Decision Trees

- entropy
- information gain
Another Classification Idea

- We learned about linear classification (e.g., logistic regression), and nearest neighbors. Any other idea?
  - Pick an attribute, do a simple test
  - Conditioned on a choice, pick another attribute, do another test
  - In the leaves, assign a class with majority vote
  - Do other branches as well
Another Classification Idea

- Gives *axes aligned decision boundaries*
Decision Tree: Example

width > 6.5cm?

- Yes
  - height > 9.5cm?
    - Yes
      - lemon
    - No
      - orange
  - No
    - height > 6.0cm?
      - Yes
        - lemon
      - No
        - orange
Decision Tree: Classification

Test example

width > 6.5cm?

height > 9.5cm?

height > 6.0cm?

Yes

No

Yes

No

Yes

No
Example with Discrete Inputs

- What if the attributes are discrete?

<table>
<thead>
<tr>
<th>Example</th>
<th>Input Attributes</th>
<th>Goal</th>
<th>WillWait</th>
</tr>
</thead>
<tbody>
<tr>
<td>x₁</td>
<td>Yes No No Yes Some $$$$ No Yes French 0–10</td>
<td>y₁ = Yes</td>
<td></td>
</tr>
<tr>
<td>x₂</td>
<td>Yes No No Yes Full $ No No Thai 30–60</td>
<td>y₂ = No</td>
<td></td>
</tr>
<tr>
<td>x₃</td>
<td>No Yes No No Some $ No No Burger 0–10</td>
<td>y₃ = Yes</td>
<td></td>
</tr>
<tr>
<td>x₄</td>
<td>Yes No Yes Yes Full $ Yes No Thai 10–30</td>
<td>y₄ = Yes</td>
<td></td>
</tr>
<tr>
<td>x₅</td>
<td>Yes No Yes No Full $$$$ No Yes French &gt;60</td>
<td>y₅ = No</td>
<td></td>
</tr>
<tr>
<td>x₆</td>
<td>No Yes No Yes Some $$ Yes Yes Italian 0–10</td>
<td>y₆ = Yes</td>
<td></td>
</tr>
<tr>
<td>x₇</td>
<td>No Yes No No None $ Yes No Burger 0–10</td>
<td>y₇ = No</td>
<td></td>
</tr>
<tr>
<td>x₈</td>
<td>No No No Yes Some $$ Yes Yes Thai 0–10</td>
<td>y₈ = No</td>
<td></td>
</tr>
<tr>
<td>x₉</td>
<td>No Yes Yes No Full $ Yes No Burger &gt;60</td>
<td>y₉ = Yes</td>
<td></td>
</tr>
<tr>
<td>x₁₀</td>
<td>Yes Yes Yes Yes Full $$$$ No Yes Italian 10–30</td>
<td>y₁₀ = No</td>
<td></td>
</tr>
<tr>
<td>x₁₁</td>
<td>No No No No None $ No No Thai 0–10</td>
<td>y₁₁ = No</td>
<td></td>
</tr>
<tr>
<td>x₁₂</td>
<td>Yes Yes Yes Yes Full $ No No Burger 30–60</td>
<td>y₁₂ = Yes</td>
<td></td>
</tr>
</tbody>
</table>

Attributes:

1. Alternate: whether there is a suitable alternative restaurant nearby.
2. Bar: whether the restaurant has a comfortable bar area to wait in.
3. Fri/Sat: true on Fridays and Saturdays.
4. Hungry: whether we are hungry.
5. Patrons: how many people are in the restaurant (values are None, Some, and Full).
6. Price: the restaurant’s price range ($, $$, $$$$).
7. Raining: whether it is raining outside.
8. Reservation: whether we made a reservation.
9. Type: the kind of restaurant (French, Italian, Thai or Burger).
10. WaitEstimate: the wait estimated by the host (0-10 minutes, 10-30, 30-60, >60).
The tree to decide whether to wait (T) or not (F)
- Internal nodes **test attributes**
- Branching is determined by **attribute value**
- Leaf nodes are **outputs** (class assignments)
Decision Tree: Algorithm

- Choose an attribute on which to descend at each level
- Condition on earlier (higher) choices
- Generally, restrict only one dimension at a time
- Declare an output value when you get to the bottom
- In the orange/lemon example, we only split each dimension once, but that is not required
Each path from root to a leaf defines a region $R_m$ of input space

Let $\{(x^{(m_1)}, t^{(m_1)}), \ldots, (x^{(m_k)}, t^{(m_k)})\}$ be the training examples that fall into $R_m$

**Classification tree:**
- discrete output
- leaf value $y^m$ typically set to the most common value in $\{t^{(m_1)}, \ldots, t^{(m_k)}\}$

**Regression tree:**
- continuous output
- leaf value $y^m$ typically set to the mean value in $\{t^{(m_1)}, \ldots, t^{(m_k)}\}$

Note: We will only talk about classification

[Slide credit: S. Russell]
Expressiveness

- **Discrete-input, discrete-output case:**
  - Decision trees can express any function of the input attributes.
  - E.g., for Boolean functions, truth table row $\rightarrow$ path to leaf:

- **Continuous-input, continuous-output case:**
  - Can approximate any function arbitrarily closely.
  - Trivially, there is a consistent decision tree for any training set w/ one path to leaf for each example (unless $f$ nondeterministic in $x$) but it probably won’t generalize to new examples.

Need some kind of regularization to ensure more **compact** decision trees.

[Slide credit: S. Russell]
How do we Learn a Decision Tree?

- How do we construct a useful decision tree?
Learning the simplest (smallest) decision tree is an NP complete problem [if you are interested, check: Hyafil & Rivest’76]

- Resort to a **greedy heuristic:**
  - Start from an empty decision tree
  - Split on next best attribute
  - Recurse

- What is **best** attribute?

- We use **information theory** to guide us

[Slide credit: D. Sontag]
Choosing a Good Attribute

Which attribute is better to split on, $X_1$ or $X_2$?

Idea: Use counts at leaves to define probability distributions, so we can measure uncertainty.
Choosing a Good Attribute

Which attribute is better to split on, $X_1$ or $X_2$?

- Deterministic: good (all are true or false; just one class in the leaf)
- Uniform distribution: bad (all classes in leaf equally probable)
- What about distributions in between?

Note: Let’s take a slight detour and remember concepts from information theory

[Slide credit: D. Sontag]
We Flip Two Different Coins

Sequence 1:
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 ... ?

Sequence 2:
0 1 0 1 0 1 1 1 0 1 0 0 1 1 0 1 0 1 ... ?

16
2
8 10
0 1
t

versus

0 1
Quantifying Uncertainty

**Entropy** $H$:

$$H(X) = - \sum_{x \in X} p(x) \log_2 p(x)$$

- $\frac{8}{9} \log_2 \frac{8}{9} - \frac{1}{9} \log_2 \frac{1}{9} \approx \frac{1}{2}$
- $\frac{4}{9} \log_2 \frac{4}{9} - \frac{5}{9} \log_2 \frac{5}{9} \approx 0.99$

- How surprised are we by a new value in the sequence?
- How much information does it convey?
Quantifying Uncertainty

\[ H(X) = - \sum_{x \in X} p(x) \log_2 p(x) \]
Entrophy

• **“High Entropy”:**
  - Variable has a uniform like distribution
  - Flat histogram
  - Values sampled from it are less predictable

• **“Low Entropy”**
  - Distribution of variable has many peaks and valleys
  - Histogram has many lows and highs
  - Values sampled from it are more predictable

This slide seems wrong

[Slide credit: Vibhav Gogate]
Entropy of a Joint Distribution

Example: \( X = \{ \text{Raining, Not raining} \} \), \( Y = \{ \text{Cloudy, Not cloudy} \} \)

<table>
<thead>
<tr>
<th></th>
<th>Cloudy</th>
<th>Not Cloudy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raining</td>
<td>24/100</td>
<td>1/100</td>
</tr>
<tr>
<td>Not Raining</td>
<td>25/100</td>
<td>50/100</td>
</tr>
</tbody>
</table>

\[
H(X, Y) = - \sum_{x \in X} \sum_{y \in Y} p(x, y) \log_2 p(x, y)
\]

\[
= - \frac{24}{100} \log_2 \frac{24}{100} - \frac{1}{100} \log_2 \frac{1}{100} - \frac{25}{100} \log_2 \frac{25}{100} - \frac{50}{100} \log_2 \frac{50}{100}
\]

\[
\approx 1.56 \text{bits}
\]
Specific Conditional Entropy

- Example: \( X = \{\text{Raining, Not raining}\}, \ Y = \{\text{Cloudy, Not cloudy}\} \)

<table>
<thead>
<tr>
<th></th>
<th>Cloudy</th>
<th>Not Cloudy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raining</td>
<td>24/100</td>
<td>1/100</td>
</tr>
<tr>
<td>Not Raining</td>
<td>25/100</td>
<td>50/100</td>
</tr>
</tbody>
</table>

- What is the entropy of cloudiness \( Y \), **given that it is raining**?

\[
H(Y|X = x) = -\sum_{y \in Y} p(y|x) \log_2 p(y|x)
\]

\[
= -\frac{24}{25} \log_2 \frac{24}{25} - \frac{1}{25} \log_2 \frac{1}{25}
\]

\[
\approx 0.24 \text{bits}
\]

- We used: \( p(y|x) = \frac{p(x,y)}{p(x)} \), and \( p(x) = \sum_y p(x,y) \) (sum in a row)
### Conditional Entropy

<table>
<thead>
<tr>
<th></th>
<th>Cloudy</th>
<th>Not Cloudy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raining</td>
<td>24/100</td>
<td>1/100</td>
</tr>
<tr>
<td>Not Raining</td>
<td>25/100</td>
<td>50/100</td>
</tr>
</tbody>
</table>

The expected conditional entropy:

\[
H(Y|X) = \sum_{x \in X} p(x) H(Y|X = x)
\]

\[
= -\sum_{x \in X} \sum_{y \in Y} p(x, y) \log_2 p(y|x)
\]
Example: $X = \{\text{Raining, Not raining}\}$, $Y = \{\text{Cloudy, Not cloudy}\}$

<table>
<thead>
<tr>
<th></th>
<th>Cloudy</th>
<th>Not Cloudy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raining</td>
<td>24/100</td>
<td>1/100</td>
</tr>
<tr>
<td>Not Raining</td>
<td>25/100</td>
<td>50/100</td>
</tr>
</tbody>
</table>

What is the entropy of cloudiness, given the knowledge of whether or not it is raining?

$$H(Y|X) = \sum_{x \in X} p(x)H(Y|X = x)$$

$$= \frac{1}{4} H(\text{cloudy|is raining}) + \frac{3}{4} H(\text{cloudy|not raining})$$

$$\approx 0.75 \text{ bits}$$
Conditional Entropy

Some useful properties:

- $H$ is always non-negative
- Chain rule: $H(X, Y) = H(X|Y) + H(Y) = H(Y|X) + H(X)$
- If $X$ and $Y$ independent, then $X$ doesn’t tell us anything about $Y$: $H(Y|X) = H(Y)$
- But $Y$ tells us everything about $Y$: $H(Y|Y) = 0$
- By knowing $X$, we can only decrease uncertainty about $Y$: $H(Y|X) \leq H(Y)$
Information Gain

<table>
<thead>
<tr>
<th></th>
<th>Cloudy</th>
<th>Not Cloudy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raining</td>
<td>24/100</td>
<td>1/100</td>
</tr>
<tr>
<td>Not Raining</td>
<td>25/100</td>
<td>50/100</td>
</tr>
</tbody>
</table>

How much information about cloudiness do we get by discovering whether it is raining?

\[
IG(Y|X) = H(Y) - H(Y|X) \\
\approx \ 0.25 \text{ bits}
\]

Also called information gain in \( Y \) due to \( X \)

- If \( X \) is completely uninformative about \( Y \): \( IG(Y|X) = 0 \)
- If \( X \) is completely informative about \( Y \): \( IG(Y|X) = H(Y) \)

How can we use this to construct our decision tree?
I made the fruit data partitioning just by eyeballing it.

We can use the information gain to automate the process.

At each level, one must choose:

1. Which variable to split.
2. Possibly where to split it.

Choose them based on how much information we would gain from the decision! (choose attribute that gives the highest gain)
Decision Tree Construction Algorithm

- Simple, greedy, recursive approach, builds up tree node-by-node

1. pick an attribute to split at a non-terminal node
2. split examples into groups based on attribute value
3. for each group:
   ▶ if no examples – return majority from parent
   ▶ else if all examples in same class – return class
   ▶ else loop to step 1
### Back to Our Example

<table>
<thead>
<tr>
<th>Example</th>
<th>Input Attributes</th>
<th>Goal</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>Yes  No  No  Yes</td>
<td>Some</td>
<td>$$$$  No Yes French</td>
</tr>
<tr>
<td>$x_2$</td>
<td>Yes  No  No  Yes</td>
<td>Full</td>
<td>$       No No Thai</td>
</tr>
<tr>
<td>$x_3$</td>
<td>No   Yes No  No</td>
<td>Some</td>
<td>$       No No Burger</td>
</tr>
<tr>
<td>$x_4$</td>
<td>Yes  No  Yes Yes</td>
<td>Full</td>
<td>$       Yes No Thai</td>
</tr>
<tr>
<td>$x_5$</td>
<td>Yes  No  Yes No</td>
<td>Full</td>
<td>$$$$  No Yes French</td>
</tr>
<tr>
<td>$x_6$</td>
<td>No   Yes No  Yes</td>
<td>Some</td>
<td>$$      Yes Yes Italian</td>
</tr>
<tr>
<td>$x_7$</td>
<td>No   Yes No  No</td>
<td>None</td>
<td>$       Yes No Burger</td>
</tr>
<tr>
<td>$x_8$</td>
<td>No   No No  Yes</td>
<td>Some</td>
<td>$$      Yes Yes Thai</td>
</tr>
<tr>
<td>$x_9$</td>
<td>No   Yes Yes No</td>
<td>Full</td>
<td>$$      Yes No Burger</td>
</tr>
<tr>
<td>$x_{10}$</td>
<td>Yes  Yes Yes Yes</td>
<td>Full</td>
<td>$$$$  No Yes Italian</td>
</tr>
<tr>
<td>$x_{11}$</td>
<td>No   No No  No</td>
<td>None</td>
<td>$       No No Thai</td>
</tr>
<tr>
<td>$x_{12}$</td>
<td>Yes  Yes Yes Yes</td>
<td>Full</td>
<td>$       No No Burger</td>
</tr>
</tbody>
</table>

1. Alternate: whether there is a suitable alternative restaurant nearby.
2. Bar: whether the restaurant has a comfortable bar area to wait in.
3. Fri/Sat: true on Fridays and Saturdays.
4. Hungry: whether we are hungry.
5. Patrons: how many people are in the restaurant (values are None, Some, and Full).
6. Price: the restaurant's price range ($, $$, $$$).
7. Raining: whether it is raining outside.
8. Reservation: whether we made a reservation.
9. Type: the kind of restaurant (French, Italian, Thai or Burger).
10. WaitEstimate: the wait estimated by the host (0-10 minutes, 10-30, 30-60, >60).

([from: Russell & Norvig])
Attribute Selection

\[
IG(Y) = H(Y) - H(Y|X)
\]

\[
IG(type) = 1 - \left[ \frac{2}{12} H(Y|Fr.) + \frac{2}{12} H(Y|It.) + \frac{4}{12} H(Y|Thai) + \frac{4}{12} H(Y|Bur.) \right] = 0
\]

\[
IG(Patrons) = 1 - \left[ \frac{2}{12} H(0, 1) + \frac{4}{12} H(1, 0) + \frac{6}{12} H(\frac{2}{6}, \frac{4}{6}) \right] \approx 0.541
\]
Not too small: need to handle important but possibly subtle distinctions in data

Not too big:
  - Computational efficiency (avoid redundant, spurious attributes)
  - Avoid over-fitting training examples

Occam’s Razor: find the simplest hypothesis (smallest tree) that fits the observations

Inductive bias: small trees with informative nodes near the root
Problems:

- You have exponentially less data at lower levels
- Too big of a tree can overfit the data
- Greedy algorithms don’t necessarily yield the global optimum

In practice, one often regularizes the construction process to try to get small but highly-informative trees

Decision trees can also be used for regression on real-valued outputs, but it requires a different formalism
Comparison to k-NN

K-Nearest Neighbors

- Decision boundaries: piece-wise linear
- Test complexity: non-parametric, few parameters besides (all?) training examples

Decision Trees

- Decision boundaries: axis-aligned, tree structured
- Test complexity: attributes and splits
Applications of Decision Trees: XBox!

- Decision trees are in XBox

Applications of Decision Trees: XBox!

- Decision trees are in XBox: Classifying body parts
Applications of Decision Trees: XBox!

- Trained on million(s) of examples
Applications of Decision Trees: XBox!

- Trained on million(s) of examples

Results:
Applications of Decision Trees

- Can express any Boolean function, but most useful when function depends critically on few attributes
- Bad on: parity, majority functions; also not well-suited to continuous attributes
- Practical Applications:
  - Flight simulator: 20 state variables; 90K examples based on expert pilot’s actions; auto-pilot tree
  - Yahoo Ranking Challenge
  - Random Forests: Microsoft Kinect Pose Estimation