CS 4700: Foundations of Artificial Intelligence

Prof. Bart Selman
selman@cs.cornell.edu

Machine Learning: Decision Trees
R&N 18.3
Big Data: Sensors Everywhere

Data collected and stored at enormous speeds (GB/hour)

Cars
Cellphones
Remote Controls
Traffic lights,
ATM machines
Appliances
Motion sensors
Surveillance cameras
etc etc
Data collected and stored at enormous speeds (GB/hour)

- remote sensors on a satellite
- telescopes scanning the skies
- microarrays generating gene expression data
- scientific simulations generating terabytes of data

Traditional statistical techniques infeasible to deal with the data TUSNAMI – they don’t scale up!!!

→ Machine Learning Techniques

(adapted from Vipin Kumar)
Machine Learning Tasks

Prediction Methods
- Use some variables to predict unknown or future values of other variables.

Description Methods
- Find human-interpretable patterns that describe the data.
**Supervised learning:**
We are given a set of examples with the correct answer - classification and regression

**Unsupervised learning:** “just make sense of the data”
Example: Supervised Learning

object recognition

Classification

\[ f(x) \quad \text{giraffe} \quad \text{giraffe} \quad \text{giraffe} \quad \text{llama} \quad \text{llama} \quad \text{llama} \]

Target Function

From: Stuart Russell
Example: Supervised Learning

Object Recognition

Classification

\[ f(x) = \text{giraffe} \quad \text{giraffe} \quad \text{giraffe} \quad \text{llama} \quad \text{llama} \quad \text{llama} \]

Target Function

From: Stuart Russell
Classifying Galaxies

Early

Class:
• Stages of Formation

Intermediate

Attributes:
• Image features,
• Characteristics of light waves received, etc.

Late

Data Size:
• 72 million stars, 20 million galaxies
• Object Catalog: 9 GB
• Image Database: 150 GB

Courtesy: http://aps.umn.edu
Supervised learning: curve fitting
Regression
Supervised learning: curve fitting
Regression
Supervised learning: curve fitting
Regression
Supervised learning: curve fitting

Regression
Supervised learning: curve fitting
Regression
Unsupervised Learning: Clustering


Each ecoregion is a different random color. Blue filled circles mark locations most representative of mean conditions of each region.

Ecoregion Analysis of Alaska using clustering

In **classification** – inputs belong two or more classes. 
Goal: the learner must produce *a model that assigns unseen inputs* to one (or **multi-label classification**) or more of these classes. Typically supervised learning.
- Example –  
  - Spam filtering is an example of classification, where the inputs are email (or other) messages and the classes are "spam" and "not spam".

In **regression**, also typically supervised, the outputs are continuous rather than discrete.

In **clustering**, a set of inputs is to be divided into groups. Typically done in an **unsupervised** way (i.e., no labels, the groups are not known beforehand).
Supervised learning: Big Picture

Goal: To learn an unknown target function $f$

Input: a training set of labeled examples $(x_j, y_j)$ where $y_j = f(x_j)$
- E.g., $x_j$ is an image, $f(x_j)$ is the label “giraffe”
- E.g., $x_j$ is a seismic signal, $f(x_j)$ is the label “explosion”

Output: hypothesis $h$ that is “close” to $f$, i.e., predicts well on unseen examples ("test set")

Many possible hypothesis families for $h$
- Linear models, logistic regression, neural networks, support vector machines, decision trees, examples (nearest-neighbor), grammars, kernelized separators, etc etc
Big Picture of Supervised Learning

Learning can be seen as fitting a function to the data. We can consider different target functions and therefore different hypothesis spaces.

Examples:
- Propositional if-then rules
- Decision Trees
- First-order if-then rules
- First-order logic theory
- Linear functions
- Polynomials of degree at most k
- Neural networks
- Java programs
- Turing machine
- Etc

A learning problem is realizable if its hypothesis space contains the true function.

Tradeoff between expressiveness of a hypothesis space and the complexity of finding simple, consistent hypotheses within the space.
Can we learn how counties vote?

New York Times
April 16, 2008

Decision Trees: a sequence of tests. Representation very natural for humans. Style of many “How to” manuals and trouble-shooting procedures.

In the nominating contests so far, Senator Barack Obama has won the vast majority of counties with large black or highly educated populations. Senator Hillary Rodham Clinton has a commanding lead in less-educated counties dominated by whites. Follow the arrows for a more detailed split.

Is a county more than 20 percent black?

NO There are not many African-Americans in this county.
YES The county has a large African-American population.

And is the high school graduation rate higher than 76 percent?

NO This is a county with less-educated voters.
YES This is a county with more educated voters.

Clinton wins these counties 704 to 89.

And where is the county?
Northeast or South
West or Mid West

Clinton wins these counties
182 to 79.

In 2000, were many households poor?

YES At least 47% earned less than $30,000.
NO At least 53% earned more than $30,000.

Clinton wins these counties
52 to 25.

What’s the population density?

Very rural

>61.5 people per sq mile

Obama wins these counties
201 to 83.

In 2004, did Bush beat Kerry badly? (by more than 16.5 percentage points)

YES

Obama wins these counties
56 to 35.

NO

Clinton wins these counties
48 to 13.

Note: Chart excludes Florida and Michigan. County-level results are not available in Alaska, Hawaii, Kansas, Nebraska, New Mexico, North Dakota, or Maine. Texas counties are included twice: once for primary voters and once for caucus participants.
In the nominating contests so far, Senator Barack Obama has won the vast majority of counties with large black or highly educated populations. Senator Hillary Rodham Clinton has a commanding lead in less-educated counties dominated by whites. Follow the arrows for a more detailed split.

**Note: order of tests matters (in general)!

![Diagram of decision tree showing the Obama-Clinton divide.](image-url)
Decision tree learning approach can construct tree (with test thresholds) from example counties.
Decision Tree Learning
Decision Tree Learning

Task:

– Given: collection of examples (x, f(x))
– Return: a function $h$ (*hypothesis*) that approximates $f$
– $h$ is a *decision tree*

**Input:** an object or situation described by a set of attributes (or features)
**Output:** a “decision” – the predicts output value for the input.

The input attributes and the outputs can be *discrete* or *continuous*.

We will focus on decision trees for **Boolean classification:**
each example is classified as *positive* or *negative.*
What is a decision tree?

A tree with two types of nodes:

- Decision nodes
- Leaf nodes

**Decision node:** Specifies a choice or test of some attribute with 2 or more alternatives; every decision node is part of a path to a leaf node.

**Leaf node:** Indicates classification of an example.
### Instance Space X:
Set of all possible objects described by attributes (often called features).

### Target Function f:
Mapping from Attributes to Target Feature (often called label) \((f \text{ is unknown})\)

### Hypothesis Space H:
Set of all classification rules \(h_i\) we allow.

### Training Data D:
Set of instances labeled with Target Feature

<table>
<thead>
<tr>
<th>Food (3)</th>
<th>Chat (2)</th>
<th>Fast (2)</th>
<th>Price (3)</th>
<th>Bar (2)</th>
<th>BigTip</th>
</tr>
</thead>
<tbody>
<tr>
<td>great</td>
<td>yes</td>
<td>yes</td>
<td>normal</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>great</td>
<td>no</td>
<td>yes</td>
<td>normal</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>mediocre</td>
<td>yes</td>
<td>no</td>
<td>high</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>great</td>
<td>yes</td>
<td>yes</td>
<td>normal</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

Etc.
Is the decision tree we learned consistent?

Yes, it agrees with all the examples!

Data: Not all $2 \times 2 \times 3 = 12$ tuples
Also, some repeats! These are literally “observations.”
Learning decision trees:
Another example (waiting at a restaurant)

Problem: decide whether to wait for a table at a restaurant. What attributes would you use?

Attributes used by R&N
1. Alternate: is there an alternative restaurant nearby?
2. Bar: is there a comfortable bar area to wait in?
3. Fri/Sat: is today Friday or Saturday?
4. Hungry: are we hungry?
5. Patrons: number of people in the restaurant (None, Some, Full)
6. Price: price range ($, $$, $$$)
7. Raining: is it raining outside?
8. Reservation: have we made a reservation?
9. Type: kind of restaurant (French, Italian, Thai, Burger)
10. WaitEstimate: estimated waiting time (0-10, 10-30, 30-60, >60)

Goal predicate: WillWait?

What about restaurant name?

What about restaurant name?

It could be great for generating a small tree but …

It doesn’t generalize!
**Attribute-based representations**

Examples described by **attribute values** (Boolean, discrete, continuous)

E.g.

<table>
<thead>
<tr>
<th>Example</th>
<th>Alt</th>
<th>Bar</th>
<th>Fri</th>
<th>Hun</th>
<th>Pat</th>
<th>Price</th>
<th>Rain</th>
<th>Res</th>
<th>Type</th>
<th>Est</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_1$</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>Some</td>
<td>$$$</td>
<td>F</td>
<td>T</td>
<td>French</td>
<td>0–10</td>
<td>T</td>
</tr>
<tr>
<td>$X_2$</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>Full</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Thai</td>
<td>30–60</td>
<td>F</td>
</tr>
<tr>
<td>$X_3$</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>Some</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Burger</td>
<td>0–10</td>
<td>T</td>
</tr>
<tr>
<td>$X_4$</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>Full</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Thai</td>
<td>10–30</td>
<td>T</td>
</tr>
<tr>
<td>$X_5$</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>Full</td>
<td>$$$</td>
<td>F</td>
<td>T</td>
<td>French</td>
<td>&gt;60</td>
<td>F</td>
</tr>
<tr>
<td>$X_6$</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>Some</td>
<td>$</td>
<td>T</td>
<td>T</td>
<td>Italian</td>
<td>0–10</td>
<td>T</td>
</tr>
<tr>
<td>$X_7$</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>None</td>
<td>$</td>
<td>T</td>
<td>F</td>
<td>Burger</td>
<td>0–10</td>
<td>F</td>
</tr>
<tr>
<td>$X_8$</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>Some</td>
<td>$</td>
<td>T</td>
<td>T</td>
<td>Thai</td>
<td>0–10</td>
<td>T</td>
</tr>
<tr>
<td>$X_9$</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>Full</td>
<td>$</td>
<td>T</td>
<td>F</td>
<td>Burger</td>
<td>&gt;60</td>
<td>F</td>
</tr>
<tr>
<td>$X_{10}$</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>Full</td>
<td>$$$</td>
<td>F</td>
<td>T</td>
<td>Italian</td>
<td>10–30</td>
<td>F</td>
</tr>
<tr>
<td>$X_{11}$</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>None</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Thai</td>
<td>0–10</td>
<td>F</td>
</tr>
<tr>
<td>$X_{12}$</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>Full</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Burger</td>
<td>30–60</td>
<td>T</td>
</tr>
</tbody>
</table>

12 examples

6 +

6 -

Classification of examples is positive (T) or negative (F)
Decision trees

One possible representation for hypotheses
E.g., here is a tree for deciding whether to wait:
Decision tree learning Algorithm

Decision trees can express any Boolean function.

Goal: Finding a decision tree that agrees with training set.

We could construct a decision tree that has one path to a leaf for each example, where the path tests each attribute value to the value of the example.

What is the problem with this from a learning point of view?

**Problem:** This approach would just memorize example.

How to deal with new examples? It doesn’t generalize!

(But sometimes hard to avoid --- e.g. parity function, 1, if an even number of inputs, or majority function, 1, if more than half of the inputs are 1).

We want a compact/smallest tree.

But finding the smallest tree consistent with the examples is NP-hard!

Overall Goal: get a good classification with a small number of tests.
Basic DT Learning Algorithm

Goal: find a *small* tree consistent with the training examples

Idea: (recursively) choose "*most significant*" attribute as root of (sub)tree;
Use a *top-down greedy search* through the space of possible decision trees.
Greedy because there is *no backtracking*. It picks highest values first.

Variations of known algorithms ID3, C4.5 (Quinlan -86, -93)

Top-down greedy construction
- Which *attribute* should be tested? (ID3 Iterative Dichotomiser 3)
  - Heuristics and Statistical testing with current data
- Repeat for descendants
Big Tip Example

10 examples:

6+ 1 3 4 7 8 10

4- 2 5 6 9

Attributes:
• Food with values g, m, y
• Speedy? with values y, n
• Price, with values a, h

Let’s build our decision tree starting with the attribute Food, (3 possible values: g, m, y).
Top-Down Induction of Decision Tree: Big Tip Example

10 examples:

- Food: y [6+] [4-]
- Speedy: n [3+] [1-]
- Price: h [4+] [2-]

Node “done” when uniform label, “no further Uncertainty,” or no features left.

How many + and - examples per subclass, starting with y?

Let’s consider next the attribute Speedy.

<table>
<thead>
<tr>
<th>F</th>
<th>S</th>
<th>P</th>
<th>BigTip</th>
</tr>
</thead>
<tbody>
<tr>
<td>g</td>
<td>y</td>
<td>a</td>
<td>( f(\vec{x}_1) = 1 )</td>
</tr>
<tr>
<td>g</td>
<td>n</td>
<td>h</td>
<td>( f(\vec{x}_2) = 0 )</td>
</tr>
<tr>
<td>g</td>
<td>y</td>
<td>h</td>
<td>( f(\vec{x}_3) = 1 )</td>
</tr>
<tr>
<td>g</td>
<td>n</td>
<td>a</td>
<td>( f(\vec{x}_4) = 1 )</td>
</tr>
<tr>
<td>m</td>
<td>y</td>
<td>a</td>
<td>( f(\vec{x}_5) = 0 )</td>
</tr>
<tr>
<td>y</td>
<td>y</td>
<td>a</td>
<td>( f(\vec{x}_6) = 0 )</td>
</tr>
<tr>
<td>g</td>
<td>y</td>
<td>a</td>
<td>( f(\vec{x}_7) = 1 )</td>
</tr>
<tr>
<td>g</td>
<td>y</td>
<td>h</td>
<td>( f(\vec{x}_8) = 1 )</td>
</tr>
<tr>
<td>m</td>
<td>y</td>
<td>a</td>
<td>( f(\vec{x}_9) = 0 )</td>
</tr>
<tr>
<td>g</td>
<td>y</td>
<td>a</td>
<td>( f(\vec{x}_{10}) = 1 )</td>
</tr>
</tbody>
</table>
Top-Down Induction of DT (simplified)

IF(all examples in D have same class c)
   - Return leaf with class c (or class c_{def}, if D is empty)

ELSE IF(no attributes left to test)
   - Return leaf with class c of majority in D

ELSE
   - Pick A as the “best” decision attribute for next node
   - FOR each value v_i of A create a new descendent of node
     - D_i = {(\vec{x}, y) ∈ D : attribute A of \vec{x} has value v_i}
     - Subtree t_i for v_i is TDIDT(D_i, c_{def})
   - RETURN tree with A as root and t_i as subtrees

Training Data:  D = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n)\}
Picking the Best Attribute to Split

Ockham’s Razor:
  – All other things being equal, choose the simplest explanation

Decision Tree Induction:
  – Find the smallest tree that classifies the training data correctly

Problem
  – Finding the smallest tree is computationally hard 😞!

Approach
  – Use heuristic search (greedy search)

Key Heuristics:
  – Pick attribute that maximizes information (Information Gain)
    i.e. “most informative”
  – Other statistical tests
Attribute-based representations

Examples described by attribute values (Boolean, discrete, continuous)

E.g.

<table>
<thead>
<tr>
<th>Example</th>
<th>Attributes</th>
<th>Target</th>
<th>Wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>X₁</td>
<td>Alt Bar Fri Hun Pat Price Rain Res Type Est</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>X₂</td>
<td>T F F T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>X₃</td>
<td>F T F F</td>
<td>Full</td>
<td>$</td>
</tr>
<tr>
<td>X₄</td>
<td>T F T T</td>
<td>Some</td>
<td>$</td>
</tr>
<tr>
<td>X₅</td>
<td>T F T F</td>
<td>Full</td>
<td>$$$</td>
</tr>
<tr>
<td>X₆</td>
<td>F T F T</td>
<td>Some</td>
<td>$</td>
</tr>
<tr>
<td>X₇</td>
<td>F T F F</td>
<td>None</td>
<td>$</td>
</tr>
<tr>
<td>X₈</td>
<td>F F F T</td>
<td>Some</td>
<td>$</td>
</tr>
<tr>
<td>X₉</td>
<td>F T T F</td>
<td>Full</td>
<td>$</td>
</tr>
<tr>
<td>X₁₀</td>
<td>T T T T</td>
<td>Full</td>
<td>$$$</td>
</tr>
<tr>
<td>X₁₁</td>
<td>F F F F</td>
<td>None</td>
<td>$</td>
</tr>
<tr>
<td>X₁₂</td>
<td>T T T T</td>
<td>Full</td>
<td>$</td>
</tr>
</tbody>
</table>

Classification of examples is positive (T) or negative (F)
Choosing an attribute: Information Gain

Goal: trees with short paths to leaf nodes

Is this a good attribute to split on? Which one should we pick?

A perfect attribute would ideally divide the examples into sub-sets that are all positive or all negative… i.e. maximum information gain.
Information Gain

Most useful in classification
- how to measure the ‘worth’ of an attribute *information gain*
- how well attribute separates examples according to their classification

Next
- precise definition for gain

→ measure from Information Theory
Shannon and Weaver 49

One of the most successful and impactful mathematical theories known.
“Information” answers questions. Entropy is a measure of unpredictability of information content.

The more clueless I am about a question, the more information the answer to the question contains.

Example – fair coin $\rightarrow$ prior $\langle 0.5, 0.5 \rangle$

By definition Information of the prior (or entropy of the prior)

$$ I(P1,P2) = - P1 \log_2(P1) - P2 \log_2(P2) = $$

$$ I(0.5,0.5) = -0.5 \log_2(0.5) - 0.5 \log_2(0.5) = 1 $$

We need 1 bit to convey the outcome of the flip of a fair coin.

Scale: 1 bit = answer to Boolean question with prior $\langle 0.5, 0.5 \rangle$

Why does a biased coin have less information?
Information in an answer given possible answers $v_1, v_2, \ldots v_n$:

$$I(P(v_1), \ldots, P(v_n)) = \sum_{i=1}^{n} -P(v_i) \log_2(P(v_i))$$

— $v_1, \ldots, v_n$ possible answers
— $P(v_i)$ probability of answer $v_i$

(Also called entropy of the prior.)

Example – biased coin $\Rightarrow$ prior $<1/100, 99/100>$

$$I(1/100, 99/100) = -1/100 \log_2(1/100) - 99/100 \log_2(99/100)$$

$= 0.08$ bits (so not much information gained from “answer.”)

Example – fully biased coin $\Rightarrow$ prior $<1, 0>$

$$I(1, 0) = -1 \log_2(1) - 0 \log_2(0) = 0 \text{ bits}$$

$$0 \log_2(0) = 0$$

i.e., no uncertainty left in source!
The more uniform the probability distribution, the greater is its entropy.
Information or Entropy measures the “randomness” of an arbitrary collection of examples.

We don’t have exact probabilities but our training data provides an estimate of the probabilities of positive vs. negative examples given a set of values for the attributes.

For a collection S, entropy is given as:

$$I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) = -\frac{p}{p+n} \log_2\left(\frac{p}{p+n}\right) - \frac{n}{p+n} \log_2\left(\frac{n}{p+n}\right)$$

For a collection S having positive and negative examples

- $p$ - # positive examples;
- $n$ - # negative examples
Attribute-based representations

Examples described by attribute values (Boolean, discrete, continuous)
E.g., situations where I will/won't wait for a table:

<table>
<thead>
<tr>
<th>Example</th>
<th>Alt</th>
<th>Bar</th>
<th>Fri</th>
<th>Hun</th>
<th>Pat</th>
<th>Price</th>
<th>Rain</th>
<th>Res</th>
<th>Type</th>
<th>Est</th>
<th>Wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>X₁</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>Some</td>
<td>$$$</td>
<td>F</td>
<td>T</td>
<td>French</td>
<td>0–10</td>
<td>T</td>
</tr>
<tr>
<td>X₂</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>Full</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Thai</td>
<td>30–60</td>
<td>F</td>
</tr>
<tr>
<td>X₃</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>Some</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Burger</td>
<td>0–10</td>
<td>T</td>
</tr>
<tr>
<td>X₄</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>Full</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Thai</td>
<td>10–30</td>
<td>T</td>
</tr>
<tr>
<td>X₅</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>Full</td>
<td>$$$</td>
<td>F</td>
<td>T</td>
<td>French</td>
<td>&gt;60</td>
<td>F</td>
</tr>
<tr>
<td>X₆</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>Some</td>
<td>$$</td>
<td>T</td>
<td>T</td>
<td>Italian</td>
<td>0–10</td>
<td>T</td>
</tr>
<tr>
<td>X₇</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>None</td>
<td>$</td>
<td>T</td>
<td>F</td>
<td>Burger</td>
<td>0–10</td>
<td>F</td>
</tr>
<tr>
<td>X₈</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>Some</td>
<td>$$</td>
<td>T</td>
<td>T</td>
<td>Thai</td>
<td>0–10</td>
<td>T</td>
</tr>
<tr>
<td>X₉</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>Full</td>
<td>$</td>
<td>T</td>
<td>F</td>
<td>Burger</td>
<td>&gt;60</td>
<td>F</td>
</tr>
<tr>
<td>X₁₀</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>Full</td>
<td>$$$</td>
<td>F</td>
<td>T</td>
<td>Italian</td>
<td>10–30</td>
<td>F</td>
</tr>
<tr>
<td>X₁₁</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>None</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Thai</td>
<td>0–10</td>
<td>F</td>
</tr>
<tr>
<td>X₁₂</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>Full</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Burger</td>
<td>30–60</td>
<td>T</td>
</tr>
</tbody>
</table>

12 examples
6 +
6 -

What’s the entropy of this collection of examples?

Classification of examples is positive (T) or negative (F)

\[ p = n = 6; \ I(0.5,0.5) = -0.5 \log_2(0.5) -0.5 \log_2(0.5) = 1 \]

So, we need 1 bit of info to classify a randomly picked example, assuming no other information is given about the example.
Choosing an attribute: Information Gain

Intuition: Pick the attribute that reduces the entropy (the uncertainty) the most.

So we measure the information gain after testing a given attribute $A$:

$$Gain(A) = I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) - \text{Remainder}(A)$$

Remainder($A$) → gives us the remaining uncertainty after getting info on attribute $A$. 
Choosing an attribute: Information Gain

Remainder(A)

→ gives us the amount information we still need after testing on A.

Assume A divides the training set E into E_1, E_2, … E_v, corresponding to the different v distinct values of A.

Each subset E_i has p_i positive examples and n_i negative examples.

So for total information content, we need to weigh the contributions of the different subclasses induced by A

\[
Remainder(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p + n} \cdot I\left(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i}\right)
\]
Choosing an attribute: Information Gain

Measures the expected reduction in entropy. The higher the Information Gain (IG), or just Gain, with respect to an attribute \( A \), the more is the expected reduction in entropy.

\[
Gain(S, A) = Entropy(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \text{Entropy}(S_v)
\]

where \( \text{Values}(A) \) is the set of all possible values for attribute \( A \), \( S_v \) is the subset of \( S \) for which attribute \( A \) has value \( v \).
Interpretations of gain

Gain(S, A)

- expected reduction in entropy caused by knowing A
- information provided about the target function value given the value of A
- number of bits saved in the coding a member of S knowing the value of A

Used in ID3 (Iterative Dichotomiser 3) Ross Quinlan
Information gain

For the training set, \( p = n = 6 \), \( I(6/12, 6/12) = 1 \) bit

Consider the attributes \( \text{Type} \) and \( \text{Patrons} \):

\[
IG(\text{Type}) = 1 - \left[ \frac{2}{12} I\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{2}{12} I\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{4}{12} I\left(\frac{2}{4}, \frac{2}{4}\right) + \frac{4}{12} I\left(\frac{2}{4}, \frac{2}{4}\right) \right] = 0 \text{ bits}
\]

\[
IG(\text{Patrons}) = 1 - \left[ \frac{2}{12} I(0,1) + \frac{4}{12} I(1,0) + \frac{6}{12} I\left(\frac{2}{6}, \frac{4}{6}\right) \right] = 0.541 \text{ bits}
\]

\( \text{Patrons} \) has the highest IG of all attributes and so is chosen by the DTL algorithm as the root.
Example contd.

Decision tree learned from the 12 examples:

Substantially simpler than “true” tree --- but a more complex hypothesis isn’t justified from just the data.
Expressiveness of Decision Trees

Any particular decision tree hypothesis for WillWait goal predicate can be seen as a disjunction of a conjunction of tests, i.e., an assertion of the form:

$$\forall s \text{ WillWait}(s) \leftrightarrow (P_1(s) \lor P_2(s) \lor \ldots \lor P_n(s))$$

Where each condition $P_i(s)$ is a conjunction of tests corresponding to the path from the root of the tree to a leaf with a positive outcome.
Decision trees can express any Boolean function of the input attributes. E.g., for Boolean functions, truth table row → path to leaf:

\[ \begin{array}{ccc}
\text{A} & \text{B} & \text{A xor B} \\
F & F & F \\
F & T & T \\
T & F & T \\
T & T & F \\
\end{array} \]
Expressiveness:
Boolean Function with 2 attributes $\Rightarrow 2^{2^2}$ DTs
Expressiveness: 2 attribute $\Rightarrow 2^2$ DTs
Expressiveness:
2 attribute $\Rightarrow 2^2$ DTs
Expressiveness:
2 attribute $\Rightarrow 2^2$ DTs

A AND-NOT B

NOT A AND B

B

TRUE

A OR NOT B

NOR A OR B

NOT B

FALSE
How many distinct decision trees with 10 Boolean attributes?

= number of Boolean functions with 10 propositional symbols

How many entries does this table have?

\[ 2^{10} \]

So how many Boolean functions with 10 Boolean attributes are there, given that each entry can be 0/1?

\[ = 2^{2^{10}} \]
Hypothesis spaces

How many distinct decision trees with \( n \) Boolean attributes?

\[ = \text{number of Boolean functions} \]

\[ = \text{number of distinct truth tables with } 2^n \text{ rows} \quad = 2^{2^n} \]

E.g. how many Boolean functions on 6 attributes? A lot…

With 6 Boolean attributes, there are 18,446,744,073,709,551,616 possible trees!

Google's calculator could not handle 10 attributes 😊!
Evaluation Methodology
General for Machine Learning
Evaluation Methodology

How to evaluate the quality of a learning algorithm, i.e.,:

How good are the hypotheses produce by the learning algorithm?
How good are they at classifying unseen examples?

Standard methodology ("Holdout Cross-Validation"):

1. Collect a large set of examples.
2. Randomly divide collection into two disjoint sets: training set and test set.
3. Apply learning algorithm to training set generating hypothesis \( h \)
4. Measure performance of \( h \) w.r.t. test set (a form of cross-validation)
   \( \rightarrow \) measures generalization to unseen data

Important: keep the training and test sets disjoint! "No peeking"!

Note: The first two questions about any learning result: Can you describe your training and your test set? What’s your error on the test set?
Test/Training Split

Real-world Process

drawn randomly

split randomly

Data $D$

split randomly

Training Data $D_{\text{train}}$

$((x_1, y_1), \ldots, (x_n, y_n))$

Test Data $D_{\text{test}}$

$((x_1, y_1), \ldots, (x_k, y_k))$

Also validation set for meta-parameters.
Measuring Prediction Performance

Definition: The training error $\text{Err}_{D_{\text{train}}}(h)$ on training data $D_{\text{train}} = ((\tilde{x}_1, y_1), \ldots, (\tilde{x}_n, y_n))$ of a hypothesis $h$ is $\text{Err}_{D_{\text{train}}}(h) = \frac{1}{n} \sum_{i=1}^{n} \Delta(h(\tilde{x}_i), y_i)$.

Definition: The test error $\text{Err}_{D_{\text{test}}}(h)$ on test data $D_{\text{test}} = ((\tilde{x}_1, y_1), \ldots, (\tilde{x}_k, y_k))$ of a hypothesis $h$ is $\text{Err}_{D_{\text{test}}}(h) = \frac{1}{k} \sum_{i=1}^{k} \Delta(h(\tilde{x}_i), y_i)$.

Definition: The prediction/generalization/true error $\text{Err}_{P}(h)$ of a hypothesis $h$ for a learning task $P(X,Y)$ is $\text{Err}_{P}(h) = \sum_{\tilde{x} \in X, y \in Y} \Delta(h(\tilde{x}), y) P(X = \tilde{x}, Y = y)$.

Definition: The zero/one-loss function $\Delta(a,b)$ returns 1 if $a \neq b$ and 0 otherwise.
Performance Measures

Error Rate
- Fraction (or percentage) of false predictions

Accuracy
- Fraction (or percentage) of correct predictions

Precision/Recall

Example: binary classification problems (classes pos/neg)
- Precision: Fraction (or percentage) of correct predictions among all examples predicted to be positive
- Recall: Fraction (or percentage) of correct predictions among all real positive examples

(Can be generalized to multi-class case.)
Extensions of the Decision Tree Learning Algorithm

Noisy data

Overfitting and Model Selection

Cross Validation

Missing Data (R&N, Section 18.3.6)

Using gain ratios (R&N, Section 18.3.6)

Real-valued data (R&N, Section 18.3.6)

Generation of rules and pruning

DT Ensembles

Regression DT
How well does it work?

Many case studies have shown that decision trees are at least as accurate as human experts.

- A study for diagnosing breast cancer had humans correctly classifying the examples 65% of the time, and the decision tree classified 72% correct.
- British Petroleum designed a decision tree for gas-oil separation for offshore oil platforms that replaced an earlier rule-based expert system.
- Cessna designed an airplane flight controller using 90,000 examples and 20 attributes per example.
Bird Distributions
Machine Learning and Citizen Science

State of the Birds Report
(officially released by Secretary of Interior)

Novel Approaches
To Conservation
Based on eBird Models

Distribution Models for
400+ species with
weekly estimates at fine spatial
resolution (3km²)

Adaptive Spatio-Temporal
Machine Learning
Models and Algorithms
(STEM Models)

Boosted Regression DT Ensemble

Relate environmental predictors to observed patterns of occurrences and absences

Bird Observations

300K+ volunteer birders
300M+ bird observations
22M+ hours of field work (2500+ years)

Remote Sensing
Weather
Land Cover

Distribution Models,
Revealing, at a fine resolution, Species’ Habitat Preferences

Patterns of occurrence of the Tree Swallow for different months of the year Source: Daniel Fink

80,000+ CPU Hours (~ 10 Years!!!)
Summary: When to use Decision Trees

Instances presented as attribute-value pairs
Method of approximating discrete-valued functions
  Target function has discrete values: classification problems

Robust to noisy data:
  Training data may contain
    – errors
    – missing attribute values
Typical bias: prefer smaller trees (Ockham's razor)

Widely used, practical and easy to interpret results
Inducing decision trees is one of the most widely used learning methods in practice.

Can outperform human experts in many problems.

**Strengths** include:
- Fast
- Simple to implement
- Human readable
- Can convert result to a set of easily interpretable rules
- Empirically valid in many commercial products
- Handles noisy data

**Weaknesses** include:
- "Univariate" splits/partitioning using only one attribute at a time so limits types of possible trees
- Large decision trees may be hard to understand
- Requires fixed-length feature vectors
- Non-incremental (i.e., batch method)

Can be a legal requirement! Why?