Machine Learning: Decision Trees

Chapter 19.3

Some material adopted from notes by Chuck Dyer
Decision Trees (DTs)

• A *supervised* learning method used for *classification* and *regression*

• Given a set of training tuples, learn model to predict one value from the others
  – Learned value typically a class (e.g., goodRisk)

• Resulting model is simple to understand, interpret, visualize, and apply
Learning a Concept

The red groups are **negative** examples, blue **positive**

Attributes

- **Size**: large, small
- **Color**: red, green, blue
- **Shape**: square, circle

Task

Classify new object with a size, color & shape as positive or negative
## Training data

<table>
<thead>
<tr>
<th>Size</th>
<th>Color</th>
<th>Shape</th>
<th>class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large</td>
<td>Green</td>
<td>Square</td>
<td>Negative</td>
</tr>
<tr>
<td>Large</td>
<td>Green</td>
<td>Circle</td>
<td>Negative</td>
</tr>
<tr>
<td>Small</td>
<td>Green</td>
<td>Square</td>
<td>Positive</td>
</tr>
<tr>
<td>Small</td>
<td>Green</td>
<td>Circle</td>
<td>Positive</td>
</tr>
<tr>
<td>Large</td>
<td>Red</td>
<td>Square</td>
<td>Positive</td>
</tr>
<tr>
<td>Large</td>
<td>Red</td>
<td>Circle</td>
<td>Positive</td>
</tr>
<tr>
<td>Small</td>
<td>Red</td>
<td>Square</td>
<td>Positive</td>
</tr>
<tr>
<td>Small</td>
<td>Red</td>
<td>Circle</td>
<td>Positive</td>
</tr>
<tr>
<td>Large</td>
<td>Blue</td>
<td>Square</td>
<td>Negative</td>
</tr>
<tr>
<td>Small</td>
<td>Blue</td>
<td>Square</td>
<td>Positive</td>
</tr>
<tr>
<td>Large</td>
<td>Blue</td>
<td>Circle</td>
<td>Positive</td>
</tr>
<tr>
<td>Small</td>
<td>Blue</td>
<td>Circle</td>
<td>Positive</td>
</tr>
</tbody>
</table>
A decision tree-induced partition

The red groups are negative examples, blue positive

Negative things are big, green shapes and big, blue squares
Learning decision trees

- **Goal:** Build decision tree to classify examples as positive or negative instances of concept using supervised learning from training data
- **A decision tree** is a tree where
  - non-leaf nodes have an attribute (feature)
  - leaf nodes have a classification (+ or -)
  - each arc has a possible value of its attribute
- **Generalization:** allow for >2 classes
  - e.g., classify stocks as {sell, hold, buy}
Expressiveness of Decision Trees

• Can express any function of 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>

• There’s a consistent decision tree for any training set with one path to leaf for each example, but it probably won't generalize to new examples

• Prefer more **compact** decision trees
Inductive learning and bias

• Suppose that we want to learn a function \( f(x) = y \) and we’re given sample \((x,y)\) pairs, as in figure (a)
• Can make several hypotheses about \( f \), e.g.: (b), (c) & (d)
• Preference reveals learning technique **bias**, e.g.:
  – prefer piece-wise functions (b)
  – prefer a smooth function (c)
  – prefer a simple function and treat outliers as noise (d)
Preference bias: Occam’s Razor

• William of Ockham (1285-1347)
  – “non sunt multiplicanda entia praeter necessitatem”
  – entities are not to be multiplied beyond necessity

• **Simplest** consistent explanation is the best

• **Smaller** decision trees correctly classifying training examples preferred over larger ones

• Finding the smallest decision tree is NP-hard, so we use algorithms that find reasonably small ones
R&N’s restaurant domain

• Develop decision tree that customers make when deciding whether to wait for a table or leave

• **Two classes**: wait, leave


• Set of **12 training examples**

• ~7000 possible cases
### Attribute-based representations

<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>

- Examples described by **attribute values** (Boolean, discrete, continuous), e.g., situations where will/won't wait for a table
- **Classification** of examples is **positive** (T) or **negative** (F)
- Serves as a training set
Decision tree from introspection

- **Patrons?**
  - None
    - F
  - Some
    - T
  - Full
    - Wait\(\text{Estimate}\)?
      - >60
        - F
      - 30–60
        - Alternate? (No)
      - 10–30
        - Alternate? (Yes)
      - 0–10
        - Hungry?
          - No
            - Reservation?
              - No
                - Bar?
                  - No
                    - F
                  - Yes
                    - T
              - Yes
                - T
          - Yes
            - Alternate? (No)
            - Raining?
              - No
                - F
              - Yes
                - T
            - Yes
              - Alternate? (No)
                - Bar?
                  - No
                    - F
                  - Yes
                    - T
              - Yes
                - Alternate? (Yes)
                  - Raining?
                    - No
                      - F
                    - Yes
                      - T
Issues

• It’s like 20 questions

• We can generate many decision trees depending on what attributes we ask about and in what order

• How do we decide?

• What makes one decision tree better than another: number of nodes? number of leaves? maximum depth?
ID3 / C4.5 / J48 Algorithm

• Greedy algorithm for decision tree construction developed by Ross Quinlan circa 1987

• Top-down construction of tree by recursively selecting best attribute to use at current node
  – Once attribute selected for current node, generate child nodes, one for each possible attribute value
  – Partition examples using values of attribute, & assign these subsets of examples to the child nodes
  – Repeat for each child node until examples associated with a node are all positive or negative
Choosing best attribute

• Key problem: choose attribute to split given set of examples

• Possibilities for choosing attribute:
  – Random: Select one at random
  – Least-values: one with smallest # of possible values
  – Most-values: one with largest # of possible values
  – Max-gain: one with largest expected information gain
  – Gini impurity: one with smallest gini impurity value

• The last two measure the homogeneity of the target variable within the subsets

• The ID3 algorithm uses max-gain
## Restaurant: type vs patrons?

**Random:** Patrons or Wait-time; **Least-values:** Patrons; **Most-values:** Type; **Max-gain:** ???

<table>
<thead>
<tr>
<th>Type variable</th>
<th>Patrons variable</th>
<th>Empty</th>
<th>Some</th>
<th>Full</th>
</tr>
</thead>
<tbody>
<tr>
<td>French</td>
<td>Y</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Italian</td>
<td>Y</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thai</td>
<td>N</td>
<td></td>
<td>Y</td>
<td>N Y</td>
</tr>
<tr>
<td>Burger</td>
<td>N</td>
<td></td>
<td>Y</td>
<td>N Y</td>
</tr>
</tbody>
</table>

---
Choosing an attribute

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

Which is better: *Patrons?* or *Type?*
Choosing an attribute

Idea: good attribute splits examples into subsets that are (ideally) *all positive* or *all negative*

- **Patrons**: for six examples we know the answer and for six we can predict with prob. 0.67
- **Type**: our prediction is no better than chance (0.50)
Splitting examples by testing attributes

(a) Patrons?
- None
  - +: x7, x11
  - -: x1, x3, x6, x8

(b) Type?
- French
  +: x1
  -: x5
- Italian
  +: x6
  -: x10
- Thai
  +: x4, x8
  -: x2, x11
- Burger
  +: x3, x12
  -: x7, x9

(c) Patrons?
- None
  - +: x7, x11
  - -: x1, x3, x6, x8
- Some
  - +: x1, x3, x6, x8
  - -: x4, x12
- Full
  - +: x4, x12
  -: x2, x10
ID3-induced decision tree
Compare the two Decision Trees

Human-generated decision tree

ID3-generated decision tree
Information theory 101

• Sprang fully formed from Claude Shannon’s seminal work: Mathematical Theory of Communication in 1948

• Intuitions
  – Common words (a, the, dog) shorter than less common ones (parliamentarian, foreshadowing)
  – Morse code: common letters have shorter encodings

• Information inherent in data/message (information entropy) measured in minimum number of bits needed to store/send using a good encoding
Information theory 101

• **Information entropy** ... tells how much information there is in an event. More uncertain an event is, more information it contains.

• Receiving a message is an event.

• How much information is in these messages:
  – The sun rose today!
  – It’s sunny today in Honolulu!
  – The coin toss is heads!
  – It’s sunny today in Seattle!
  – Life discovered on Mars!

  None
  A lot
Information theory 101

• For **n equally probable** possible messages or data values, each has probability $1/n$

• Information of a message is $-\log(p) = \log(n)$
  
  e.g., with 16 messages, then $\log(16) = 4$ and we need 4 bits to identify/send each message

• For **probability distribution** $P(p_1, p_2 \ldots p_n)$ for $n$ messages, its information (aka H or **entropy**) is:

  $$I(P) = -(p_1 \cdot \log(p_1) + p_2 \cdot \log(p_2) + \ldots + p_n \cdot \log(p_n))$$
Entropy of a distribution

\[ I(P) = -(p_1 \log(p_1) + p_2 \log(p_2) + \ldots + p_n \log(p_n)) \]

• Examples:
  – If \( P \) is (0.5, 0.5) then \( I(P) = 0.5 \times 1 + 0.5 \times 1 = 1 \)
  – If \( P \) is (0.67, 0.33) then \( I(P) = -(2/3 \log(2/3) + 1/3 \log(1/3)) = 0.92 \)
  – If \( P \) is (1, 0) then \( I(P) = 1 \times 1 + 0 \times \log(0) = 0 \)

• More **uniform probability** distribution, **greater its information**: more information is conveyed by a message telling you which event actually occurred

• Entropy is the average number of bits/message needed to represent a stream of messages
Example: Huffman code

• In 1952, MIT student David Huffman devised (for a homework assignment!) a coding scheme that’s optimal when all data probabilities are powers of 1/2

• A Huffman code can be built as followings:
  – Rank symbols in order of probability of occurrence
  – Successively combine 2 symbols of lowest probability to form new symbol; eventually we get binary tree where each node is probability of nodes below
  – Trace path to each leaf, noting direction at each node
Huffman code example

<table>
<thead>
<tr>
<th>M</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>.125</td>
</tr>
<tr>
<td>B</td>
<td>.125</td>
</tr>
<tr>
<td>C</td>
<td>.25</td>
</tr>
<tr>
<td>D</td>
<td>.5</td>
</tr>
</tbody>
</table>
If we use this code to many messages (A, B, C or D) with this probability distribution, then, over time, the average bits/message should approach 1.75.
To divide $T$ records into disjoint “answer” classes ($C_1..C_k$), the information needed to identify class of element of $T$ is:

$$\text{Info}(T) = I(P)$$

where $P$ is the probability distribution of partition ($C_1,C_2,..,C_k$):

$$\text{Info}(T) = (\frac{|C_1|}{|T|}, \frac{|C_2|}{|T|}, ..., \frac{|C_k|}{|T|})$$
Information for classification II

If we further divide $T$ w.r.t. attribute $X$ into sets $\{T_1, T_2, .., T_n\}$, the information needed to identify class of an element of $T$ is weighted average of information needed to identify class of an element of $T_i$, i.e., weighted average of $\text{Info}(T_i)$:

$$\text{Info}(X,T) = \sum \frac{|T_i|}{|T|} \times \text{Info}(T_i)$$

<table>
<thead>
<tr>
<th>C1</th>
<th>C2</th>
<th>C3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>High information</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C1</th>
<th>C2</th>
<th>C3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Low information</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Information gain

- $\text{Gain}(X,T) = \text{Info}(T) - \text{Info}(X,T)$ is difference of
  - info needed to identify element of $T$ and
  - info needed to identify element of $T$ after value of attribute $X$ known
- This is gain in information due to attribute $X$
- Used to rank attributes and build DT where each node uses attribute with greatest gain of those not yet considered in path from root
- goal: create small DTs to minimize questions
# Computing Information Gain

Should we ask about restaurant type or how many patrons there are?

- $I(T) = ?$
- $I(P, T) = ?$
- $I(\text{Type}, T) = ?$

Gain (Patrons, $T$) = ?
Gain (Type, $T$) = ?

$I(P) = -(p_1 \cdot \log(p_1) + p_2 \cdot \log(p_2) + \ldots + p_n \cdot \log(p_n))$
Computing information gain

\[ I(T) = \]
\[ - \left( \frac{1}{2} \log \frac{1}{2} + \frac{1}{2} \log \frac{1}{2} \right) \]
\[ = \frac{1}{2} + \frac{1}{2} = 1 \]

\[ I(P) = -(p_1 \log p_1 + p_2 \log p_2 + \ldots + p_n \log p_n) \]

\[ I(Pat, T) = \]
\[ \frac{2}{12} (0) + \frac{4}{12} (0) + \frac{6}{12} \left( \frac{4}{6} \log \frac{4}{6} + \frac{2}{6} \log \frac{2}{6} \right) \]
\[ = \frac{1}{2} \left( \frac{2}{3} \times 0.6 + \frac{1}{3} \times 1.6 \right) \]
\[ = 0.47 \]

\[ I(Type, T) = \]
\[ \frac{2}{12} (1) + \frac{2}{12} (1) + \frac{4}{12} (1) + \frac{4}{12} (1) = 1 \]

Gain (Patrons, T) = 1 - 0.47 = 0.53
Gain (Type, T) = 1 - 1 = 0
Computing information gain

\[ I(T) = - (0.5 \log 0.5 + 0.5 \log 0.5) = 0.5 + 0.5 = 1 \]

\[ I(P, T) = \frac{2}{12} (0) + \frac{4}{12} (0) + \frac{6}{12} (- (4/6 \log 4/6 + 2/6 \log 2/6)) \]
\[ = \frac{1}{2} (2/3 \times 0.6 + 1/3 \times 1.6) \]
\[ = 0.47 \]

\[ I(type, T) = \frac{2}{12} (1) + \frac{2}{12} (1) + \frac{4}{12} (1) + \frac{4}{12} (1) = 1 \]

Gain (Patrons, T) = 1 - 0.47 = 0.53
Gain (Type, T) = 1 - 1 = 0

\[ I(P) = -(p_1 \log p_1 + p_2 \log p_2 + \ldots + p_n \log p_n) \]
How well does it work?

Case studies show that decision trees often at least as accurate as human experts

– Study for diagnosing breast cancer had humans correctly classifying examples 65% of the time; DT classified 72% correct

– British Petroleum designed DT 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
Extensions of ID3

- Using alternate selection metric gain ratios, ...
- Real-valued data
- Noisy data and overfitting
- Generation of rules
- Setting parameters
- Cross-validation for experimental validation of performance
- **C4.5**: extension of ID3 accounting for unavailable values, continuous attribute value ranges, pruning of decision trees, rule derivation, etc.
Real-valued data?

Many ML systems work only on nominal data
• Select thresholds defining intervals so each becomes a discrete value of attribute
• Use heuristics: e.g., always divide into quartiles
• Use domain knowledge: e.g., divide age into infant (0-2), toddler (3-5), school-aged (5-8)
• Or treat this as another learning problem
  – Try different ways to discretize continuous variable; see which yield better results w.r.t. some metric
  – E.g., try midpoint between every pair of values
Noisy data?

ML systems must deal with *noise* in training data

- Two examples have same attribute/value pairs, but different classifications
- Some attribute values wrong due to errors in the data acquisition or preprocessing phase
- Classification is wrong (e.g., + instead of -) because of some error
- Some attributes irrelevant to decision-making, e.g., color of a die is irrelevant to its outcome

Bias in the training data is a related problem
If it’s cloudy, that’s a tank

• You may hear the story of a machine learning system designed to classify images as those with camouflaged tanks and those without
• It was trained on N images with tanks and M images with no tanks
• But the positive examples were all taken on a cloudy day; the negative on a sunny one
• System worked well, but had learned to detect the weather 😞

• The story is too good to be true; see Neural Net Tank Urban Legend
Overfitting 😞

- **Overfitting** occurs when a statistical model describes random error or noise instead of underlying relationship.

- If hypothesis space has many dimensions (many attributes) we may find **meaningless regularity** in data irrelevant to true distinguishing features. Students with an *m* in first name, born in July, & whose SSN digits sum to a prime number get better grades in AI.

- If we have **too little training data**, even a reasonable hypothesis space can overfit.
Avoiding Overfitting

• Remove obviously irrelevant features
  – E.g., remove ‘year observed’, ‘month observed’, ‘day observed’, ‘observer name’ from feature vector

• Getting more training data

• Pruning lower nodes in a decision tree
  – E.g., if gain of best attribute at a node is below a threshold, stop and make this node a leaf rather than generating children nodes
Pruning decision trees

• Pruning a decision tree is done by replacing a whole subtree by a leaf node.
• Replacement takes place if the expected error rate in the subtree is greater than in the single leaf, e.g.,
  – Training: 1 training red success and 2 training blue failures
  – Test: 3 red failures and one blue success
  – Consider replacing this subtree by a single Failure node.
• After replacement, only 2 errors instead of 5
Converting decision trees to rules

• Easy to get rules from decision tree: write rule for each path from the root to leaf

• Rule’s left-hand side built from the label of the nodes & labels of arcs

• Resulting rules set can be simplified:
  – Let LHS be the left hand side of a rule
  – LHS’ obtained from LHS by eliminating some conditions
  – Replace LHS by LHS' in this rule if the subsets of the training set satisfying LHS and LHS' are equal
  – A rule may be eliminated by using meta-conditions such as “if no other rule applies”
Summary: decision tree learning

• Widely used learning methods in practice for problems with relatively **few features**

• Strengths
  – Fast and simple to implement
  – Can convert result to a set of easily interpretable rules
  – Empirically valid in many commercial products
  – Handles noisy data
  – Easy for people to understand

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