Data Mining
Part 5. Prediction

5.4. Rule-Based Classification

Spring 2010

Instructor: Dr. Masoud Yaghini
Outline

- Using IF-THEN Rules for Classification
- Rule Extraction from a Decision Tree
- 1R Algorithm
- Sequential Covering Algorithms
- PRISM Algorithm
- FOIL Algorithm
- References
Using IF-THEN Rules for Classification
Using IF-THEN Rules for Classification

- A **rule-based classifier** uses a set of IF-THEN rules for classification.
- An IF-THEN rule is an expression of the form:
  
  \[
  \text{IF \hspace{5pt} condition \hspace{5pt} THEN \hspace{5pt} conclusion.}
  \]

  - where
    - Condition (or LHS) is rule antecedent/precondition
    - Conclusion (or RHS) is rule consequent
Using IF-THEN rules for classification

- An example is rule $R1$:
  
  $$R1: \text{IF } age = \text{youth AND student} = \text{yes THEN buys\_computer} = \text{yes}$$

  - The condition consists of one or more attribute tests that are logically ANDed
    - such as $age = \text{youth, and student} = \text{yes}$
  - The rule’s consequent contains a class prediction
    - we are predicting whether a customer will buy a computer

- $R1$ can also be written as
  
  $$R1: (age = \text{youth}) \land (student = \text{yes}) \Rightarrow (buys\_computer = \text{yes})$$
Assessment of a Rule

Assessment of a rule:

- **Coverage of a rule:**
  - The percentage of instances that satisfy the antecedent of a rule (i.e., whose attribute values hold true for the rule’s antecedent).

- **Accuracy of a rule:**
  - The percentage of instances that satisfy both the antecedent and consequent of a rule.
Rule Coverage and Accuracy

- **Rule accuracy and coverage:**

\[
\text{coverage}(R) = \frac{n_{\text{covers}}}{|D|}
\]

\[
\text{accuracy}(R) = \frac{n_{\text{correct}}}{n_{\text{covers}}}
\]

- **where**
  - \(D\): class labeled data set
  - \(|D|\): number of instances in \(D\)
  - \(n_{\text{covers}}\): number of instances covered by \(R\)
  - \(n_{\text{correct}}\): number of instances correctly classified by \(R\)

---

Rule-Based Classification
### Example: AllElectronics

<table>
<thead>
<tr>
<th>RID</th>
<th>age</th>
<th>income</th>
<th>student</th>
<th>credit_rating</th>
<th>Class: buys_computer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>youth</td>
<td>high</td>
<td>no</td>
<td>fair</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>youth</td>
<td>high</td>
<td>no</td>
<td>excellent</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>middle_aged</td>
<td>high</td>
<td>no</td>
<td>fair</td>
<td>yes</td>
</tr>
<tr>
<td>4</td>
<td>senior</td>
<td>medium</td>
<td>no</td>
<td>fair</td>
<td>yes</td>
</tr>
<tr>
<td>5</td>
<td>senior</td>
<td>low</td>
<td>yes</td>
<td>fair</td>
<td>yes</td>
</tr>
<tr>
<td>6</td>
<td>senior</td>
<td>low</td>
<td>yes</td>
<td>excellent</td>
<td>no</td>
</tr>
<tr>
<td>7</td>
<td>middle_aged</td>
<td>low</td>
<td>yes</td>
<td>excellent</td>
<td>yes</td>
</tr>
<tr>
<td>8</td>
<td>youth</td>
<td>medium</td>
<td>no</td>
<td>fair</td>
<td>no</td>
</tr>
<tr>
<td>9</td>
<td>youth</td>
<td>low</td>
<td>yes</td>
<td>fair</td>
<td>yes</td>
</tr>
<tr>
<td>10</td>
<td>senior</td>
<td>medium</td>
<td>yes</td>
<td>fair</td>
<td>yes</td>
</tr>
<tr>
<td>11</td>
<td>youth</td>
<td>medium</td>
<td>yes</td>
<td>excellent</td>
<td>yes</td>
</tr>
<tr>
<td>12</td>
<td>middle_aged</td>
<td>medium</td>
<td>no</td>
<td>excellent</td>
<td>yes</td>
</tr>
<tr>
<td>13</td>
<td>middle_aged</td>
<td>high</td>
<td>yes</td>
<td>fair</td>
<td>yes</td>
</tr>
<tr>
<td>14</td>
<td>senior</td>
<td>medium</td>
<td>no</td>
<td>excellent</td>
<td>no</td>
</tr>
</tbody>
</table>

### Rule-Based Classification
Coverage and Accuracy

- The rule R1:
  \[ R1: \text{IF} \ age = \text{youth AND student} = \text{yes THEN buys_computer} = \text{yes} \]
  - R1 covers 2 of the 14 instances
  - It can correctly classify both instances
- Therefore:
  - \( \text{Coverage}(R1) = \frac{2}{14} = 14.28\% \)
  - \( \text{Accuracy}(R1) = \frac{2}{2} = 100\% \).
Executing a rule set

- **Two ways of executing a rule set:**
  - Ordered set of rules (“decision list”)
    - Order is important for interpretation
  - Unordered set of rules
    - Rules may overlap and lead to different conclusions for the same instance
How We Can Use Rule-based Classification

- **Example:** We would like to classify $X$ according to $buys\_computer$:

  $$X = (age = youth, income = medium, student = yes, credit\_rating = fair).$$

- If a rule is satisfied by $X$, the rule is said to be **triggered**

- **Potential problems:**
  - If more than one rule is satisfied by $X$
    - Solution: *conflict resolution strategy*
  - If no rule is satisfied by $X$
    - Solution: *Use a default class*
Conflict Resolution

- Conflict resolution strategies:
  - Size ordering
  - Rule Ordering
    - Class-based ordering
    - Rule-based ordering

- **Size ordering** (rule antecedent size ordering)
  - Assign the highest priority to the triggering rules that is measured by the rule *precondition size*. (i.e., with the most attribute test)
  - the rules are unordered
Conflict Resolution

- **Class-based ordering:**
  - Decreasing order of **most frequent**
    - That is, all of the rules for the most frequent class come first, the rules for the next most frequent class come next, and so on.
  - Decreasing order of **misclassification cost per class**
  - Most popular strategy
Conflict Resolution

- **Rule-based ordering** *(Decision List)*:
  - Rules are organized into one long priority list, according to some measure of rule quality such as:
    - accuracy
    - coverage
    - by experts
Default Rule

- If no rule is satisfied by $X$:
  - A default rule can be set up to specify a default class, based on a training set.
  - This may be the class in majority or the majority class of the instances that were not covered by any rule.
  - The default rule is evaluated at the end, if and only if no other rule covers $X$.
  - The condition in the default rule is empty.
  - In this way, the rule fires when no other rule is satisfied.
Rule Extraction from a Decision Tree
Building Classification Rules

- Direct Method: extract rules directly from data
  - 1R Algorithm
  - Sequential covering algorithms
    - e.g.: PRISM, RIPPER, CN2, FOIL, and AQ

- Indirect Method: extract rules from other classification models
  - e.g. decision trees
Rule Extraction from a Decision Tree

- Decision trees can become large and difficult to interpret.
  - Rules are easier to understand than large trees
  - One rule is created for each path from the root to a leaf
  - Each attribute-value pair along a path forms a precondition: the leaf holds the class prediction
  - The order of the rules does not matter

- Rules are
  - **Mutually exclusive**: no two rules will be satisfied for the same instance
  - **Exhaustive**: there is one rule for each possible attribute-value combination
Example: *AllElectronics*

\[ \text{age?} \]

- youth
- middle_aged
- senior

\[ \text{student?} \]

- yes
- no

\[ \text{credit_rating?} \]

- fair
- excellent

\[ \begin{align*}
R1: \text{IF } & \text{age = youth AND student = no} & \text{THEN buys_computer = no} \\
R2: \text{IF } & \text{age = youth AND student = yes} & \text{THEN buys_computer = yes} \\
R3: \text{IF } & \text{age = middle_aged} & \text{THEN buys_computer = yes} \\
R4: \text{IF } & \text{age = senior AND credit_rating = excellent} & \text{THEN buys_computer = yes} \\
R5: \text{IF } & \text{age = senior AND credit_rating = fair} & \text{THEN buys_computer = no}
\end{align*} \]

**Rule-Based Classification**
Pruning the Rule Set

- The resulting set of rules extracted can be large and difficult to follow
  - Solution: pruning the rule set
- For a given rule any condition that does not improve the estimated accuracy of the rule can be pruned (i.e., removed)
- C4.5 extracts rules from an unpruned tree, and then prunes the rules using an approach similar to its tree pruning method
1R Algorithm
1R algorithm

- An easy way to find very simple classification rule
- 1R: rules that test one particular attribute
- Basic version
  - One branch for each value
  - 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 (*assumes nominal attributes*)
- “Missing” is treated as a separate attribute value
Pseudocode or 1R Algorithm

For each attribute,
   For each value of that 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.

Rule-Based Classification
**Example: The weather problem**

<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>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>

**Rule-Based Classification**
## Evaluating the weather attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Rules</th>
<th>Errors</th>
<th>Total errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 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>2 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>3 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>4 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>
The attribute with the smallest number of errors

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Rules</th>
<th>Errors</th>
<th>Total errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 outlook</td>
<td>sunny $\rightarrow$ no</td>
<td>2/5</td>
<td>4/14</td>
</tr>
<tr>
<td></td>
<td>overcast $\rightarrow$ yes</td>
<td>0/4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>rainy $\rightarrow$ yes</td>
<td>2/5</td>
<td></td>
</tr>
<tr>
<td>2 temperature</td>
<td>hot $\rightarrow$ no*</td>
<td>2/4</td>
<td>5/14</td>
</tr>
<tr>
<td></td>
<td>mild $\rightarrow$ yes</td>
<td>2/6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>cool $\rightarrow$ yes</td>
<td>1/4</td>
<td></td>
</tr>
<tr>
<td>3 humidity</td>
<td>high $\rightarrow$ no</td>
<td>3/7</td>
<td>4/14</td>
</tr>
<tr>
<td></td>
<td>normal $\rightarrow$ yes</td>
<td>1/7</td>
<td></td>
</tr>
<tr>
<td>4 windy</td>
<td>false $\rightarrow$ yes</td>
<td>2/8</td>
<td>5/14</td>
</tr>
<tr>
<td></td>
<td>true $\rightarrow$ no*</td>
<td>3/6</td>
<td></td>
</tr>
</tbody>
</table>

Rule-Based Classification
Dealing with numeric attributes

- Discretize numeric attributes
- Divide each attribute’s range into intervals
  - Sort instances according to attribute’s values
  - Place breakpoints where class changes (majority class)
  - This minimizes the total error
## Weather data with some numeric attributes

<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>85</td>
<td>85</td>
<td>false</td>
<td>no</td>
</tr>
<tr>
<td>sunny</td>
<td>80</td>
<td>90</td>
<td>true</td>
<td>no</td>
</tr>
<tr>
<td>overcast</td>
<td>83</td>
<td>86</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>rainy</td>
<td>70</td>
<td>96</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>rainy</td>
<td>68</td>
<td>80</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>rainy</td>
<td>65</td>
<td>70</td>
<td>true</td>
<td>no</td>
</tr>
<tr>
<td>overcast</td>
<td>64</td>
<td>65</td>
<td>true</td>
<td>yes</td>
</tr>
<tr>
<td>sunny</td>
<td>72</td>
<td>95</td>
<td>false</td>
<td>no</td>
</tr>
<tr>
<td>sunny</td>
<td>69</td>
<td>70</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>rainy</td>
<td>75</td>
<td>80</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>sunny</td>
<td>75</td>
<td>70</td>
<td>true</td>
<td>yes</td>
</tr>
<tr>
<td>overcast</td>
<td>72</td>
<td>90</td>
<td>true</td>
<td>yes</td>
</tr>
<tr>
<td>overcast</td>
<td>81</td>
<td>75</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>rainy</td>
<td>71</td>
<td>91</td>
<td>true</td>
<td>no</td>
</tr>
</tbody>
</table>

**Rule-Based Classification**
Example: temperature from weather data

<p>| | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>65</td>
<td>68</td>
<td>69</td>
<td>70</td>
<td>71</td>
<td>72</td>
<td>72</td>
<td>75</td>
<td>75</td>
<td>80</td>
<td>81</td>
<td>83</td>
</tr>
<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>no</td>
</tr>
</tbody>
</table>

- Discretization involves partitioning this sequence by placing breakpoints wherever the class changes,

<p>| | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></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>no</td>
</tr>
</tbody>
</table>

Rule-Based Classification
The problem of overfitting

- Overfitting is likely to occur whenever an attribute has a large number of possible values.
- This procedure is very sensitive to noise.
  - One instance with an incorrect class label will probably produce a separate interval.
- Attribute will have zero errors.
- **Simple solution**: enforce minimum number of instances in majority class per interval.
Minimum is set at 3 for temperature attribute

- The partitioning process begins
  
  yes no yes yes | yes ...

- The next example is also yes, we lose nothing by including that in the first partition
  
  yes no yes yes yes | no no yes yes yes | no yes yes no

- Thus the final discretization is
  
  yes no yes yes yes no no yes yes yes yes | no yes yes no

- The rule set

  temperature:  ≤ 77.5 → yes
  > 77.5 → no
## Resulting rule set with overfitting avoidance

<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>Windy</td>
<td>&gt; 95.5 → Yes</td>
<td>0/1</td>
<td></td>
</tr>
<tr>
<td></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>

**Rule-Based Classification**
Sequential Covering Algorithms
Sequential Covering Algorithms

A sequential covering algorithm:

- The rules are learned sequentially (one at a time)
- Each rule for a given class will ideally cover many of the instances of that class (and hopefully none of the instances of other classes).
- Each time a rule is learned, the instances covered by the rule are removed, and the process repeats on the remaining instances.
Sequential Covering Algorithms

**while** (enough target instances left)

- generate a rule
- remove positive target instances satisfying this rule

![Diagram showing rule-based classification with overlapping ellipses representing instances covered by different rules.](Diagram)

**Rule-Based Classification**
Sequential Covering Algorithms

- Typical Sequential covering algorithms:
  - PRISM
  - FOIL
  - AQ
  - CN2
  - RIPPER

- Sequential covering algorithms are the most widely used approach to mining classification rules

- Comparison with decision-tree induction:
  - Decision tree is learning a set of rules simultaneously
Basic Sequential Covering Algorithm

Algorithm: Sequential covering. Learn a set of IF-THEN rules for classification.

Input:

- $D$, a data set class-labeled tuples;
- $Att_{vals}$, the set of all attributes and their possible values.

Output: A set of IF-THEN rules.
Basic Sequential Covering Algorithm

Method:

(1) $Rule\_set = \{\}$; // initial set of rules learned is empty
(2) for each class $c$ do
(3)     repeat
(4)         $Rule = \text{Learn\_One\_Rule}(D, Att\_vals, c)$;
(5)     remove tuples covered by $Rule$ from $D$;
(6)     until terminating condition;
(7)     $Rule\_set = Rule\_set + Rule$; // add new rule to rule set
(8) endfor
(9) return $Rule\_Set$;
Basic Sequential Covering Algorithm

Steps:

- Rules are learned one at a time
- Each time a rule is learned, the instances covered by the rules are removed
- The process repeats on the remaining instances unless termination condition
  - e.g., when no more training examples or when the quality of a rule returned is below a user-specified level
Generating A Rule

• Typically, rules are grown in a *general-to-specific manner*
• We start with an empty rule and then gradually keep appending attribute tests to it.
• We append by adding the attribute test as a logical conjunct to the existing condition of the rule antecedent.
Example: Generating A Rule

- Example:
  - Suppose our training set, $D$, consists of loan application data.
  - Attributes regarding each applicant include their:
    - age
    - income
    - education level
    - residence
    - credit rating
    - the term of the loan.
  - The classifying attribute is *loan\_decision*, which indicates whether a loan is accepted (considered *safe*) or rejected (considered *risky*).
Example: Generating A Rule

- To learn a rule for the class “accept,” we start off with the most general rule possible, that is, the condition of the rule precondition is empty.
  - The rule is:
    
    \[
    \text{IF } \quad \text{THEN } \text{loan\textunderscore decision} = \text{accept}. \n    \]

- We then consider each possible attribute test that may be added to the rule.
Example: Generating A Rule

- Each time it is faced with adding a new attribute test to the current rule, it picks the one that most improves the rule quality, based on the training samples.
- The process repeats, where at each step, we continue to greedily grow rules until the resulting rule meets an acceptable quality level.
Example: Generating A Rule

- A general-to-specific search through rule space
Possible rule set for class “a”:
if true then class = a
Example: Generating A Rule

- Possible rule set for class “a”:

  If \( x > 1.2 \) then class = a
Example: Generating A Rule

- Possible rule set for class “a”:

  If $x > 1.2$ and $y > 2.6$ then class = a
Decision tree for the same problem

- Corresponding decision tree: (produces exactly the same predictions)

\[
\begin{align*}
\text{if } x > 1.2 \text{ then } & \text{go to node } b \\
\text{if } y > 2.6 \text{ then } & \text{go to node } a \\
\end{align*}
\]
Rules vs. trees

- Both methods might first split the dataset using the $x$ attribute and would probably end up splitting it at the same place ($x = 1.2$)
- But: **rule sets** can be more clear 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
PRISM Algorithm
PRISM Algorithm

- **PRISM method** generates a rule by adding tests that maximize rule’s accuracy.
- Each new test reduces rule’s coverage:

![Diagram showing the space of examples, rule so far, and rule after adding a new term.](image)

Rule-Based Classification
Selecting a test

- Goal: maximize 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 lens data

<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>no</td>
<td>reduced</td>
<td>none</td>
</tr>
<tr>
<td>young</td>
<td>myope</td>
<td>no</td>
<td>normal</td>
<td>soft</td>
</tr>
<tr>
<td>young</td>
<td>myope</td>
<td>yes</td>
<td>reduced</td>
<td>none</td>
</tr>
<tr>
<td>young</td>
<td>hypermetrope</td>
<td>no</td>
<td>reduced</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>soft</td>
</tr>
<tr>
<td>pre-presbyopic</td>
<td>myope</td>
<td>no</td>
<td>reduced</td>
<td>none</td>
</tr>
<tr>
<td>pre-presbyopic</td>
<td>myope</td>
<td>no</td>
<td>normal</td>
<td>soft</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>hypermetrope</td>
<td>no</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>no</td>
<td>normal</td>
<td>soft</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>no</td>
<td>reduced</td>
<td>none</td>
</tr>
<tr>
<td>presbyopic</td>
<td>myope</td>
<td>no</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>no</td>
<td>reduced</td>
<td>none</td>
</tr>
<tr>
<td>presbyopic</td>
<td>hypermetrope</td>
<td>no</td>
<td>normal</td>
<td>soft</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>

Rule-Based Classification
Example: contact lens data

- To begin, we seek a rule:
  
  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 = hypermetropia 1/12
  - astigmatism = no 0/12
  - astigmatism = yes 4/12
  - tear production rate = reduced 0/12
  - tear production rate = normal 4/12

Rule-Based Classification
Create the rule

- Rule with best test added and covered instances:

  If astigmatism = yes then recommendation = hard

<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>
<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:

  age = young  2/4
  age = pre-presbyopic  1/4
  age = presbyopic  1/4
  spectacle prescription = myope  3/6
  spectacle prescription = hypermetrope  1/6
  tear production rate = reduced  0/6
  tear production rate = normal  4/6
Modified rule and resulting data

- Rule with best test added:

  If astigmatism = 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 LHS 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 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
Separate and conquer

- Methods like PRISM (for dealing with one class) are *separate-and-conquer* algorithms:
  - First, identify a useful rule
  - Then, separate out all the instances it covers
  - Finally, “conquer” the remaining instances
FOIL Algorithm
(First Order Inductive Learner Algorithm)
Coverage or Accuracy?

Rule-Based Classification
Coverage or Accuracy?

- Consider the two rules:
  - $R_1$: correctly classifies 38 of the 40 instances it covers
  - $R_2$: covers only two instances, which it correctly classifies
- Their accuracies are 95% and 100%
- $R_2$ has greater accuracy than $R_1$, but it is not the better rule because of its small coverage
- Accuracy on its own is not a reliable estimate of rule quality
- Coverage on its own is not useful either
Consider Both Coverage and Accuracy

- If our current rule is $R$:
  
  $\text{IF } \text{condition} \text{ THEN } \text{class} = c$

- We want to see if logically ANDing a given attribute test to $\text{condition}$ would result in a better rule

- We call the new condition, $\text{condition}'$, where $R'$:
  
  $\text{IF } \text{condition}' \text{ THEN } \text{class} = c$
  
  - is our potential new rule

- In other words, we want to see if $R'$ is any better than $R$
FOIL Information Gain

- FOIL_Gain (in FOIL & RIPPER): assesses info_gain by extending condition

\[
 FOIL\_Gain = pos' \times (\log_2 \frac{pos'}{pos' + neg'} - \log_2 \frac{pos}{pos + neg})
\]

where

- \( pos, (neg) \) be the number of positive (negative) instances covered by \( R \)
- \( pos', (neg') \) be the number of positive (negative) instances covered by \( R' \)

- It favors rules that have **high accuracy** and cover **many positive instances**
To generate a rule

\[
\text{while}(\text{true})
\]

find the best predicate \( p \)

\[
\text{if } \text{FOIL_GAIN}(p) > \text{threshold} \text{ then add } p \text{ to current rule}
\]

\[
\text{else break}
\]
Rule Pruning: FOIL method

- Assessments of rule quality as described above are made with instances from the training data.
- Rule pruning based on an independent set of test instances.

\[
FOIL_{\text{Prune}}(R) = \frac{\text{pos} - \text{neg}}{\text{pos} + \text{neg}}
\]

- If \( FOIL_{\text{Prune}} \) is higher for the pruned version of \( R \), prune \( R \).
References

- J. Han, M. Kamber, *Data Mining: Concepts and Techniques*, Elsevier Inc. (2006). (Chapter 6)

The end