Supervised Learning (contd)
Decision Trees

Mausam
(based on slides by UW-AI faculty)
Decision Trees

To play or not to play?
Example data for learning the concept “Good day for tennis”

<table>
<thead>
<tr>
<th>Day</th>
<th>Outlook</th>
<th>Humid</th>
<th>Wind</th>
<th>PlayTennis?</th>
</tr>
</thead>
<tbody>
<tr>
<td>d1</td>
<td>s</td>
<td>h</td>
<td>w</td>
<td>n</td>
</tr>
<tr>
<td>d2</td>
<td>s</td>
<td>h</td>
<td>s</td>
<td>n</td>
</tr>
<tr>
<td>d3</td>
<td>o</td>
<td>h</td>
<td>w</td>
<td>y</td>
</tr>
<tr>
<td>d4</td>
<td>r</td>
<td>h</td>
<td>w</td>
<td>y</td>
</tr>
<tr>
<td>d5</td>
<td>r</td>
<td>n</td>
<td>w</td>
<td>y</td>
</tr>
<tr>
<td>d6</td>
<td>r</td>
<td>n</td>
<td>s</td>
<td>y</td>
</tr>
<tr>
<td>d7</td>
<td>o</td>
<td>n</td>
<td>s</td>
<td>y</td>
</tr>
<tr>
<td>d8</td>
<td>s</td>
<td>h</td>
<td>w</td>
<td>n</td>
</tr>
<tr>
<td>d9</td>
<td>s</td>
<td>n</td>
<td>w</td>
<td>y</td>
</tr>
<tr>
<td>d10</td>
<td>r</td>
<td>n</td>
<td>w</td>
<td>y</td>
</tr>
<tr>
<td>d11</td>
<td>s</td>
<td>n</td>
<td>s</td>
<td>y</td>
</tr>
<tr>
<td>d12</td>
<td>o</td>
<td>h</td>
<td>s</td>
<td>y</td>
</tr>
<tr>
<td>d13</td>
<td>o</td>
<td>n</td>
<td>w</td>
<td>y</td>
</tr>
<tr>
<td>d14</td>
<td>r</td>
<td>h</td>
<td>s</td>
<td>n</td>
</tr>
</tbody>
</table>

- **Outlook** = sunny, overcast, rain
- **Humidity** = high, normal
- **Wind** = weak, strong
A Decision Tree for the Same Data

Decision Tree for “PlayTennis?”

Leaves = classification output
Arcs = choice of value
for parent attribute

Decision tree is equivalent to logic in disjunctive normal form
PlayTennis ⇔ (Sunny ∧ Normal) ∨ Overcast ∨ (Rain ∧ Weak)
**Decision Trees**

**Input:** Description of an object or a situation through a set of attributes

**Output:** a decision that is the predicted output value for the input

Both input and output can be discrete or continuous

Discrete-valued functions lead to classification problems
Example: Decision Tree for Continuous Valued Features and Discrete Output

Input real number attributes \((x_1, x_2)\), Classification output: 0 or 1

How do we branch using attribute values \(x_1\) and \(x_2\) to partition the space correctly?
Example: Classification of Continuous Valued Inputs

Decision trees divide the feature space into axis-parallel rectangles, and label each rectangle with one of the $K$ classes.
Decision trees can express any function of the input attributes.

E.g., for Boolean functions, truth table row = path to leaf:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A xor B</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
</tbody>
</table>

Trivially, there is a consistent decision tree for any training set with one path to leaf for each example.

- But most likely won't generalize to new examples

Prefer to find more compact decision trees
Learning Decision Trees

Example: When should I wait for a table at a restaurant?

Attributes (features) relevant to Wait? decision:

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)
Example Decision tree

A decision tree for *Wait?* based on personal “rules of thumb”:

```
Patrons?
  None  Some  Full
    F     T

WaitEstimate?
  >60  30–60  10–30
    F     T

Alternate?
  No  Yes

Hungry?
  No  Yes

Reservation?
  No  Yes  No  Yes

Fri/Sat?
  T
  No  T

Bar?
  No  Yes

Alternate?
  No  Yes

Raining?
  F  T
  T
```
# Input Data for Learning

Past examples when I did/did not 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>Target</th>
<th>$Wait$</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>$$$</td>
<td>F</td>
<td>T</td>
<td>French</td>
<td>0–10</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>$X_2$</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>Full</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Thai</td>
<td>30–60</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>$X_3$</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>Some</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Burger</td>
<td>0–10</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>$X_4$</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>Full</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Thai</td>
<td>10–30</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>$X_5$</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>Full</td>
<td>$$$</td>
<td>F</td>
<td>T</td>
<td>French</td>
<td>&gt;60</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>$X_6$</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>Some</td>
<td>$</td>
<td>T</td>
<td>T</td>
<td>Italian</td>
<td>0–10</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>$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>
<td></td>
</tr>
<tr>
<td>$X_8$</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>Some</td>
<td>$</td>
<td>T</td>
<td>T</td>
<td>Thai</td>
<td>0–10</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>$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>
<td></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>
<td></td>
</tr>
<tr>
<td>$X_{11}$</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>None</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Thai</td>
<td>0–10</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>$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>
<td></td>
</tr>
</tbody>
</table>

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

Aim: find a small tree consistent with training examples
Idea: (recursively) choose "most significant" attribute as root of (sub)tree

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same classification then return the classification
    else if attributes is empty then return MODE(examples)
    else
        best ← CHOOSE-ATTRIBUTE(attributes, examples)
        tree ← a new decision tree with root test best
        for each value $v_i$ of best do
            $e_i \leftarrow \{\text{elements of examples with } best = v_i\}$
            $s \leftarrow \text{DTL}(e_i, attributes - best, \text{MODE}(examples))$
            add a branch to tree with label $v_i$ and subtree $s$
    return tree
```
Choosing an attribute to split on

Idea: a good attribute should reduce uncertainty
  • E.g., splits the examples into subsets that are (ideally) "all positive" or "all negative"

*Patrons*? is a better choice

For *Type*?, to wait or not to wait is still at 50%
How do we quantify uncertainty?
Using information theory to quantify uncertainty

Entropy measures the amount of uncertainty in a probability distribution.

Entropy (or Information Content) of an answer to a question with possible answers $v_1, \ldots, v_n$:

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

Imagine we have \( p \) examples with \( \text{Wait} = \text{True} \) (positive) and \( n \) examples with \( \text{Wait} = \text{false} \) (negative).

Our best estimate of the probabilities of \( \text{Wait} = \text{true} \) or \( \text{false} \) is given by:

\[
P(\text{true}) \approx \frac{p}{p + n} \\
P(\text{false}) \approx \frac{n}{p + n}
\]

Hence the entropy of \( \text{Wait} \) is given by:

\[
I\left(\frac{p}{p + n}, \frac{n}{p + n}\right) = -\frac{p}{p + n} \log_2 \left(\frac{p}{p + n}\right) - \frac{n}{p + n} \log_2 \left(\frac{n}{p + n}\right)
\]
Entropy is highest when uncertainty is greatest.

$P(Wait = T)$
Choosing an attribute to split on

Idea: a good attribute should reduce uncertainty and result in “gain in information”

How much information do we gain if we disclose the value of some attribute?

Answer:

uncertainty before – uncertainty after
Before choosing an attribute:

Entrophy = $- \frac{6}{12} \log(\frac{6}{12}) - \frac{6}{12} \log(\frac{6}{12})$

$= - \log(1/2) = \log(2) = 1$ bit

There is “1 bit of information to be discovered”
If we choose Type: Go along branch “French”: we have entropy = 1 bit; similarly for the others.
  Information gain = 1-1 = 0 along any branch

If we choose Patrons:
In branch “None” and “Some”, entropy = 0
For “Full”, entropy = $-2/6 \log(2/6)-4/6 \log(4/6) = 0.92$
  Info gain = (1-0) or (1-0.92) bits > 0 in both cases
So choosing Patrons gains more information!
Entropy across branches

- How do we combine entropy of different branches?
- Answer: Compute average entropy
- Weight entropies according to probabilities of branches
  2/12 times we enter “None”, so weight for “None” = 1/6
  “Some” has weight: 4/12 = 1/3
  “Full” has weight 6/12 = 1/2

\[
\text{AvgEntropy} = \sum_{i=1}^{n} \frac{p_i + n_i}{p + n} \text{Entropy}(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i})
\]

weight for each branch

entropy for each branch
Information Gain (IG) or reduction in entropy from using attribute A:

\[ IG(A) = \text{Entropy before} - \text{AvgEntropy after choosing A} \]

Choose the attribute with the largest IG
Information gain in our example

\[ IG(\text{Patrons}) = 1 - \left[ \frac{2}{12} I(0,1) + \frac{4}{12} I(1,0) + \frac{6}{12} I\left(\frac{2}{6}, \frac{4}{6}\right) \right] = .541 \text{ bits} \]

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

*Patrons* has the highest IG of all attributes

⇒ DTL algorithm chooses *Patrons* as the root
Should I stay or should I go?

Learned Decision Tree

Decision tree learned from the 12 examples:

Substantially simpler than “rules-of-thumb” tree
  • more complex hypothesis not justified by small amount of data
Performance Evaluation

How do we know that the learned tree $h \approx f$?
Answer: Try $h$ on a new test set of examples.

Learning curve = % correct on test set as a function of training set size.

![Learning curve graph](image)
Overfitting

Accuracy

0.9

On training data

0.8

On test data

0.7

0.6

Number of Nodes in Decision tree

26
Overfitting

Consider error of hypothesis $h$ over

- training data: $error_{train}(h)$
- entire distribution $\mathcal{D}$ of data: $error_{\mathcal{D}}(h)$

Hypothesis $h \in H$ overfits training data if there is an alternative hypothesis $h' \in H$ such that

$$error_{train}(h) < error_{train}(h')$$

and

$$error_{\mathcal{D}}(h) > error_{\mathcal{D}}(h')$$
Rule #2 of Machine Learning

The best hypothesis almost never achieves 100% accuracy on the training data.

(Rule #1 was: you can’t learn anything without inductive bias)
Avoiding Overfitting

How can we avoid overfitting?

- Stop growing when data split not statistically significant
- Grow full tree, then post-prune

How to select “best” tree:

- Measure performance over training data
- Measure performance over separate validation data set
- Add complexity penalty to performance measure
### Reduced Error Pruning

1. **Split data into train and validation set**

2. **Repeat until pruning is harmful**
   - Remove each subtree and replace it with majority class and evaluate on validation set.
   - Remove subtree that leads to largest gain in accuracy.

---

<table>
<thead>
<tr>
<th>Split data into train and validation set</th>
<th>Tune</th>
<th>Tune</th>
<th>Tune</th>
</tr>
</thead>
<tbody>
<tr>
<td>Repeat until pruning is harmful</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Remove each subtree and replace it with majority class and evaluate on validation set.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Remove subtree that leads to largest gain in accuracy</td>
<td></td>
<td></td>
<td>Test</td>
</tr>
</tbody>
</table>
Reduced Error Pruning Example

Validation set accuracy = 0.75
Reduced Error Pruning Example

Validation set accuracy = 0.80
Reduced Error Pruning Example

Validation set accuracy = 0.70
Reduced Error Pruning Example

Use this as final tree
Early Stopping

Number of Nodes in Decision tree

Accuracy

On training data
On test data
On validation data

Remember this tree and use it as the final classifier
Post Rule Pruning

• Split data into train and validation set

• Prune each rule independently
  – Remove each pre-condition and evaluate accuracy
  – Pick pre-condition that leads to largest improvement in accuracy

• Note: ways to do this using training data and statistical tests
Conversion to Rule

- **Outlook = Sunny ∧ Humidity = High ⇒ Don’t play**
- **Outlook = Sunny ∧ Humidity = Low ⇒ Play**
- **Outlook = Overcast ⇒ Play**
Scaling Up

- ID3, C4.5, etc. assume data fits in main memory (OK for up to hundreds of thousands of examples)

- SPRINT, SLIQ: multiple sequential scans of data (OK for up to millions of examples)

- VFDT: at most one sequential scan (OK for up to billions of examples)
Decision Trees - Strengths

Very Popular Technique
Fast
Useful when

* Instances are attribute-value pairs
* Target Function is discrete
* Concepts are likely to be disjunctions
* Attributes may be noisy
Decision Trees - Weaknesses

Less useful for continuous outputs

Can have difficulty with continuous input features as well...

• E.g., what if your target concept is a circle in the $x_1, x_2$ plane?
  - Hard to represent with decision trees...
  - Very simple with instance-based methods we’ll discuss later...