 2024

Lecture 05-1a: Decision Trees

Classification by Flow Chart

Motivating example: 20 Questions: I'm thinking of a book...

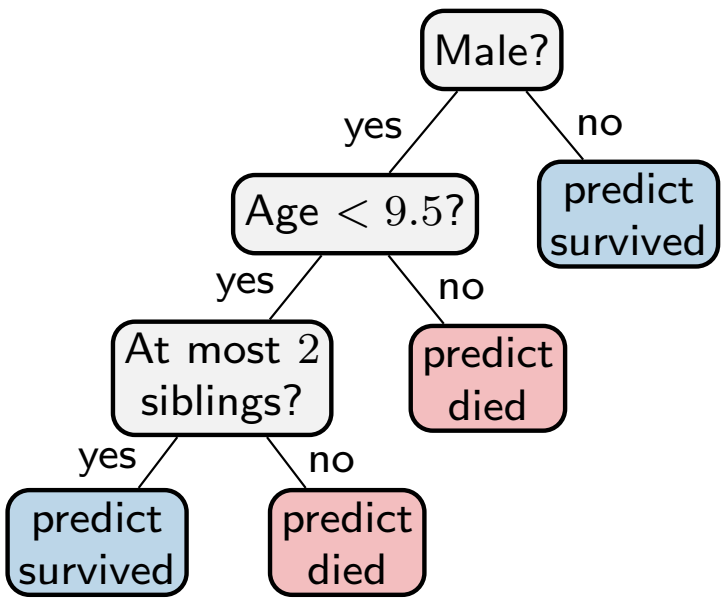


Decision Trees

Definition

A **decision tree** is a tree that recursively partitions the feature space into regions based on splits in attribute values.

Example: Predicting survival of passengers on the Titanic:

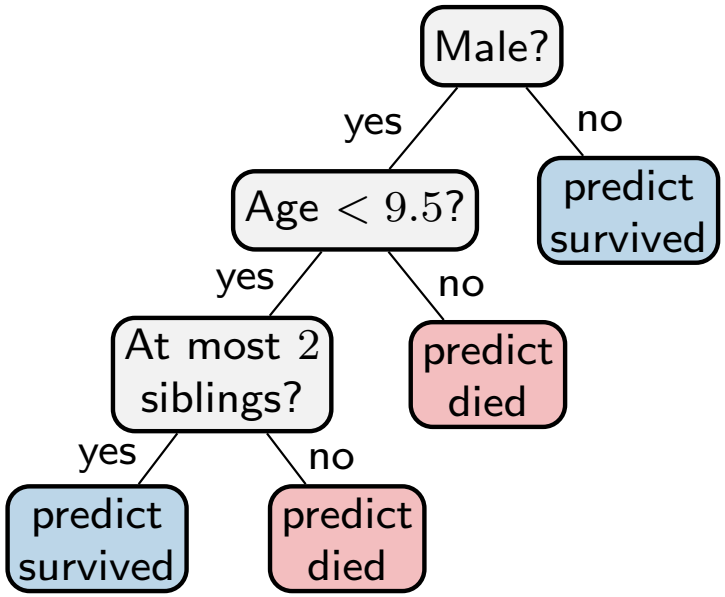


A decision tree has the following properties:

1. each non-leaf node corresponds to a question about an attribute
2. each branch (edge) corresponds to an answer to the question at the parent node
3. each leaf node corresponds to a predicted output

Basics of Decision Trees

Example: Predicting survival of passengers on the Titanic:



Splitting on an attribute j depends on attribute type:

- if quantitative, splits generally take the form "$x_j < t$?" for some $t \in \mathbb{R}$
- if categorical, split on category values

Each leaf node represents a **region** in feature space.

The predicted value $\hat{y}$ for a point $\mathbf{x}$ is determined by following the path through the tree to identify the region to which $\mathbf{x}$ belongs.

- In **classification**, $\hat{y}$ is the **majority class** in the region
- In **regression**, $\hat{y}$ is the average output value of points in the region

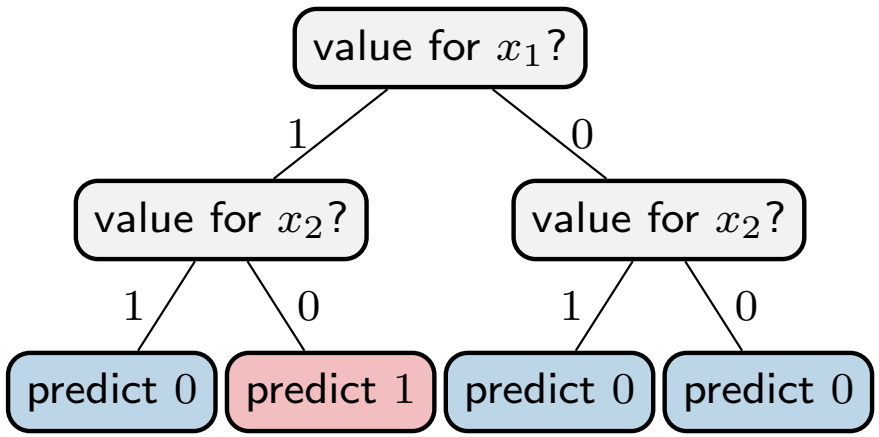
Expressiveness of Decision Trees

Decision trees can be used to express **any deterministic function**!

Example: Consider the function below with two binary inputs and one binary output:

x_1	x_2	$y = x_1 \wedge \neg x_2$
1	1	0
1	0	1
0	1	0
0	0	0

Corresponding decision tree:



More generally, any boolean function can be encoded as a decision tree:

- Each row in the truth table corresponds to a path in the tree
- Leaf nodes encode the output of the function with the path from the root as input

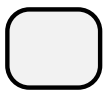
Lecture 05-1b:
Building Decision Trees

Building Decision Trees, Take 1

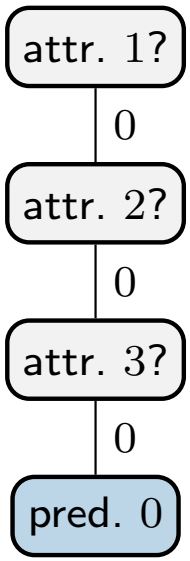
To **build** a decision tree from a data set $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, we could try to create different paths from the root for each input $\mathbf{x}_i$.

	Attributes			Output
Example	1	2	3	y
$\mathbf{x}_1$	0	0	0	0
$\mathbf{x}_2$	0	1	1	1
$\mathbf{x}_3$	1	0	1	1
$\mathbf{x}_4$	1	0	0	0

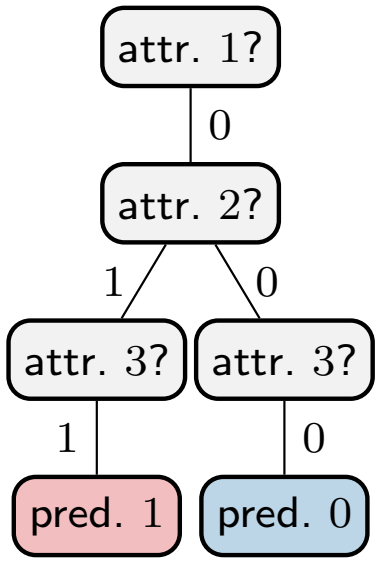
Start with empty tree:



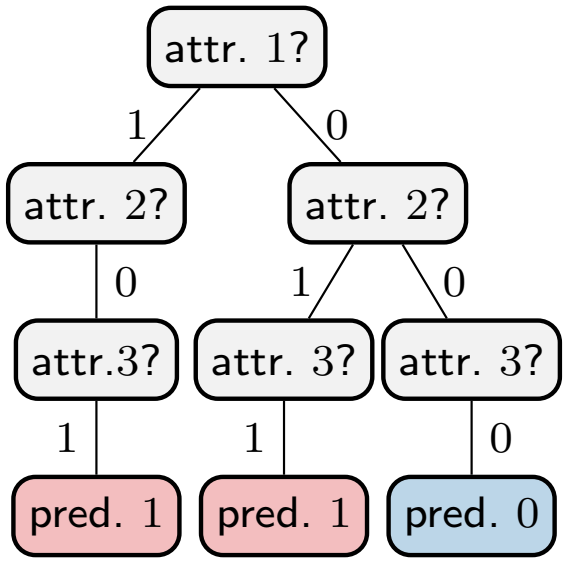
Insert $\mathbf{x}_1$:



Insert $\mathbf{x}_2$:



Insert $\mathbf{x}_3$:

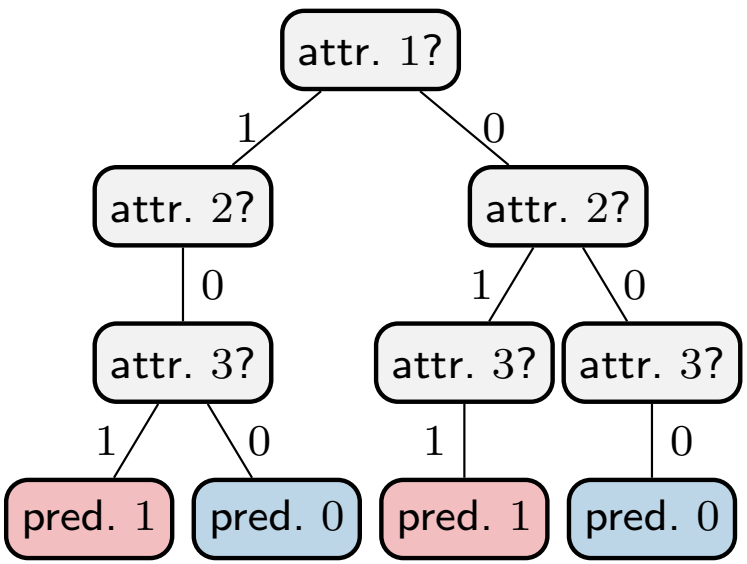


Building Decision Trees, Take 1 (continued)

To **build** a decision tree from a data set $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, we could try to create different paths from the root for each input $\mathbf{x}_i$.

	Attributes			Output
Example	1	2	3	y
$\mathbf{x}_1$	0	0	0	0
$\mathbf{x}_2$	0	1	1	1
$\mathbf{x}_3$	1	0	1	1
$\mathbf{x}_4$	1	0	0	0

Insert $\mathbf{x}_4$:



To **predict** the output $\hat{y}$ associated with a new example $\mathbf{x}$, we follow the path in the tree specified by the values of $\mathbf{x}$:

- If we reach a leaf, we use the leaf's value as our prediction
- If we reach a non-leaf node S that branches on attribute j but has no child branch corresponding to attribute value x_j, then we predict the most common output (class) from all (training) examples at or below S

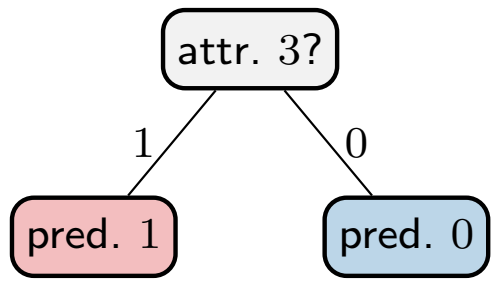
Problems with the Building Process

There are some **problems** with trees built in this way:

1. What happens with collisions?
2. They are **overfit** to the training data and do not **generalize** well to new examples.
3. They are not **compact**.

A more compact tree results from splitting on attribute 3 only:

	Attributes			Output
Example	1	2	3	y
$\mathbf{x}_1$	0	0	0	0
$\mathbf{x}_2$	0	1	1	1
$\mathbf{x}_3$	1	0	1	1
$\mathbf{x}_4$	1	0	0	0



Decision Trees: Constructive Heuristics

Instead of building a decision tree by "inserting" full paths corresponding to data points, we typically use a **constructive heuristic** to build the tree in a **top-down** fashion:

1. Initialize a root node S_0 containing (indices of) all training examples $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$
2. Split the root node S_0:
   a. Choose an attribute j from the data set
   b. Divide examples at S_0 according to the values of attribute j, placing each subset into a new subtree
3. Repeat Step 2 to split the root nodes of each subtree as needed

There are many possible implementations of this, but it is perhaps most easily understood recursively!

Recursive Splitting

For now, we'll assume that each attribute is **qualitative** (categorical). The pseudocode below shows how we could split a decision tree node:

Recursive Splitting of Nodes in a Decision Tree

```
procedure RecursiveSplit(Node S)        ▷ S is a subset of (indices of) examples
    return if S cannot or should not be split further        ▷ S is a leaf node
    Choose an attribute j for splitting
    for each possible value v of attribute j do
        S_v ← {i ∈ S | x_ij = v}
        RecursiveSplit(S_v)
```

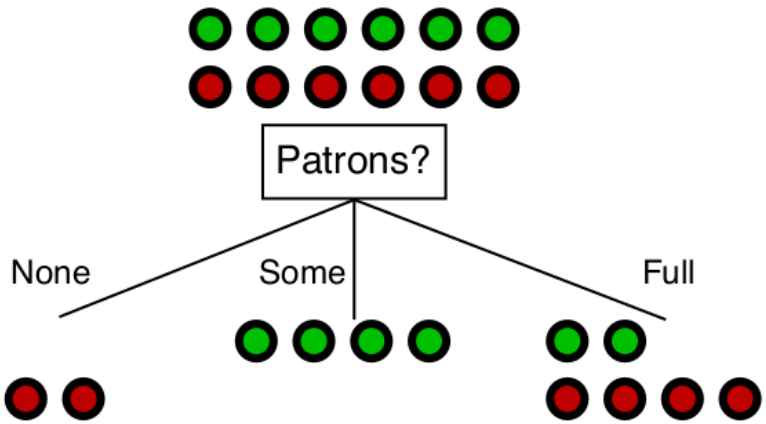
Some details are **missing** from the above pseudocode:

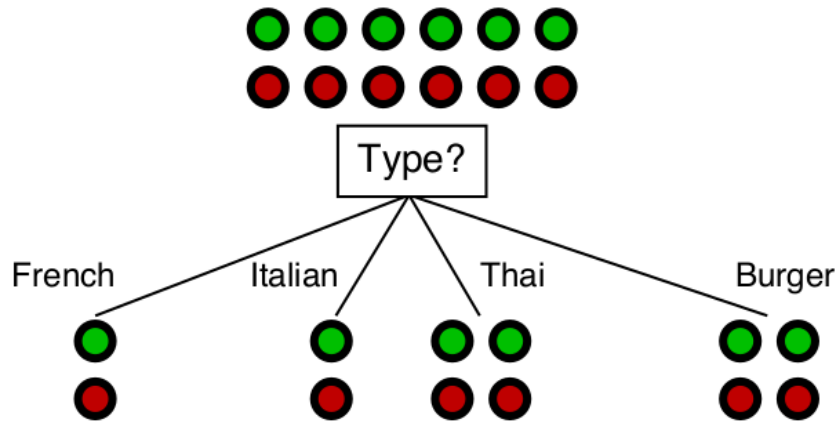
- Stopping conditions are not fully specified
- Attribute selection process is not fully specified
- Tree structure is not being stored yet

Lecture 05-2a:
Choosing Attributes

Choosing Attributes

Consider the problem of predicting whether a person will **wait** for a table (or **not wait**) at a restaurant, with two possible attribute splits at the root, either on number of patrons or restaurant type:





Intuitively, a good choice of attribute for splitting is one that gives us the **most information** about how output decisions are made.

- In the above example, $Patrons$ gives more information than $Type$ because some values of $Patrons$ yield child nodes with all examples belonging to a single class, while no values of $Type$ do this

Lecture 05-2b:
Entropy and Information Gain

Ranking Attributes by Importance

To formalize the notion of **attribute importance**, we need the following:

- Assume there are K output values (classes) labeled $1, 2, \ldots, K$
- Any node S in the decision tree contains a subset of (indices of) the training data set $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$
  - The root node S_0 contains all examples: $S_0 = \{1, 2, \ldots, n\}$
- At any node S, let $\hat{p}_k(S)$ denote the **proportion** of examples at S belonging to class k, for each $k \in \{1, 2, \ldots, K\}$:

$$\hat{p}_k(S) = \frac{|\{i \in S \mid y_i = k\}|}{|S|}$$

With this, we can define the **entropy** of a node S as

$$H(S) = -\sum_{k=1}^{K} \hat{p}_k(S) \log_2 \hat{p}_k(S).$$

Intuition for Entropy

Entropy comes from the field of information theory.

Some intuition: Consider a **Bernoulli random variable** V that represents the outcome of a coin toss with probability p of heads.

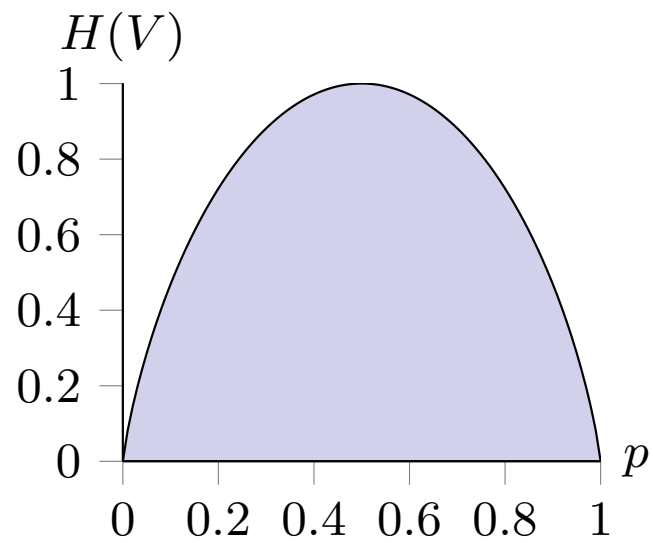
The entropy of V is

$$\begin{aligned} H(V) &= -\sum_{i \in \text{range}(V)} P(V=i) \log_2 P(V=i) \\ &= -\big[P(V=1)\log_2 P(V=1) \\ &\quad + P(V=0)\log_2 P(V=0)\big] \\ &= -\left[p \log_2 p + (1-p)\log_2(1-p)\right]. \end{aligned}$$

Entropy is

- **maximized** when $p = 0.5$ (i.e., each outcome is equally likely)
- **minimized** when $p = 0$ or $p = 1$

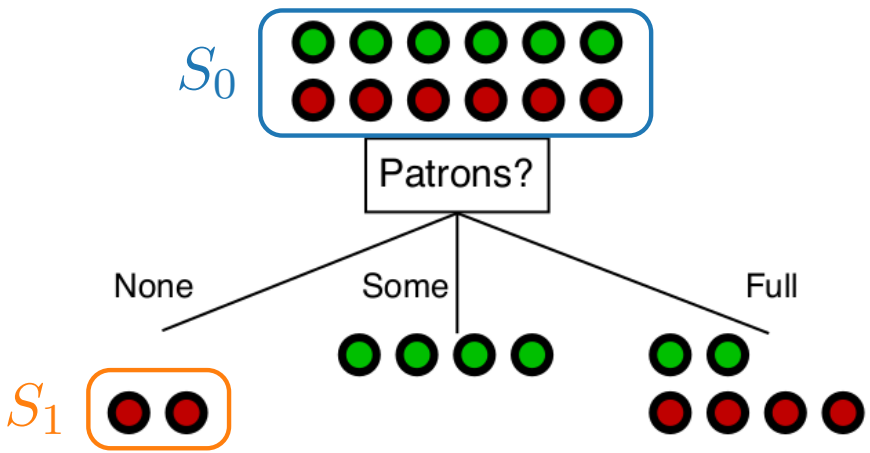
Entropy for various values of p:



Entropy for Decision Tree Nodes

In the context of decision trees, **entropy** measures the **uncertainty** about the examples at a node.

Let's revisit the restaurant problem:



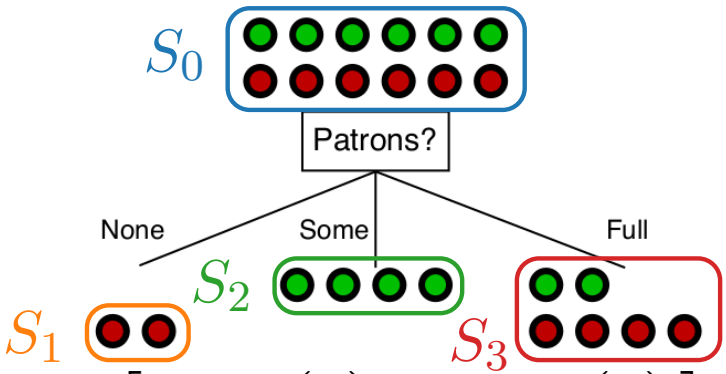
Some calculations:

$$H(S_0) = -\left[\frac{6}{12}\log_2\left(\frac{6}{12}\right) + \frac{6}{12}\log_2\left(\frac{6}{12}\right)\right]$$
$$= -\left[\frac{1}{2}\cdot(-1) + \frac{1}{2}\cdot(-1)\right] = 1.0$$
$$H(S_1) = -\left[\frac{0}{2}\log_2\left(\frac{0}{2}\right) + \frac{2}{2}\log_2\left(\frac{2}{2}\right)\right]$$
$$= -\left[0 + 2\cdot 0\right] = 0$$

Observations:

- Nodes containing examples from a single class have an entropy of 0
- Nodes with examples from multiple classes have entropy > 0

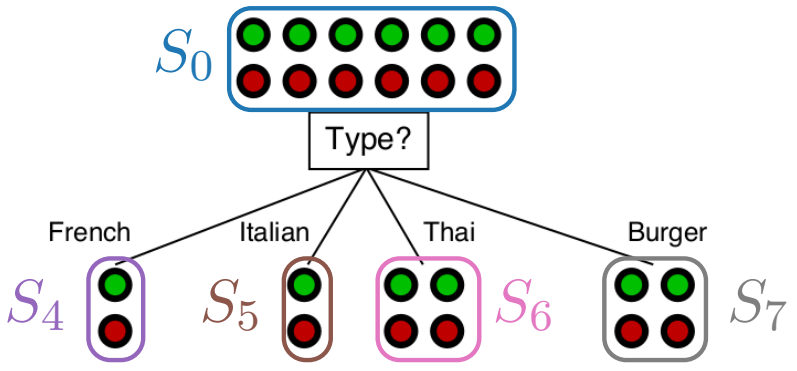
Entropy Calculations



$$H(S_1) = -\left[\frac{0}{2}\log_2\left(\frac{0}{2}\right) + \frac{2}{2}\log_2\left(\frac{2}{2}\right)\right]$$
$$= 0$$

$$H(S_2) = -\left[\frac{4}{4}\log_2\left(\frac{4}{4}\right) + \frac{0}{4}\log_2\left(\frac{0}{4}\right)\right]$$
$$= 0$$

$$H(S_3) = -\left[\frac{2}{6}\log_2\left(\frac{2}{6}\right) + \frac{4}{6}\log_2\left(\frac{4}{6}\right)\right]$$
$$\approx 0.918$$



$$H(S_4) = -\left[\frac{0}{2}\log_2\left(\frac{0}{2}\right) + \frac{2}{2}\log_2\left(\frac{2}{2}\right)\right] = 1.0$$

$$H(S_5) = -\left[\frac{4}{4}\log_2\left(\frac{4}{4}\right) + \frac{0}{4}\log_2\left(\frac{0}{4}\right)\right] = 1.0$$

$$H(S_6) = -\left[\frac{2}{6}\log_2\left(\frac{2}{6}\right) + \frac{4}{6}\log_2\left(\frac{4}{6}\right)\right] = 1.0$$

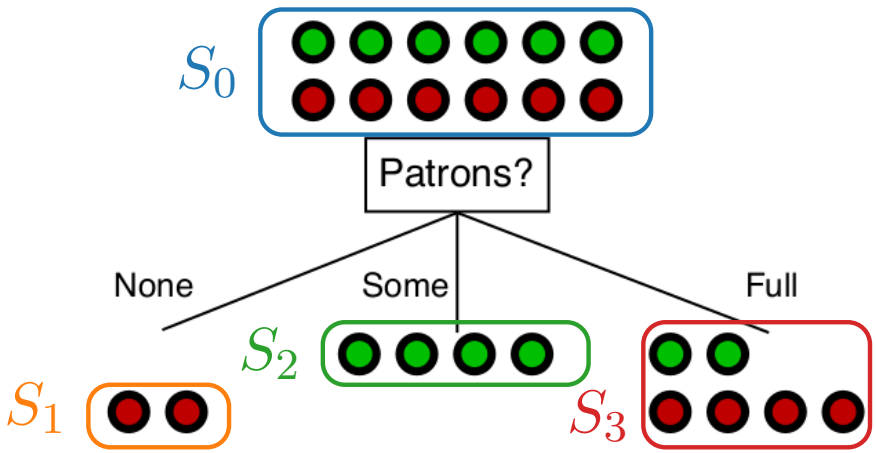
$$H(S_7) = -\left[\frac{2}{6}\log_2\left(\frac{2}{6}\right) + \frac{4}{6}\log_2\left(\frac{4}{6}\right)\right] = 1.0$$

Entropy After Splitting

Suppose we split a node S on attribute j whose set of possible values is V_j. This creates child nodes $S_v = \{i \in S \mid x_{ij} = v\}$ for each $v \in V_j$, each of which has its own entropy $H(S_v)$.

Then the **(weighted) remaining entropy** after splitting node S on attribute j is

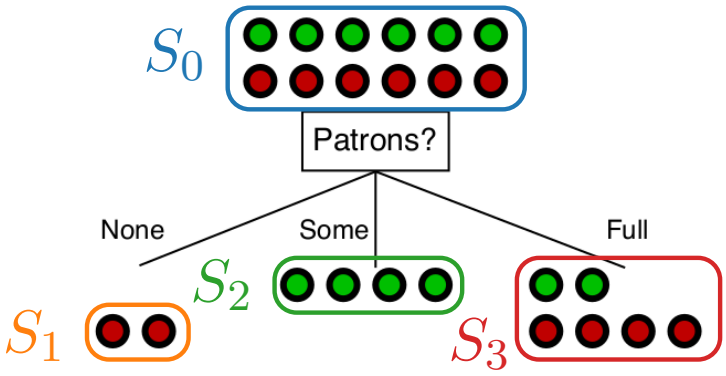
$$Remainder(S, j) = \sum_{v \in V_j} \frac{|S_v|}{|S|} H(S_v).$$

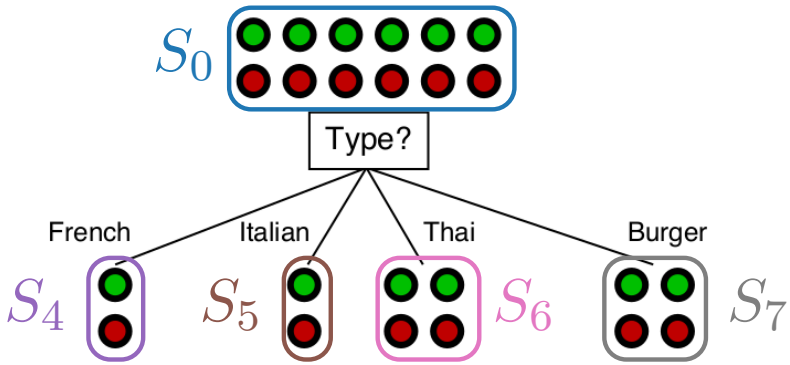


$$Remainder(S_0, Patrons)$$
$$= \frac{|S_1|}{|S_0|} H(S_1) + \frac{|S_2|}{|S_0|} H(S_2) + \frac{|S_3|}{|S_0|} H(S_3)$$
$$= \frac{2}{12} \cdot 0 + \frac{4}{12} \cdot 0 + \frac{6}{12} \cdot 0.918$$
$$\approx 0.459$$

Entropy After Splitting: Calculations

We can compute the **remaining entropy** for each possible split:





$$Remainder(S_0, Patrons)$$
$$= \frac{|S_1|}{|S_0|} H(S_1) + \frac{|S_2|}{|S_0|} H(S_2) + \frac{|S_3|}{|S_0|} H(S_3)$$
$$= \frac{2}{12} \cdot 0 + \frac{4}{12} \cdot 0 + \frac{6}{12} \cdot 0.918$$
$$\approx 0.459$$

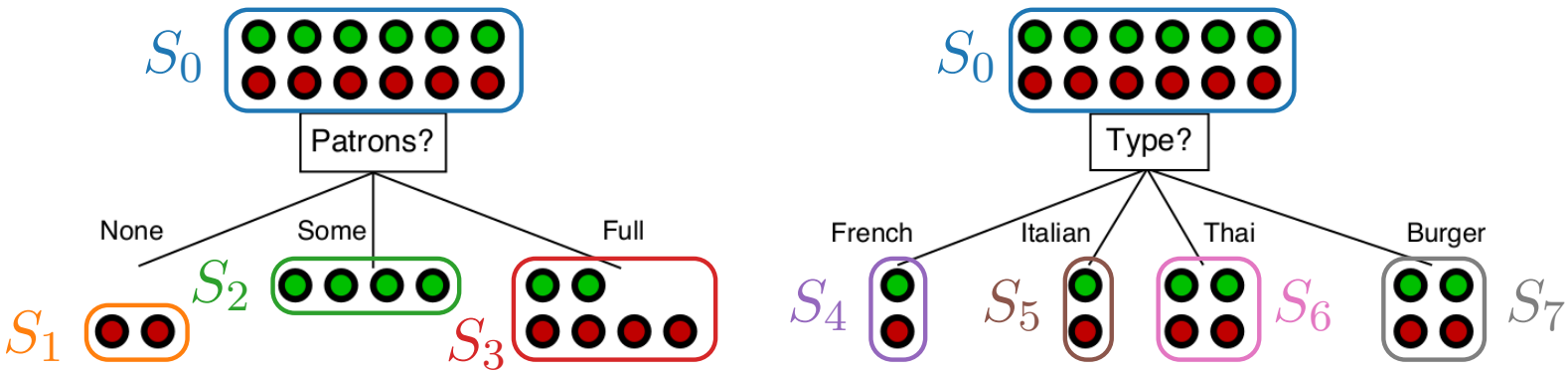
$$Remainder(S_0, Type)$$
$$= \frac{|S_4|}{|S_0|} H(S_4) + \frac{|S_5|}{|S_0|} H(S_5) + \frac{|S_6|}{|S_0|} H(S_6) + \frac{|S_7|}{|S_0|} H(S_7)$$
$$= \frac{2}{12} \cdot 1 + \frac{2}{12} \cdot 1 + \frac{4}{12} \cdot 1 + \frac{4}{12} \cdot 1$$
$$= 1.0$$

The Benefit of a Split: Information Gain

The **information gain** (entropy reduction) after splitting node S on attribute j is defined as

$$Gain(S, j) = H(S) - Remainder(S, j).$$

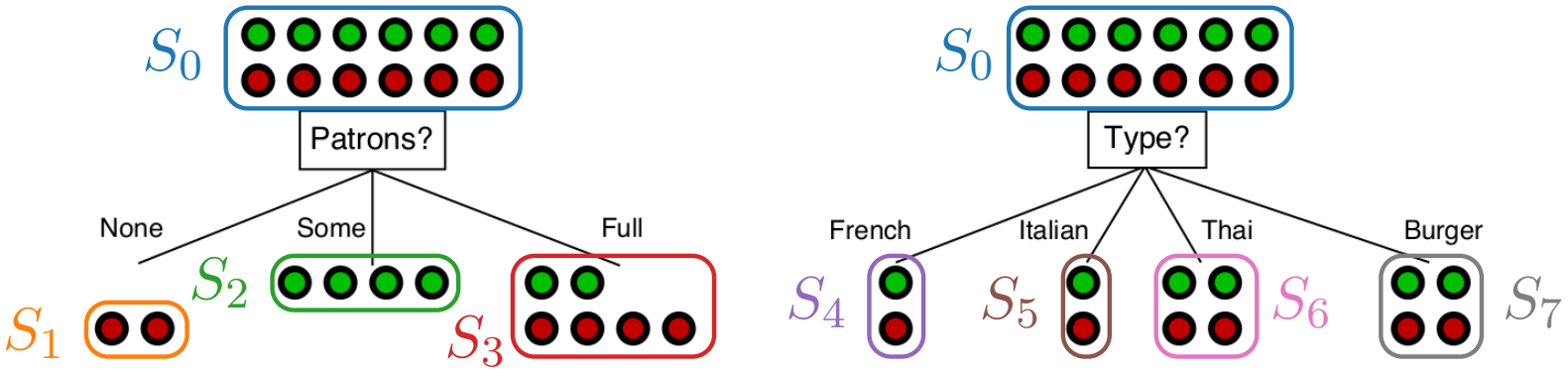
Information gain allows us to quantify the **benefit** of a split:



$$Gain(S_0, Patrons) = H(S_0) - Remainder(S_0, Patrons) \approx 1.0 - 0.459 = 0.541$$
$$Gain(S_0, Type) = H(S_0) - Remainder(S_0, Type) = 1.0 - 1.0 = 0.0$$

Ranking Splits Using Information Gain

With **information gain**, we can **rank** splits by quality:



Because

$$Gain(S_0, Patrons) \approx 0.541 > 0 = Gain(S_0, Type),$$

we would choose to split on $Patrons$ instead of $Type$.

Lecture 05-2c:
Decision Tree Learning

Decision Tree Learning Algorithm: Textbook Version

Pseudocode for Decision Tree Learning (from textbook)

procedure DECISION-TREE-LEARNING($examples$, $attributes$, $parent\_examples$)
  if $examples$ is empty **then return** PLURALITY-VALUE($parent\_examples$)
  else if all $examples$ have the same class **then return** the class
  else if $attributes$ is empty **then return** PLURALITY-VALUE($examples$)
  else
    $A \leftarrow \arg\max_{a \in attributes}$ IMPORTANCE(a, $examples$)
    $tree \leftarrow$ a new decision tree with root test A
    for each value v_k of A **do**
      $exs \leftarrow \{e : e \in examples$ **and** $e.A = v_k\}$
      $subtree \leftarrow$ DECISION-TREE-LEARNING(exs, $attributes - \{A\}$, $examples$)
      add a branch to $tree$ with label $(A = v_k)$ and subtree $subtree$
    return $tree$

- PLURALITY-VALUE() returns output value for majority of given examples
- IMPORTANCE() rates attributes based on their importance in making decisions for the given set of examples (e.g., using **information gain**)

Decision Tree Learning Algorithm: Another Version

Pseudocode for Decision Tree Learning (my version)

procedure DECISION-TREE-LEARN($\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, with p attributes and y_i discrete)
  $root \leftarrow$ **new** NODE(DATA $= \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, ATTRIBUTES $= \{1, 2, \ldots, p\}$)
  SPLIT($root$)

procedure SPLIT($node$)
  $J \leftarrow node$.ATTRIBUTES ▷ relabeling for convenience
  if $node$.DATA consists of a single class **or** $J = \emptyset$ **then return**
  $j \leftarrow \arg\max_{j' \in J}$ IMPORTANCE(j', $node$.DATA)
  for each value v of attribute j **do**
    $exs \leftarrow \{(\mathbf{x}_i, y_i) \in node.\text{DATA} \mid x_{ij} = v\}$
    if $exs \neq \emptyset$ **then**
      $child \leftarrow$ **new** NODE(DATA $= exs$, ATTRIBUTES $= J - \{j\}$)
      Add a branch to $node$ with label $(j = v)$ and child node $child$
      SPLIT($child$)

- IMPORTANCE() rates attributes based on their importance in making decisions for the given set of examples (e.g., using **information gain**)

Decision Tree Prediction Algorithm

Pseudocode for Decision Tree Prediction

procedure DECISION-TREE-PREDICT($node, \mathbf{x}$)
  for each $child$ of $node$ with label $(j = v)$ **do**
    if $x_j = v$ **then return** DECISION-TREE-PREDICT($child, \mathbf{x}$)
  return PLURALITY-VALUE($node$.DATA)

- PLURALITY-VALUE() returns output value for majority of given examples

The pseudocode on this slide and the previous one captures the general idea of building and using decision trees, but some things can be improved. In particular, having each node store its subset of data is space-inefficient. With some modifications, we can eliminate this (left as as exercise for the reader).

Lecture 05-3a:
Impact of Heuristics

Information Gain and Other Heuristics

Some remaining issues with attribute selection:

1. What should be done in the case of ties in information gain?
   - Deterministic or random selection?
   - Use another measure as a tie-breaker (e.g., select the attribute that produces the largest number of pure child nodes)
2. Do other measures for comparing attributes do better?

There is **no single correct answer** to either of these questions.

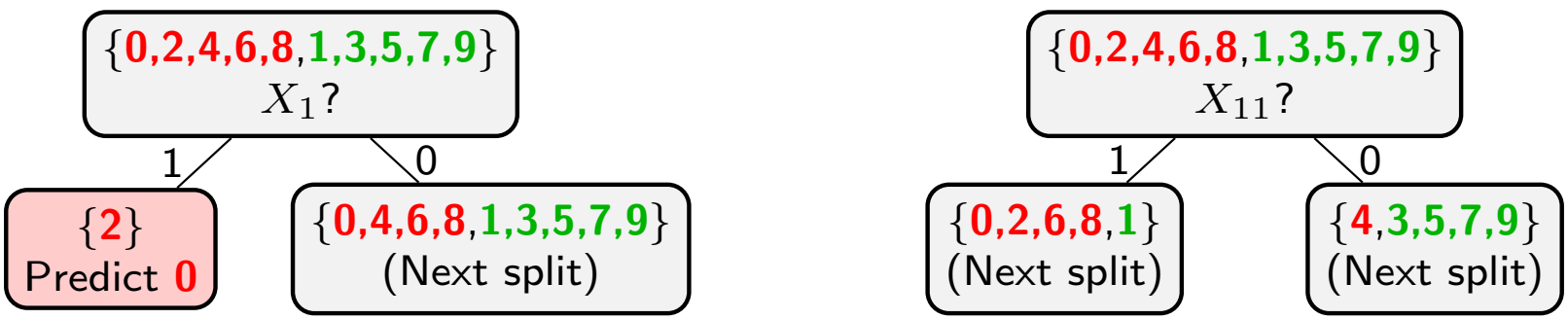
Definition

A **heuristic** is a method or approach to a problem that tends to work well in practice, but is not provably optimal.

Information gain is a **heuristic**! In most cases it works well, but there are situations in which other measures would produce better trees. (Still need some notion of "better" for trees, too though!)

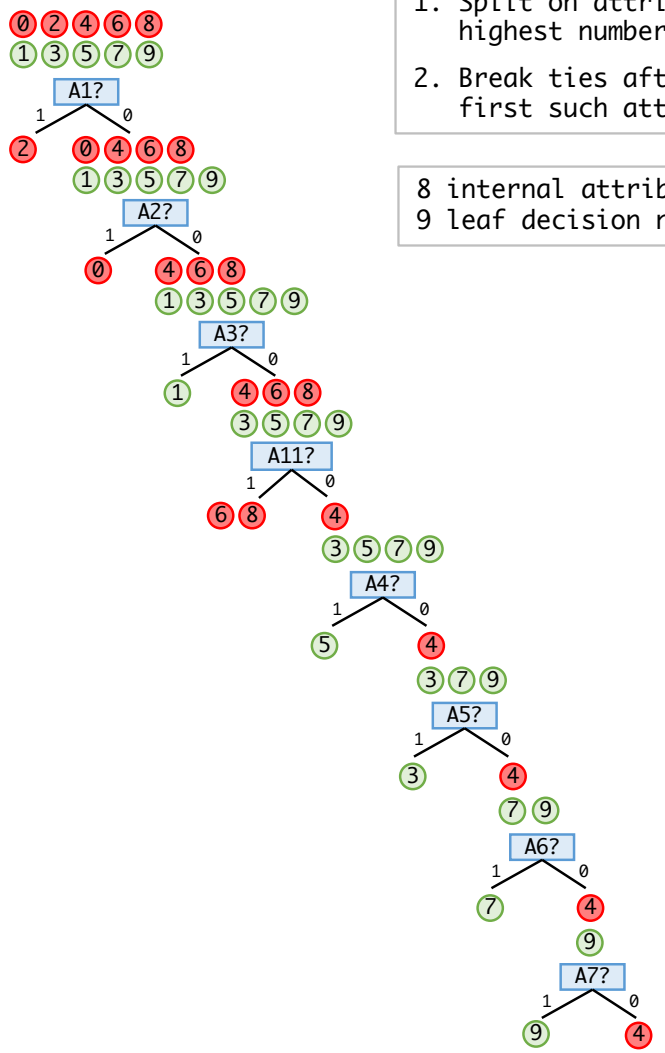
Different Heuristics, Different Trees

Index	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	X_{11}	Y
0	0	1	0	0	0	0	0	0	0	0	1	**0**
1	0	0	1	0	0	0	0	0	0	0	1	**1**
2	1	0	0	0	0	0	0	0	0	0	1	**0**
3	0	0	0	0	1	0	0	0	0	0	0	**1**
4	0	0	0	0	0	0	0	0	0	1	0	**0**
5	0	0	0	1	0	0	0	0	0	0	0	**1**
6	0	0	0	0	0	0	0	0	1	0	1	**0**
7	0	0	0	0	0	1	0	0	0	0	0	**1**
8	0	0	0	0	0	0	0	1	0	0	1	**0**
9	0	0	0	0	0	0	1	0	0	0	0	**1**



Using "Decisiveness" as a Heuristic

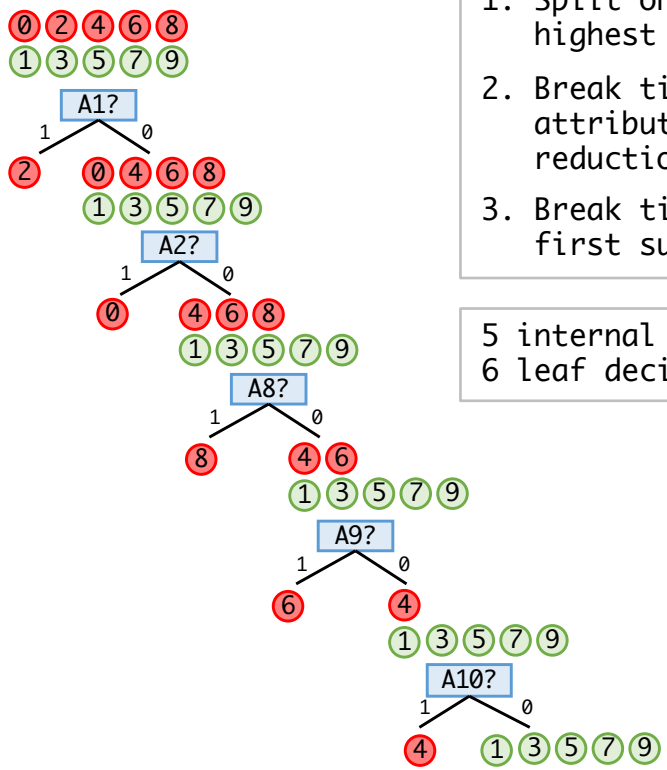
- Suppose, rather than information gain, we always chose a feature that was most decisive (having the largest number of inputs for which it gives a firm output decision)
- On this data-set, and breaking ties by always taking the first such feature in the feature-set, we get a tree as shown



1. Split on attribute that gives us highest number of decided inputs.
2. Break ties after step 1: split on first such attribute encountered.

8 internal attribute nodes,
9 leaf decision nodes (17 total).

Combining "Decisiveness" and Information Gain

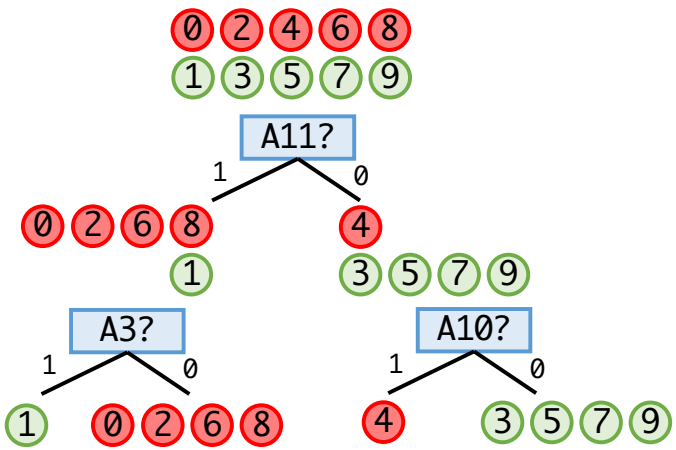


1. Split on attribute that gives us highest number of decided inputs.
2. Break ties after step 1: split on attribute that gives us biggest reduction in overall entropy.
3. Break ties after step 2: split on first such attribute encountered.

5 internal attribute nodes,
6 leaf decision nodes (11 total).

- We can improve things by combining heuristics
- If we choose always the attribute that gives us the most decisive split, but break ties using information gain as a ***secondary*** heuristic, we get a more compact tree (6 total nodes less)

Combining "Decisiveness" and Information Gain



1. Split on attribute that gives us biggest reduction in overall entropy.
2. Break ties after step 2: split on first such attribute encountered.

3 internal attribute nodes,
2 leaf decision nodes (5 total).

- The best result, however, is to simply use information gain (12 less nodes than the original, largest tree, 6 less than the one using combined heuristics)
  - Ties don't actually matter here, so no other heuristic needed
- Is this guaranteed ***always*** to happen?
  - ***No***, but the heuristic is considered useful for the reason that there will be more cases in which it works well than ones where it won't..

Another Heuristic: Gini Impurity

Another measure of quality for any node S is the **Gini impurity** or **Gini index**, denoted $G(S)$:

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

Observations:

- Interpreted as a measure of total variance across the classes.
- Pure nodes have $G(S) = 0$; others have $G(S) > 0$
- Comparable to entropy but computationally simpler (no logarithms); recall that entropy is

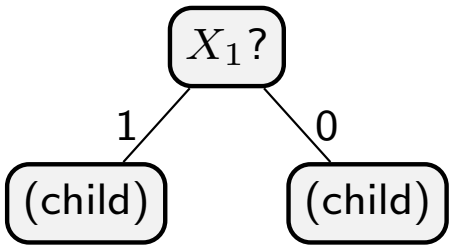
$$H(S) = -\sum_{k=1}^{K} \hat{p}_k(S) \log_2 \hat{p}_k(S)$$

Lecture 05-3b: Decision Trees with Continuous-Valued Attributes

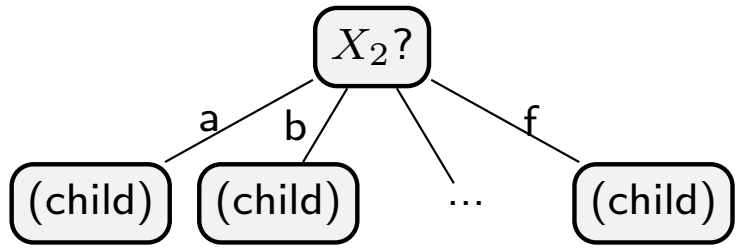
Data for Decision Trees

So far, we have looked at decision trees with binary or discrete inputs:

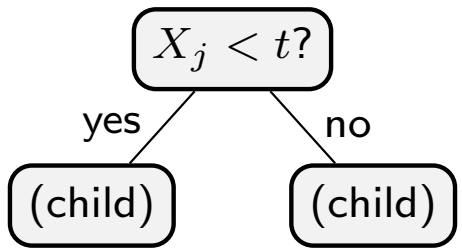
Splitting on binary feature X_1 with values $\{0, 1\}$:



Splitting on discrete feature X_2 with values $\{a, b, c, d, e, f\}$:



For a **continuous-valued** input attribute X_j, we typically use a binary split of the form "$X_j < t$?" for some $t \in \mathbb{R}$:

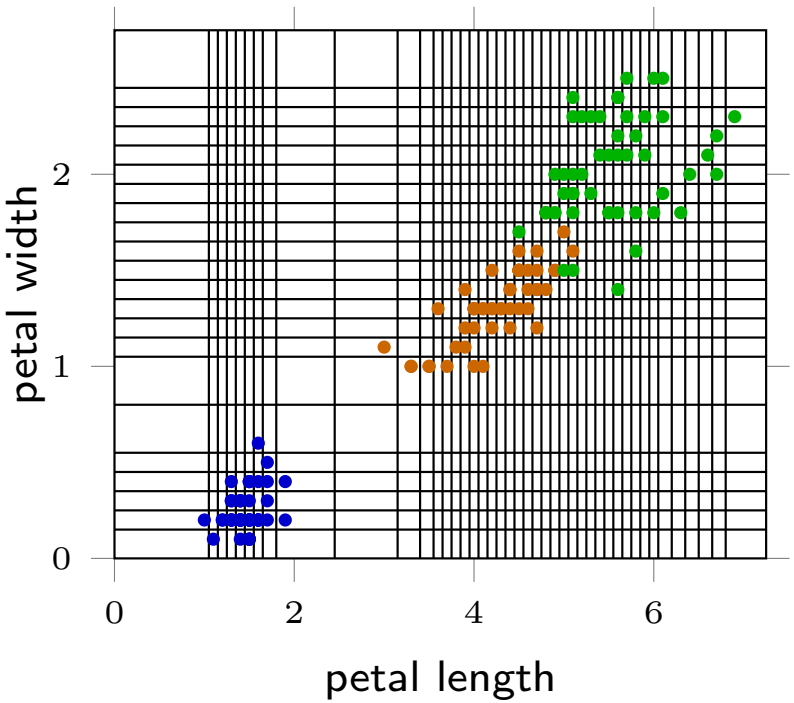


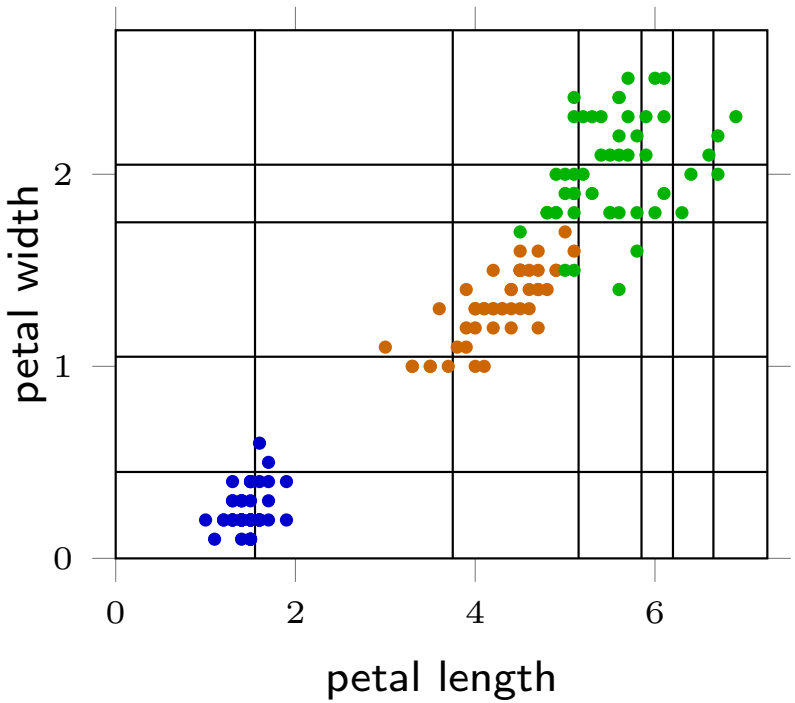
Ways to Select a Split Threshold

With some criterion for measuring the quality of a split (e.g., entropy or Gini impurity), we can choose a split threshold using either:

Best-of-all: Examine all possible attributes and all possible thresholds for a split and take the best one

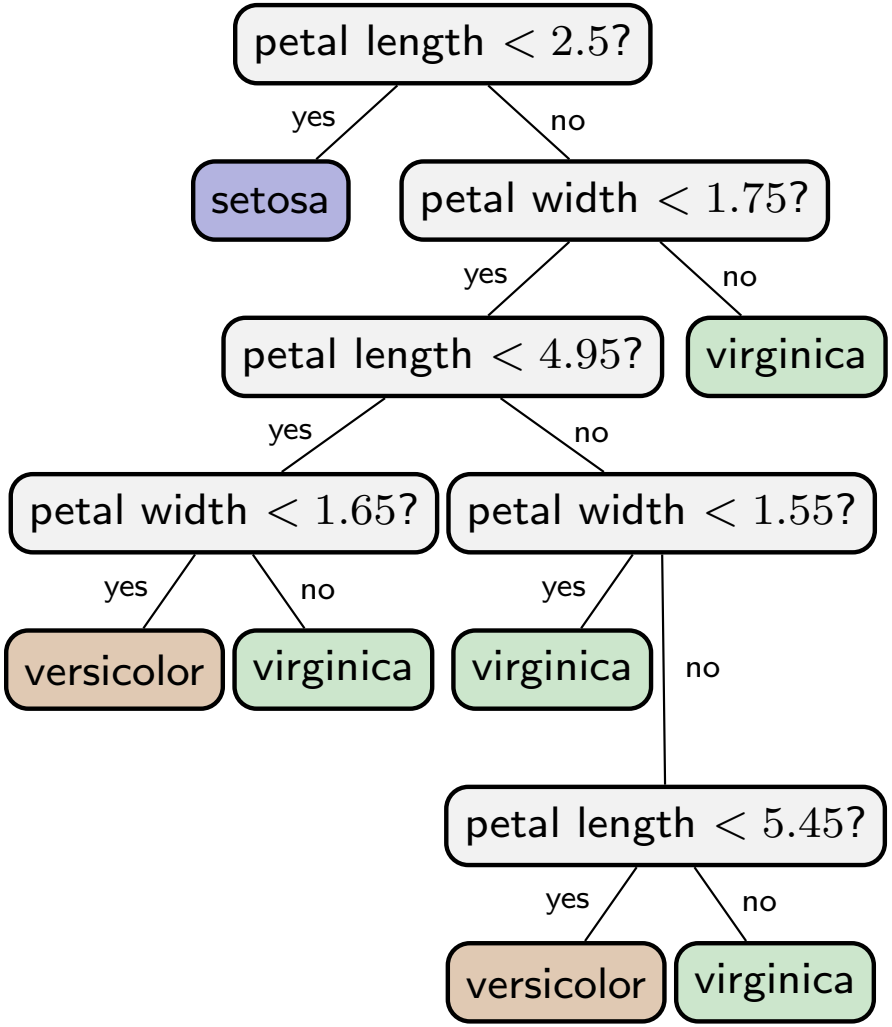
Best-of-subset: Examine a random subset of attributes and thresholds and choose the best split from there

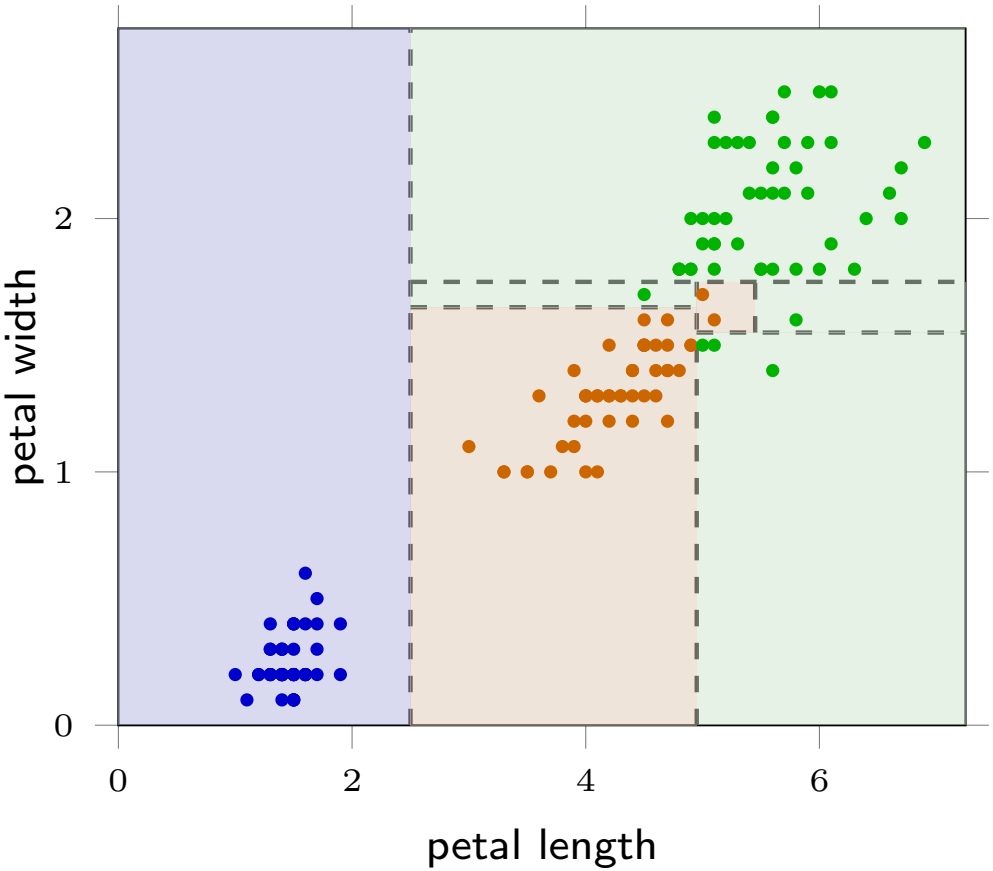




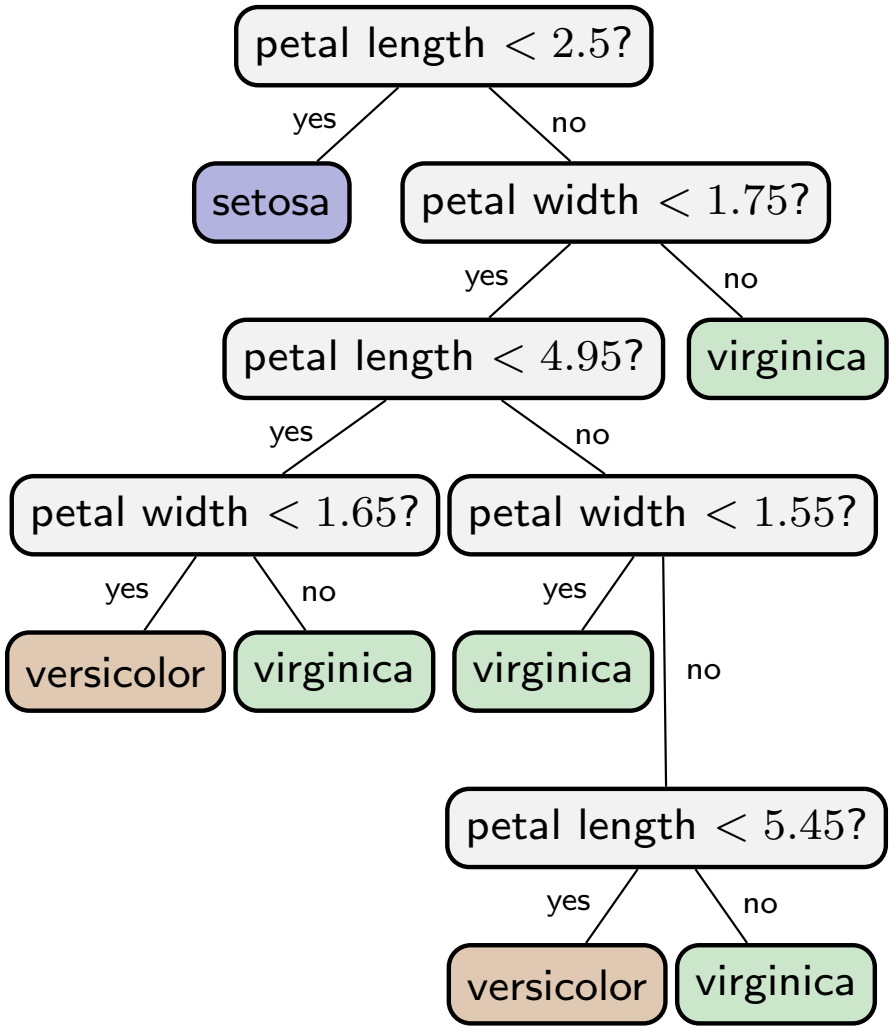
Lecture 05-4a: Decision Trees with Continuous-Valued Attributes

Example: Iris Data





Decision Tree Observations



- Some splits result in **pure nodes** corresponding to regions with points in a single class
- Other splits lead to **impure nodes** which are split further
- Picking splits halfway between points makes sense for generalizing
- There are **many different options** for splitting
- Full tree has **perfect accuracy** on the training set (assuming no identical points in opposite classes, which result in **unsplittable nodes**)

  $\Rightarrow$ Full tree almost certainly **overfits** (low bias, high variance)

Lecture 05-4b:
Decision Trees and Overfitting

Stopping Criteria

When building a decision tree, when should we **stop** splitting a node S?

Some obvious stopping criteria:

- S is **pure** (contains only points from one class)
- S is **unsplittable**
  (e.g., S contains $\mathbf{x}_i = (1, 1, 1), y_i = 0$ and $\mathbf{x}_{i'} = (1, 1, 1), y_{i'} = 1$)

Trees produced with these criteria tend to **overfit**:

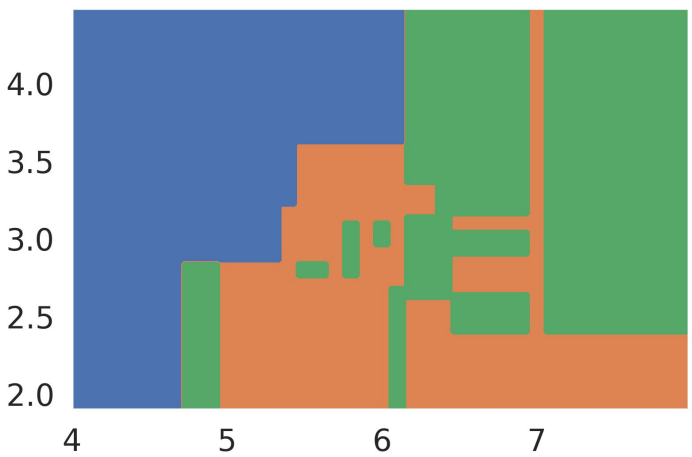


Image source: DS 100 lecture notes

Goals for Constructing Decision Trees

When building a decision tree, we ultimately want:

1. **Good accuracy** on the training set, $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$
2. **Good accuracy** on out-of-sample data

Unfortunately, these two criteria often compete with each other:

- Deep trees can differentiate between most inputs
- Longer paths to reach a decision are less likely to generalize

Instead, we try to **balance** these two goals with heuristics that prevent our trees from getting **too big**.

Dealing with Overfitting

Two general approaches to dealing with **overfitting** during construction:

1. Add **hard constraints** to prevent full growth; e.g.:
   - S is at depth d in tree
   - There are too many leaves in the tree already
     (every split replaces an existing leaf with two new ones)
   - S contains too few data points to split
   - Every split at S produces one or more child nodes with an insufficient number of data points
   - Information gain by splitting is small
     (potentially misses good splits one step ahead)
2. Build the full tree, then **prune back** low-value nodes
   (e.g., cost complexity pruning, χ^2 pruning)

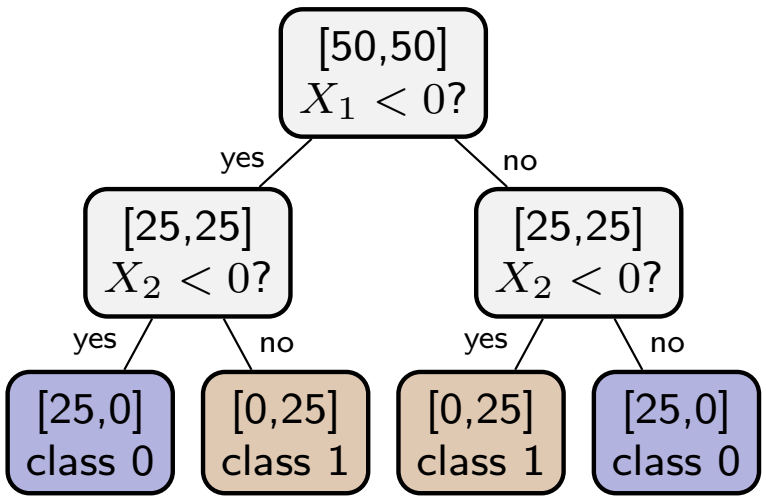
Skiena: "Decision tree construction is hacking, not science."

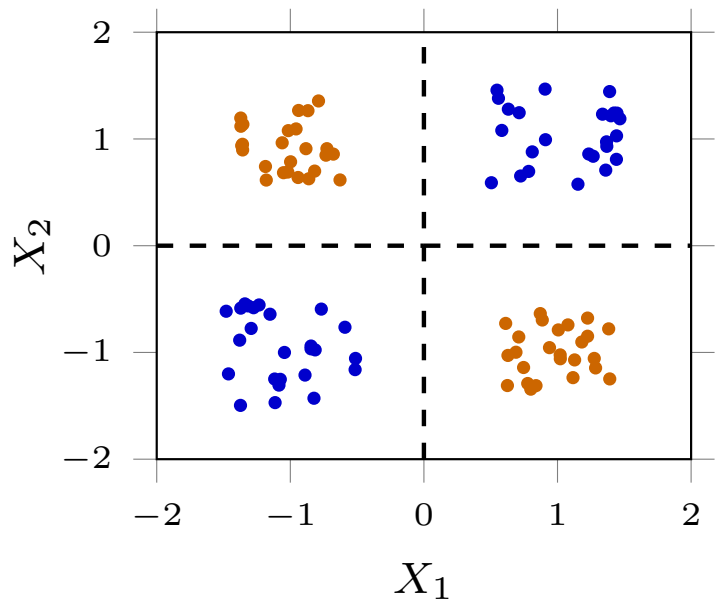
Lecture 05-4c:
Additional Notes on Decision Trees

Issues with the Greedy Constructive Approach

The decision tree construction algorithm is **greedy**: when selecting a split, we only consider the **immediate impact** of the split, not what the split **might allow us to do** further down in the tree.

Example:





- Only 3 splits needed, but first split doesn't yield any information gain.
- Better splits can be found using a look-ahead process (**more work**).

Regression Trees

If the **output** attribute Y is **not discrete**, we have a **regression** problem!

We can adapt the decision tree algorithm to create **regression trees**:

- Same types of attribute splits are used
- At any node S, the predicted value $\hat{y}(S)$ is the average output value at that node: $\hat{y}(S) = \frac{1}{|S|} \sum_{i \in S} y_i$.
- Node quality is measured as a residual sum of squares: $\sum_{i \in S} (y_i - \hat{y}(S))^2$ (variance of output values at the node)
- Splits chosen to minimize the (weighted) residual sums of squares across child nodes (similar to information gain)

Pros and Cons of Decision Trees

Pros

- Easy to interpret
- Non-linear decision boundaries
- Easily supports both categorical and numerical variables
- Insensitive to feature scaling

Cons

- Lack of elegance (math)
- Sensitive to noise in training data (tendency to overfit)
- Decision boundary is axis-aligned
- Finding globally optimal tree is NP-hard
- Does not extrapolate well
- Tend to have lower accuracy compared to other models

Decision Trees vs. Linear Models

Scenario 1 (top row):

- Ideal for logistic regression
- Problematic for decision trees with axis-aligned hyperplanes

Scenario 2 (bottom row):

- Problematic for logistic regression
- Easy for decision trees

Taken from "An Introduction to Statistical Learning, with applications in R" (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani.