Inferring rudimentary rules

- 1R: learns a 1-level decision tree
  - In other words, generates a set of rules that all test on one particular attribute

- Basic version (assuming nominal attributes)
  - One branch for each of the attribute’s values
  - Each branch assigns most frequent class
  - Error rate: proportion of instances that don’t belong to the majority class of their corresponding branch
  - Choose attribute with lowest error rate
Pseudo-code for 1R

For each attribute,
  For each value of the attribute, make a rule as follows:
    count how often each class appears
    find the most frequent class
    make the rule assign that class to this attribute-value
  Calculate the error rate of the rules
Choose the rules with the smallest error rate

- Note: “missing” is always treated as a separate attribute value
Evaluating the weather attributes

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Rules</th>
<th>Errors</th>
<th>Total errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outlook</td>
<td>Sunny → No</td>
<td>2/5</td>
<td>4/14</td>
</tr>
<tr>
<td></td>
<td>Overcast → Yes</td>
<td>0/4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Rainy → Yes</td>
<td>2/5</td>
<td></td>
</tr>
<tr>
<td>Temperature</td>
<td>Hot → No*</td>
<td>2/4</td>
<td>5/14</td>
</tr>
<tr>
<td></td>
<td>Mild → Yes</td>
<td>2/6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cool → Yes</td>
<td>1/4</td>
<td></td>
</tr>
<tr>
<td>Humidity</td>
<td>High → No</td>
<td>3/7</td>
<td>4/14</td>
</tr>
<tr>
<td></td>
<td>Normal → Yes</td>
<td>1/7</td>
<td></td>
</tr>
<tr>
<td>Windy</td>
<td>False → Yes</td>
<td>2/8</td>
<td>5/14</td>
</tr>
<tr>
<td></td>
<td>True → No*</td>
<td>3/6</td>
<td></td>
</tr>
</tbody>
</table>
Dealing with numeric attributes

- Numeric attributes are discretized: the range of the attribute is divided into a set of intervals
  - Instances are sorted according to attribute’s values
  - Breakpoints are placed where the (majority) class changes (so that the total error is minimized)
- Example: temperature from weather data

<table>
<thead>
<tr>
<th>64</th>
<th>65</th>
<th>68</th>
<th>69</th>
<th>70</th>
<th>71</th>
<th>72</th>
<th>72</th>
<th>75</th>
<th>75</th>
<th>80</th>
<th>81</th>
<th>83</th>
<th>85</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
The problem of overfitting

- Discretization procedure is very sensitive to noise
  - A single instance with an incorrect class label will most likely result in a separate interval
- Also: *time stamp* attribute will have zero errors
- Simple solution: enforce minimum number of instances in majority class per interval
- Weather data example (with minimum set to 3):

<table>
<thead>
<tr>
<th>64</th>
<th>65</th>
<th>68</th>
<th>69</th>
<th>70</th>
<th>71</th>
<th>72</th>
<th>72</th>
<th>75</th>
<th>75</th>
<th>80</th>
<th>81</th>
<th>83</th>
<th>85</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>No</td>
<td>☑</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>☑</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
Result of overfitting avoidance

- Final result for the temperature attribute:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Rules</th>
<th>Errors</th>
<th>Total errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outlook</td>
<td>Sunny → No</td>
<td>2/5</td>
<td>4/14</td>
</tr>
<tr>
<td></td>
<td>Overcast → Yes</td>
<td>0/4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Rainy → Yes</td>
<td>2/5</td>
<td></td>
</tr>
<tr>
<td>Temperature</td>
<td>≤ 77.5 → Yes</td>
<td>3/10</td>
<td>5/14</td>
</tr>
<tr>
<td></td>
<td>&gt; 77.5 → No*</td>
<td>2/4</td>
<td></td>
</tr>
<tr>
<td>Humidity</td>
<td>≤ 82.5 → Yes</td>
<td>1/7</td>
<td>3/14</td>
</tr>
<tr>
<td></td>
<td>&gt; 82.5 and ≤ 95.5 → No</td>
<td>2/6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&gt; 95.5 → Yes</td>
<td>0/1</td>
<td></td>
</tr>
<tr>
<td>Windy</td>
<td>False → Yes</td>
<td>2/8</td>
<td>5/14</td>
</tr>
<tr>
<td></td>
<td>True → No*</td>
<td>3/6</td>
<td></td>
</tr>
</tbody>
</table>

- Resulting rule sets:
Discussion of 1R

- 1R was described in a paper by Holte (1993)
  - Contains an experimental evaluation on 16 datasets (using cross-validation so that results were representative of performance on future data)
  - Minimum number of instances was set to 6 after some experimentation
  - 1R’s simple rules performed not much worse than much more complex decision trees
- Simplicity first pays off!
Constructing decision trees

- Normal procedure: top down in recursive *divide-and-conquer* fashion
  - First: attribute is selected for root node and branch is created for each possible attribute value
  - Then: the instances are split into subsets (one for each branch extending from the node)
  - Finally: procedure is repeated recursively for each branch, using only instances that reach the branch
- Process stops if all instances have the same class
Which attribute to select?
A criterion for attribute selection

- Which is the best attribute?
  - The one which will result in the smallest tree
  - Heuristic: choose the attribute that produces the “purest” nodes

- Popular *impurity criterion: information gain*
  - Information gain increases with the average purity of the subsets that an attribute produces

- Strategy: choose attribute that results in greatest information gain
Computing information

- Information is measured in *bits*
  - Given a probability distribution, the info required to predict an event is the distribution’s *entropy*
  - Entropy gives the information required in bits (this can involve fractions of bits!)

- Formula for computing the entropy:

  
  \[
  \text{entropy}(p_1, p_2, \ldots, p_n) = -p_1 \log p_1 - p_2 \log p_2 \ldots - p_n \log p_n
  \]
Example: attribute “Outlook”

- “Outlook” = “Sunny”:
  \[
  \text{info([2,3]) = entropy(2/5,3/5) = } -2/5 \log(2/5) - 3/5 \log(3/5) = 0.971 \text{ bits}
  \]

- “Outlook” = “Overcast”:
  \[
  \text{info([4,0]) = entropy(1,0) = } -1 \log(1) - 0 \log(0) = 0 \text{ bits}
  \]

- “Outlook” = “Rainy”:
  \[
  \text{info([3,2]) = entropy(3/5,2/5) = } -3/5 \log(3/5) - 2/5 \log(2/5) = 0.971 \text{ bits}
  \]

- Expected information for attribute:
  \[
  \text{info([3,2],[4,0],[3,2]) = } (5/14) \times 0.971 + (4/14) \times 0 + (5/14) \times 0.971
  = 0.693 \text{ bits}
  \]

Note: this is normally not defined.
Computing the information gain

- Information gain: information before splitting – information after splitting

\[
gain("Outlook") = \text{info}([9,5]) - \text{info}([2,3],[4,0],[3,2]) = 0.940 - 0.693 = 0.247 \text{ bits}
\]

- Information gain for attributes from weather data:

\[
gain("Outlook") = 0.247 \text{ bits}
\]
\[
gain("Temperature") = 0.029 \text{ bits}
\]
\[
gain("Humidity") = 0.152 \text{ bits}
\]
\[
gain("Windy") = 0.048 \text{ bits}
\]
Continuing to split

\[
\text{gain("Temperature")} = 0.571 \text{bits}
\]
\[
\text{gain("Humidity")} = 0.971 \text{bits}
\]
\[
\text{gain("Windy")} = 0.020 \text{bits}
\]
The final decision tree

- Note: not all leaves need to be pure; sometimes identical instances have different classes
  \[\Rightarrow\] Splitting stops when data can’t be split any further
Wishlist for a purity measure

- Properties we require from a purity measure:
  - When node is pure, measure should be zero
  - When impurity is maximal (i.e. all classes equally likely), measure should be maximal
  - Measure should obey *multistage property* (i.e. decisions can be made in several stages):
    \[
    \text{measure([23,4])} = \text{measure([27])} + (7/9) \times \text{measure([34])}
    \]
- Entropy is the only function that satisfies all three properties!
Some properties of the entropy

- **The multistage property:**

\[
\text{entropy}(p,q,r) = \text{entropy}(p,q+r) + (q+r) \times \text{entropy}\left(\frac{q}{q+r}, \frac{r}{q+r}\right)
\]

- **Simplification of computation:**

\[
\text{info}([2,3,4]) = -\frac{2}{9} \times \log\left(\frac{2}{9}\right) - \frac{3}{9} \times \log\left(\frac{3}{9}\right) - \frac{4}{9} \times \log\left(\frac{4}{9}\right) = \left[-2\log 2 - 3\log 3 - 4\log 4 + 9\log 9\right]/9
\]

- **Note:** instead of maximizing info gain we could just minimize information
Highly-branching attributes

- Problematic: attributes with a large number of values (extreme case: ID code)
- Subsets are more likely to be pure if there is a large number of values
  - Information gain is biased towards choosing attributes with a large number of values
  - This may result in overfitting (selection of an attribute that is non-optimal for prediction)
- Another problem: fragmentation
The weather data with ID code

<table>
<thead>
<tr>
<th>ID code</th>
<th>Outlook</th>
<th>Temp.</th>
<th>Humidity</th>
<th>Windy</th>
<th>Play</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>False</td>
<td>No</td>
</tr>
<tr>
<td>B</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>True</td>
<td>No</td>
</tr>
<tr>
<td>C</td>
<td>Overcast</td>
<td>Hot</td>
<td>High</td>
<td>False</td>
<td>Yes</td>
</tr>
<tr>
<td>D</td>
<td>Rainy</td>
<td>Mild</td>
<td>High</td>
<td>False</td>
<td>Yes</td>
</tr>
<tr>
<td>E</td>
<td>Rainy</td>
<td>Cool</td>
<td>Normal</td>
<td>False</td>
<td>Yes</td>
</tr>
<tr>
<td>F</td>
<td>Rainy</td>
<td>Cool</td>
<td>Normal</td>
<td>True</td>
<td>No</td>
</tr>
<tr>
<td>G</td>
<td>Overcast</td>
<td>Cool</td>
<td>Normal</td>
<td>True</td>
<td>Yes</td>
</tr>
<tr>
<td>H</td>
<td>Sunny</td>
<td>Mild</td>
<td>High</td>
<td>False</td>
<td>No</td>
</tr>
<tr>
<td>I</td>
<td>Sunny</td>
<td>Cool</td>
<td>Normal</td>
<td>False</td>
<td>Yes</td>
</tr>
<tr>
<td>J</td>
<td>Rainy</td>
<td>Mild</td>
<td>Normal</td>
<td>False</td>
<td>Yes</td>
</tr>
<tr>
<td>K</td>
<td>Sunny</td>
<td>Mild</td>
<td>Normal</td>
<td>True</td>
<td>Yes</td>
</tr>
<tr>
<td>L</td>
<td>Overcast</td>
<td>Mild</td>
<td>High</td>
<td>True</td>
<td>Yes</td>
</tr>
<tr>
<td>M</td>
<td>Overcast</td>
<td>Hot</td>
<td>Normal</td>
<td>False</td>
<td>Yes</td>
</tr>
<tr>
<td>N</td>
<td>Rainy</td>
<td>Mild</td>
<td>High</td>
<td>True</td>
<td>No</td>
</tr>
</tbody>
</table>
Tree stump for ID code attribute

- Entropy of split:

\[
\text{info("ID code")} = \text{info([0,1])} + \text{info([0,1])} + \ldots + \text{info([0,1])} = 0 \text{ bits}
\]

\[\Rightarrow\] Information gain is maximal for ID code (namely 0.940 bits)
The gain ratio

- *Gain ratio*: a modification of the information gain that reduces its bias
- Gain ratio takes number and size of branches into account when choosing an attribute
  - It corrects the information gain by taking the *intrinsic information* of a split into account
- Intrinsic information: entropy of distribution of instances into branches (i.e. how much info do we need to tell which branch an instance belongs to)
Computing the gain ratio

- Example: intrinsic information for ID code
  \[\text{info}(1,1,\ldots,1) = 14 \times (-1/14 \times \log_2 14) = 3.807 \text{ bits}\]

- Value of attribute decreases as intrinsic information gets larger

- Definition of gain ratio:
  \[\text{gain}_\text{ratio}(\text{"Attribute"}) = \frac{\text{gain}(\text{"Attribute"})}{\text{intrinsic info}(\text{"Attribute"})}\]

- Example:
  \[\text{gain}_\text{ratio}(\text{"ID_code"}) = \frac{0.940 \text{ bits}}{3.807 \text{ bits}} = 0.246\]
## Gain ratios for weather data

<table>
<thead>
<tr>
<th>Outlook</th>
<th>Temperature</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Info:</strong></td>
<td><strong>Info:</strong></td>
</tr>
<tr>
<td>0.693</td>
<td>0.911</td>
</tr>
<tr>
<td>Gain: 0.940-0.693</td>
<td>Gain: 0.940-0.911</td>
</tr>
<tr>
<td>0.247</td>
<td>0.029</td>
</tr>
<tr>
<td>Split info: info([5,4,5])</td>
<td>Split info: info([4,6,4])</td>
</tr>
<tr>
<td>1.577</td>
<td>1.362</td>
</tr>
<tr>
<td>Gain ratio: 0.247/1.577</td>
<td>Gain ratio: 0.029/1.362</td>
</tr>
<tr>
<td>0.156</td>
<td>0.021</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Humidity</th>
<th>Windy</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Info:</strong></td>
<td><strong>Info:</strong></td>
</tr>
<tr>
<td>0.788</td>
<td>0.892</td>
</tr>
<tr>
<td>Gain: 0.940-0.788</td>
<td>Gain: 0.940-0.892</td>
</tr>
<tr>
<td>0.152</td>
<td>0.048</td>
</tr>
<tr>
<td>Split info: info([7,7])</td>
<td>Split info: info([8,6])</td>
</tr>
<tr>
<td>1.000</td>
<td>0.985</td>
</tr>
<tr>
<td>Gain ratio: 0.152/1</td>
<td>Gain ratio: 0.048/0.985</td>
</tr>
<tr>
<td>0.152</td>
<td>0.049</td>
</tr>
</tbody>
</table>
More on the gain ratio

- “Outlook” still comes out top
- However: “ID code” has greater gain ratio
  - Standard fix: *ad hoc* test to prevent splitting on that type of attribute
- Problem with gain ratio: it may overcompensate
  - May choose an attribute just because its intrinsic information is very low
  - Standard fix: only consider attributes with greater than average information gain
Discussion

- Algorithm for top-down induction of decision trees ("ID3") was developed by Ross Quinlan
  - Gain ratio just one modification of this basic algorithm
  - Led to development of C4.5, which can deal with numeric attributes, missing values, and noisy data

- Similar approach: CART
- There are many other attribute selection criteria! (But almost no difference in accuracy of result.)
Covering algorithms

- Decision tree can be converted into a rule set
  - Straightforward conversion: rule set overly complex
  - More effective conversions are not trivial
- Strategy for generating a rule set directly: for each class in turn find rule set that covers all instances in it (excluding instances not in the class)
- This approach is called a *covering* approach because at each stage a rule is identified that covers some of the instances
Example: generating a rule

If \( x > 1.2 \) then class = a
If \( x > 1.2 \) and \( y > 2.6 \) then class = a
If \( x > 1.2 \) then class = a

- Possible rule set for class “b”:
  - If \( x \leq 1.2 \) then class = b
  - If \( x > 1.2 \) and \( y \leq 2.6 \) then class = b
- More rules could be added for “perfect” rule set
Rules vs. trees

- Corresponding decision tree: (produces exactly the same predictions)
- But: rule sets *can* be more perspicuous when decision trees suffer from replicated subtrees
- Also: in multiclass situations, covering algorithm concentrates on one class at a time whereas decision tree learner takes all classes into account
A simple covering algorithm

- Generates a rule by adding tests that maximize rule’s accuracy
- Similar to situation in decision trees: problem of selecting an attribute to split on
  - But: decision tree inducer maximizes overall purity
- Each new test reduces rule’s coverage:
Selecting a test

- Goal: maximizing accuracy
  - $t$: total number of instances covered by rule
  - $p$: positive examples of the class covered by rule
  - $t-p$: number of errors made by rule
  - Select test that maximizes the ratio $p/t$

- We are finished when $p/t = 1$ or the set of instances can’t be split any further
Example: contact lenses data

- **Rule we seek:** If ? then recommendation = hard
- **Possible tests:**
  - Age = Young 2/8
  - Age = Pre-presbyopic 1/8
  - Age = Presbyopic 1/8
  - Spectacle prescription = Myope 3/12
  - Spectacle prescription = Hypermetrope 1/12
  - Astigmatism = no 0/12
  - Astigmatism = yes 4/12
  - Tear production rate = Reduced 0/12
  - Tear production rate = Normal 4/12
Modified rule and resulting data

- Rule with best test added:
  
  If astigmatics = yes then recommendation = hard

- Instances covered by modified rule:

<table>
<thead>
<tr>
<th>Age</th>
<th>Spectacle prescription</th>
<th>Astigmatism</th>
<th>Tear production rate</th>
<th>Recommended lenses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Young</td>
<td>Myope</td>
<td>Yes</td>
<td>Reduced</td>
<td>None</td>
</tr>
<tr>
<td>Young</td>
<td>Myope</td>
<td>Yes</td>
<td>Normal</td>
<td>Hard</td>
</tr>
<tr>
<td>Young</td>
<td>Hypermetrope</td>
<td>Yes</td>
<td>Reduced</td>
<td>None</td>
</tr>
<tr>
<td>Young</td>
<td>Hypermetrope</td>
<td>Yes</td>
<td>Normal</td>
<td>hard</td>
</tr>
<tr>
<td>Pre-presbyopic</td>
<td>Myope</td>
<td>Yes</td>
<td>Reduced</td>
<td>None</td>
</tr>
<tr>
<td>Pre-presbyopic</td>
<td>Myope</td>
<td>Yes</td>
<td>Normal</td>
<td>Hard</td>
</tr>
<tr>
<td>Pre-presbyopic</td>
<td>Hypermetrope</td>
<td>Yes</td>
<td>Reduced</td>
<td>None</td>
</tr>
<tr>
<td>Pre-presbyopic</td>
<td>Hypermetrope</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Presbyopic</td>
<td>Myope</td>
<td>Yes</td>
<td>Reduced</td>
<td>None</td>
</tr>
<tr>
<td>Presbyopic</td>
<td>Myope</td>
<td>Yes</td>
<td>Normal</td>
<td>Hard</td>
</tr>
<tr>
<td>Presbyopic</td>
<td>Hypermetrope</td>
<td>Yes</td>
<td>Reduced</td>
<td>None</td>
</tr>
<tr>
<td>Presbyopic</td>
<td>Hypermetrope</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
</tbody>
</table>
Further refinement

- **Current state:**

  If astigmatism = yes and ? then
  recommendation = hard

- **Possible tests:**

<table>
<thead>
<tr>
<th>Test</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age = Young</td>
<td>2/4</td>
</tr>
<tr>
<td>Age = Pre-presbyopic</td>
<td>1/4</td>
</tr>
<tr>
<td>Age = Presbyopic</td>
<td>1/4</td>
</tr>
<tr>
<td>Spectacle prescription = Myope</td>
<td>3/6</td>
</tr>
<tr>
<td>Spectacle prescription = Hypermetrope</td>
<td>1/6</td>
</tr>
<tr>
<td>Tear production rate = Reduced</td>
<td>0/6</td>
</tr>
<tr>
<td>Tear production rate = Normal</td>
<td>4/6</td>
</tr>
</tbody>
</table>
Modified rule and resulting data

- Rule with best test added:
  
  If astigmatics = yes and tear production rate = normal
  then recommendation = hard

- Instances covered by modified rule:

<table>
<thead>
<tr>
<th>Age</th>
<th>Spectacle prescription</th>
<th>Astigmatism</th>
<th>Tear production rate</th>
<th>Recommended lenses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Young</td>
<td>Myope</td>
<td>Yes</td>
<td>Normal</td>
<td>Hard</td>
</tr>
<tr>
<td>Young</td>
<td>Hypermetrope</td>
<td>Yes</td>
<td>Normal</td>
<td>hard</td>
</tr>
<tr>
<td>Pre-presbyopic</td>
<td>Myope</td>
<td>Yes</td>
<td>Normal</td>
<td>Hard</td>
</tr>
<tr>
<td>Pre-presbyopic</td>
<td>Hypermetrope</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Presbyopic</td>
<td>Myope</td>
<td>Yes</td>
<td>Normal</td>
<td>Hard</td>
</tr>
<tr>
<td>Presbyopic</td>
<td>Hypermetrope</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
</tbody>
</table>
Further refinement

- **Current state:**
  - If astigmatism = yes and
  - tear production rate = normal and ?
  - then recommendation = hard

- **Possible tests:**
  - Age = Young 2/2
  - Age = Pre-presbyopic 1/2
  - Age = Presbyopic 1/2
  - Spectacle prescription = Myope 3/3
  - Spectacle prescription = Hypermetrope 1/3

- **Tie between the first and the fourth test**
  - We choose the one with greater coverage
The result

- **Final rule:**
  - If astigmatism = yes and
  - tear production rate = normal and
  - spectacle prescription = myope
  - then recommendation = hard

- **Second rule for recommending “hard lenses”:**
  - (built from instances not covered by first rule)
  - If age = young and astigmatism = yes and
  - tear production rate = normal then recommendation = hard

- **These two rules cover all “hard lenses”:**
  - Process is repeated with other two classes
Pseudo-code for PRISM

For each class C
  Initialize E to the instance set
  While E contains instances in class C
    Create a rule R with an empty left-hand side that predicts class C
    Until R is perfect (or there are no more attributes to use) do
      For each attribute A not mentioned in R, and each value v,
        Consider adding the condition A = v to the left-hand side of R
        Select A and v to maximize the accuracy p/t
        (break ties by choosing the condition with the largest p)
        Add A = v to R
    Remove the instances covered by R from E
Rules vs. decision lists

- PRISM with outer loop removed generates a decision list for one class
  - Subsequent rules are designed for rules that are not covered by previous rules
  - But: order doesn’t matter because all rules predict the same class
- Outer loop considers all classes separately
  - No order dependence implied
- Problems: overlapping rules, default rule required
Separate and conquer

- Methods like PRISM (for dealing with one class) are *separate-and-conquer* algorithms:
  - First, a rule is identified
  - Then, all instances covered by the rule are separated out
  - Finally, the remaining instances are “conquered”

- Difference to divide-and-conquer methods:
  - Subset covered by rule doesn’t need to be explored any further