Decision trees

- Extending previous approach:
  - to permit numeric attributes: straightforward
  - to deal sensibly with missing values: trickier
  - stability for noisy data: requires pruning mechanism
  - to handle regression
- End result: C4.5
  - Best-known and (probably) most widely-used learning algorithm
  - Commercial successor: C5.0

Review of Basic Strategy

- Strategy: top down
  Recursive divide-and-conquer fashion
  - First: select attribute for root node
    Create branches depending on the values of attribute at the root.
  - Then: split instances into subsets
    One for each branch extending from the node
  - Finally: repeat recursively for each branch, using only instances that reach the branch
- Stop if all instances have the same class

Review: Splitting Attribute

- Entropy function:
  \[ \text{entropy}(p_1, p_2, \ldots, p_n) = - \sum_{i=1}^{n} p_i \log p_i \]

- Example of information calculation:
  \[ \text{info}([2,3]) = \text{entropy}(2/5, 3/5) = -2/5 \log(2/5) - 3/5 \log(3/5) \]

- information gain = info[v] - info[children of v]
**Numeric attributes**

- Standard method: binary splits
  - E.g. temp < 45
- Unlike nominal attributes, every attribute has many possible split points
- Solution is straightforward extension:
  - Evaluate info gain for every possible split point of attribute
  - Choose “best” split point
  - Info gain for best split point is info gain for attribute
- Computationally more demanding

---

**Weather data (again!)**

<table>
<thead>
<tr>
<th>Outlook</th>
<th>Temperature</th>
<th>Humidity</th>
<th>Windy</th>
<th>Play</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>False</td>
<td>No</td>
</tr>
<tr>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>True</td>
<td>No</td>
</tr>
<tr>
<td>Overcast</td>
<td>Hot</td>
<td>High</td>
<td>False</td>
<td>Yes</td>
</tr>
<tr>
<td>Rainy</td>
<td>Mild</td>
<td>High</td>
<td>False</td>
<td>Yes</td>
</tr>
<tr>
<td>Rainy</td>
<td>Cool</td>
<td>Normal</td>
<td>False</td>
<td>Yes</td>
</tr>
<tr>
<td>Rainy</td>
<td>Cool</td>
<td>Normal</td>
<td>True</td>
<td>No</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

If outlook = sunny and humidity = high then play = no
If outlook = rainy and windy = true then play = no
If outlook = overcast then play = yes
If humidity = normal then play = yes
If none of the above then play = yes

---

**Example**

- Split on temperature attribute:
  
 64 65 68 69 70 71 72 75 75 80 81 83 85
  
  Yes No Yes Yes No No Yes Yes No Yes Yes No

  - E.g. temperature < 71.5: yes/4, no/2
    - temperature ≥ 71.5: yes/5, no/3

  - Info([4,2],[5,3])
    - = 6/14 info([4,2]) + 8/14 info([5,3])
    - = 0.939 bits

- Place split points halfway between values
- Can evaluate all split points in one pass!
Can avoid repeated sorting

- Sort instances by the values of the numeric attribute
- Does this have to be repeated at each node of the tree?
- No! Sort order for children can be derived from sort order for parent
  - Drawback: need to create and store an array of sorted indices for each numeric attribute

Binary vs multiway splits

- Splitting (multi-way) on a nominal attribute exhausts all information in that attribute
  - Nominal attribute is tested (at most) once on any path in the tree
- Not so for binary splits on numeric attributes!
  - Numeric attribute may be tested several times along a path in the tree
- Disadvantage: tree is hard to read
- Remedy:
  - pre-discretize numeric attributes, or
  - use multi-way splits instead of binary ones

Missing values

- Simplest strategy: send instances down the popular branch.
- More sophisticated: Split instances with missing values into pieces
  - A piece going down a branch receives a weight proportional to the popularity of the branch
  - weights sum to 1
  - During classification, split the instance into pieces in the same way

Pruning

- Prevent overfitting to noise in the data
- “Prune” the decision tree
- Two strategies:
  - Postpruning
take a fully-grown decision tree and discard unreliable parts
  - Prepruning
stop growing a branch when information becomes unreliable
- Postpruning preferred in practice—prepruning can “stop early”
Prepruning

- Based on statistical significance test
  - Stop growing the tree when there is no statistically significant association between any attribute and the class at a particular node
- Most popular test: chi-squared test (ID3)

Postpruning

- First, build full tree
- Then, prune it
  - Fully-grown tree shows all attribute interactions
  - How? determine whether some subtrees might be due to chance effects
  - Two pruning operations:
    - Subtree replacement
    - Subtree raising
  - Use error estimation or statistical techniques

Subtree replacement

- Bottom-up
- Consider replacing a tree only after considering all its subtrees

Subtree raising

- Delete node
- Redistribute instances
- Slower than subtree replacement (Worthwhile?)
Estimating error rates

- Prune only if it does not increase the estimated error.
- Error on the training data is NOT a useful estimator (would result in almost no pruning).
- Use hold-out set for pruning (“reduced-error pruning”).
- C4.5’s method
  - Derive confidence interval from training data.
  - Use a heuristic limit, derived from this, for pruning.
  - Standard Bernoulli-process-based method.

Regression trees

- Similar to decision trees but a leaf = average values of instances reaching leaf.
- Differences:
  - Splitting criterion: minimize intra-subset variation – can use standard deviation.
  - Termination criterion: std dev becomes small.
  - Prediction: Leaf predicts average class values of instances.
- More sophisticated version: model trees – each leaf represents a linear regression function.