Inductive Learning and Decision Trees

Doug Downey
EECS 349 Fall 2010

with slides from Pedro Domingos, Bryan Pardo
Outline

• Announcements
  – Homework #1 will be assigned shortly
• Inductive learning
• Decision Trees
Outline

• Announcements
  – Homework #1 will be assigned shortly

• Inductive learning

• Decision Trees
**Instances**

- E.g. Days, in terms of weather:

<table>
<thead>
<tr>
<th>Sky</th>
<th>Temp</th>
<th>Humid</th>
<th>Wind</th>
<th>Water</th>
<th>Forecast</th>
</tr>
</thead>
<tbody>
<tr>
<td>sunny</td>
<td>warm</td>
<td>normal</td>
<td>strong</td>
<td>warm</td>
<td>same</td>
</tr>
<tr>
<td>sunny</td>
<td>warm</td>
<td>high</td>
<td>strong</td>
<td>warm</td>
<td>same</td>
</tr>
<tr>
<td>rainy</td>
<td>cold</td>
<td>high</td>
<td>strong</td>
<td>warm</td>
<td>change</td>
</tr>
<tr>
<td>sunny</td>
<td>warm</td>
<td>high</td>
<td>strong</td>
<td>cool</td>
<td>change</td>
</tr>
</tbody>
</table>
Functions

- “Days on which my friend Aldo enjoys his favorite water sport”

<table>
<thead>
<tr>
<th>Sky</th>
<th>Temp</th>
<th>Humid</th>
<th>Wind</th>
<th>Water</th>
<th>Forecast</th>
<th>C(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sunny</td>
<td>warm</td>
<td>normal</td>
<td>strong</td>
<td>warm</td>
<td>same</td>
<td>1</td>
</tr>
<tr>
<td>sunny</td>
<td>warm</td>
<td>high</td>
<td>strong</td>
<td>warm</td>
<td>same</td>
<td>1</td>
</tr>
<tr>
<td>rainy</td>
<td>cold</td>
<td>high</td>
<td>strong</td>
<td>warm</td>
<td>change</td>
<td>0</td>
</tr>
<tr>
<td>sunny</td>
<td>warm</td>
<td>high</td>
<td>strong</td>
<td>cool</td>
<td>change</td>
<td>1</td>
</tr>
</tbody>
</table>
### Inductive Learning!

- **Predict** the output for a new instance

<table>
<thead>
<tr>
<th>Sky</th>
<th>Temp</th>
<th>Humid</th>
<th>Wind</th>
<th>Water</th>
<th>Forecast</th>
<th>C(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sunny</td>
<td>warm</td>
<td>normal</td>
<td>strong</td>
<td>warm</td>
<td>same</td>
<td>1</td>
</tr>
<tr>
<td>sunny</td>
<td>warm</td>
<td>high</td>
<td>strong</td>
<td>warm</td>
<td>same</td>
<td>1</td>
</tr>
<tr>
<td>rainy</td>
<td>cold</td>
<td>high</td>
<td>strong</td>
<td>warm</td>
<td>change</td>
<td>0</td>
</tr>
<tr>
<td>sunny</td>
<td>warm</td>
<td>high</td>
<td>strong</td>
<td>cool</td>
<td>change</td>
<td>1</td>
</tr>
<tr>
<td><strong>rainy</strong></td>
<td><strong>warm</strong></td>
<td><strong>high</strong></td>
<td><strong>strong</strong></td>
<td><strong>cool</strong></td>
<td><strong>change</strong></td>
<td>?</td>
</tr>
</tbody>
</table>
General Inductive Learning Task

DEFINE:

- Set $X$ of Instances (of $n$-tuples $x = <x_1, ..., x_n>\) )
  - E.g., days described by attributes (or features):
    Sky, Temp, Humidity, Wind, Water, Forecast

- Target function $y$, e.g.:
  - EnjoySport $X \rightarrow Y = \{0,1\}$
  - HoursOfSport $X \rightarrow Y = \{0, 1, 2, 3, 4\}$
  - InchesOfRain $X \rightarrow Y = [0, 10]\)

GIVEN:

- Training examples $D$
  - examples of the target function: $<x, y(x)>$

FIND:

- A hypothesis $h$ such that $h(x)$ approximates $y(x)$. 
Another example: continuous attributes

Learn function from \( \mathbf{x} = (x_1, \ldots, x_d) \) to \( f(\mathbf{x}) \in \{0, 1\} \) given labeled examples \((\mathbf{x}, f(\mathbf{x}))\)
Hypothesis Spaces

- **Hypothesis space** $H$ is a **subset** of all $y: X \rightarrow Y$ e.g.:
  - Linear separators
  - Conjunctions of constraints on attributes (humidity must be low, and outlook != rain)
  - Etc.

- In machine learning, we restrict ourselves to $H$
  - The *subset* thing turns out to be important
Examples

• Credit Risk Analysis
  – \( X \): Properties of customer and proposed purchase
  – \( f(x) \): Approve (1) or Disapprove (0)

• Disease Diagnosis
  – \( X \): Properties of patient (symptoms, lab tests)
  – \( f(x) \): Disease (if any)

• Face Recognition
  – \( X \): Bitmap image
  – \( f(x) \): Name of person

• Automatic Steering
  – \( X \): Bitmap picture of road surface in front of car
  – \( f(x) \): Degrees to turn the steering wheel
Appropriate applications

• Situations in which:
  – there is no human expert
  – Humans can perform the task but can’t describe how
  – The desired function changes frequently
  – Each user needs a customized $f$
Outline

• Announcements
  – Homework #1 will be assigned shortly

• Inductive learning

• Decision Trees
## Task: Will I wait for a table?

<table>
<thead>
<tr>
<th>Example</th>
<th>Attributes</th>
<th>Target</th>
<th>WillWait</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Alt Bar Fri Hun Pat Price Rain Res Type Est</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X₁</td>
<td>T F F T</td>
<td>Some</td>
<td>$$$</td>
</tr>
<tr>
<td>X₂</td>
<td>T F F T</td>
<td>Full</td>
<td>$</td>
</tr>
<tr>
<td>X₃</td>
<td>F T F F</td>
<td>Some</td>
<td>$</td>
</tr>
<tr>
<td>X₄</td>
<td>T F T T</td>
<td>Full</td>
<td>$</td>
</tr>
<tr>
<td>X₅</td>
<td>T F T F</td>
<td>Full</td>
<td>$$$</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)
Decision Trees!

One possible representation for hypotheses
E.g., here is the “true” tree for deciding whether to wait:

```
Patrons?
  None  Full
     F    T
    /\    /
  Some <60 30-60
   T   F

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

Alternate?
  No  Yes
   T   F

Hungry?
  No  Yes
   T   F

Reservation?
  No  Yes  No  Yes
   T   F   T

Fri/Sat?
  No  Yes  No  Yes
   T   F   T

Bar?
  No  Yes
   T   F

Alternate?
  No  Yes
   T   F

Raining?
  No  Yes
   F   T
```
Expressiveness of D-Trees

Decision trees can express any function of the input attributes. E.g., for Boolean functions, truth table row $\rightarrow$ 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 w/ one path to leaf for each example (unless $f$ nondeterministic in $x$) but it probably won’t generalize to new examples

Prefer to find more compact decision trees
A learned decision tree

Decision tree learned from the 12 examples:

Substantially simpler than “true” tree—a more complex hypothesis isn’t justified by small amount of data
Inductive Bias

• To learn, we must prefer some functions to others

  – **Selection bias**
    • use a **restricted** hypothesis space, e.g.:
      – linear separators
      – 2-level decision trees

  – **Preference bias**
    • use the whole function space, but state a **preference** over concepts, e.g.:
      – *Lowest-degree* polynomial that separates the data
      – *shortest* decision tree that fits the data
Decision Tree Learning (ID3)

Aim: find a small tree consistent with the 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
        \( \text{examples}_i \) ← \{elements of examples with best = \( v_i \}\}
        subtree ← DTL(examples_i, attributes – best, MODE(examples))
        add a branch to tree with label \( v_i \) and subtree subtree
    return tree
```
Recap

• Inductive learning
  – Goal: generate a hypothesis – a function from instances described by attributes to an output – using training examples.
  – Requires inductive bias
    • a restricted hypothesis space, or preferences over hypotheses.

• Decision Trees
  – Simple representation of hypotheses, recursive learning algorithm
  – Prefer smaller trees!
Outline

• Homework #1 assigned, due next Tuesday
• Decision Trees (cont.)
  – Choosing attributes for splitting
  – Measuring performance
  – Overfitting
  – Odds & Ends
Decision Trees!

One possible representation for hypotheses
E.g., here is the “true” tree for deciding whether to wait:

```
Patrons?
  None  F
  Some T
  Full

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

Alternate?
  No F
  Yes T

Hungry?
  No F
  Yes T

Reservation?
  No F
  Yes T

Fri/Sat?
  No F
  Yes T

Alternate?
  No F
  Yes T

Bar?
  No F
  Yes T

Raining?
  No F
  Yes T
```
Choosing an attribute

Idea: a good attribute splits the examples into subsets that are (ideally) “all positive” or “all negative”

Patrons? is a better choice—gives information about the classification
Information

Information answers questions

The more clueless I am about the answer initially, the more information is contained in the answer.

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

Information in an answer when prior is \( \langle P_1, \ldots, P_n \rangle \) is

\[
H(\langle P_1, \ldots, P_n \rangle) = \sum_{i=1}^{n} - P_i \log_2 P_i
\]

(also called entropy of the prior)
Entropy

The entropy $H(V)$ of a Boolean random variable $V$ as the probability of $V = 0$ varies from 0 to 1.
Using Information

Suppose we have \( p \) positive and \( n \) negative examples at the root

\[
\Rightarrow \quad H\left(\langle p/(p+n), n/(p+n)\rangle\right) \text{ bits needed to classify a new example}
\]

E.g., for 12 restaurant examples, \( p = n = 6 \) so we need 1 bit

An attribute splits the examples \( E \) into subsets \( E_i \), each of which (we hope) needs less information to complete the classification

Let \( E_i \) have \( p_i \) positive and \( n_i \) negative examples

\[
\Rightarrow \quad H\left(\langle p_i/(p_i+n_i), n_i/(p_i+n_i)\rangle\right) \text{ bits needed to classify a new example}
\]

\[
\Rightarrow \quad \text{expected} \quad \text{number of bits per example over all branches is}
\]

\[
\sum_i \frac{p_i + n_i}{p + n} \cdot H\left(\langle p_i/(p_i+n_i), n_i/(p_i+n_i)\rangle\right)
\]

For \( Patrons? \), this is 0.459 bits, for \( Type \) this is (still) 1 bit

\[
\Rightarrow \quad \text{choose the attribute that minimizes the remaining information needed}
\]
Outline

• Homework #1 assigned, due next Tuesday
• Decision Trees (cont.)
  – Choosing attributes for splitting
  – Measuring performance
  – Overfitting
  – Odds & Ends
Measuring Performance

How do we know that $h \approx f$? (Hume’s Problem of Induction)

1) Use theorems of computational/statistical learning theory

2) Try $h$ on a new test set of examples
   (use same distribution over example space as training set)

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

   ![Learning curve graph](image-url)
What the learning curve tells us

Learning curve depends on
- **realizable** (can express target function) vs. **non-realizable**
  - non-realizability can be due to missing attributes
  - or restricted hypothesis class (e.g., thresholded linear function)
- redundant expressiveness (e.g., loads of irrelevant attributes)
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)
What’s “good” performance?

- **Accuracy** = fraction of output classifications that are correct on the test data
- But what about these tasks:
  - Is a randomly drawn Web doc about baseball?
  - Is record M in DB1 the same as M’ in DB2?

- Answering “no” all the time ensures >99% accuracy!
  - Useful to compare with baseline (e.g. “ZeroR”), or use metrics that fit the application (e.g. precision, recall)
Outline

• Homework #1 assigned, due next Tuesday
• Decision Trees (cont.)
  – Choosing attributes for splitting
  – Measuring performance
  – Overfitting
  – Odds & Ends
Overfitting
Example of Overfitting

Learn function from $\mathbf{x} = (x_1, \ldots, x_d)$ to $y \in \{0, 1\}$ given labeled examples $(\mathbf{x}, y)$.
Overfitting is due to “noise”

• Sources of noise:
  – Erroneous training data
    • concept variable incorrect (annotator error)
    • Attributes mis-measured
  – Much more significant:
    • Irrelevant attributes
    • Target function not deterministic in attributes
Irrelevant attributes

• If many attributes are irrelevant, information gains can be spurious, e.g.:
  • 20 noisy attributes
  • 10 training examples
  • Expected # of different depth-3 trees that split the training data perfectly using *only* noisy attributes: 13.4
Non-determinism

• In general:
  – We can’t measure all the variables we need to do perfect prediction.
  – => Target function is not uniquely determined by attribute values
Non-determinism: Example

**Decent hypothesis:**
Humidity > 0.70 → No
Otherwise → Yes

**Overfit hypothesis:**
Humidity > 0.89 → No
Humidity > 0.80
^ Humidity <= 0.89 → Yes
Humidity > 0.70
^ Humidity <= 0.80 → No
Humidity <= 0.70 → Yes
Avoiding Overfitting

• Approaches
  – Stop splitting when information gain is low or when split is not statistically significant.
  – Grow full tree and then prune it when done

• How to pick the “best” tree?
  – Performance on training data?
  – Performance on validation data?
  – Complexity penalty?
Reduced-Error Pruning

Split data into training and validation set

Do until further pruning is harmful:

1. Evaluate impact on validation set of pruning each possible node (plus those below it)

2. Greedily remove the one that most improves validation set accuracy
Effect of Reduced Error Pruning

![Graph showing the effect of reduced error pruning on accuracy](image)

- On training data
- On test data
- On test data (during pruning)

Accuracy

Size of tree (number of nodes)
C4.5 Algorithm

• Builds a decision tree from labeled training data
• Also by Ross Quinlan
• Generalizes ID3 by
  – Allowing continuous value attributes
  – Allows missing attributes in examples
  – Prunes tree after building to improve generality
Rule post pruning

• Used in C4.5

• Steps
  1. Build the decision tree
  2. Convert it to a set of logical rules
  3. Prune each rule independently
  4. Sort rules into desired sequence for use
Converting A Tree to Rules

```
Outlook
  Sunny
  Humidity
    High
    No
    Normal
    Yes
  Overcast
  Rain
    Yes
    Wind
      Strong
      No
      Weak
      Yes
```
IF \((Outlook = Sunny) \ AND \ (Humidity = High)\)  
THEN \(PlayTennis = No\)

IF \((Outlook = Sunny) \ AND \ (Humidity = Normal)\)  
THEN \(PlayTennis = Yes\)

...
Cross-Validation

• More complete version of validation set

• Partition data in \( k \) pieces (= “folds”)

• For each piece \( p \)
  – **Train** on all pieces but \( p \), **test** on \( p \)

• Cross-validation perf. = average perf. over all folds
Outline

• Homework #1 assigned, due next Tuesday
• Decision Trees (cont.)
  – Choosing attributes for splitting
  – Measuring performance
  – Overfitting
  – Odds & Ends
Scaling up

• ID3, C4.5: Assume data fits in memory
• SPRINT, SLIQ: Multiple *sequential* scans of data
• VFDT: At most one sequential scan
Unknown Attribute Values

What if some examples are missing values of $A$?
Use training example anyway, sort through tree

- If node $n$ tests $A$, assign most common value of $A$
  among other examples sorted to node $n$
- Assign most common value of $A$ among other examples
  with same target value
- Assign probability $p_i$ to each possible value $v_i$ of $A$
  Assign fraction $p_i$ of example to each descendant in tree

Classify new examples in same fashion
Continuous Attributes

Decision trees divide the feature space into axis-parallel rectangles, and label each rectangle with one of the $K$ classes.
Learning Parity with Noise

When learning exclusive-or (2-bit parity), all splits look equally good. If extra random boolean features are included, they also look equally good. Hence, decision tree algorithms cannot distinguish random noisy features from parity features.

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

$J=4$
Decision Trees Inductive Bias

- How to solve 2-bit parity:
  - Two step look-ahead, or
  - Split on pairs of attributes at once

- For $k$-bit parity, why not just do $k$-step look ahead? Or split on $k$ attribute values?

=> Parity functions are the “victims” of the decision tree’s inductive bias.
Take aways for decision trees

• Used as classifiers
• Supervised learning algorithms (ID3, C4.5)
• (mostly) Batch processing
• Good for situations where
  – The classification categories are finite
  – The data can be represented as vectors of attributes
  – *Understandability* of learned function is needed