Machine Learning Techniques for Data Mining

Eibe Frank
University of Waikato
New Zealand
PART IV

Algorithms: The basic methods
Simplicity first

- Simple algorithms often work surprisingly well
- Many different kinds of simple structure exist:
  - One attribute might do all the work
  - All attributes might contribute independently with equal importance
  - A linear combination might be sufficient
  - An instance-based representation might work best
  - Simple logical structures might be appropriate
- Success of method depends on the domain!
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>Temp</th>
<th>Label</th>
<th>Temp</th>
<th>Label</th>
<th>Temp</th>
<th>Label</th>
<th>Temp</th>
<th>Label</th>
<th>Temp</th>
<th>Label</th>
<th>Temp</th>
<th>Label</th>
<th>Temp</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>Yes</td>
<td>65</td>
<td>No</td>
<td>68</td>
<td>Yes</td>
<td>69</td>
<td>Yes</td>
<td>70</td>
<td>Yes</td>
<td>71</td>
<td>Yes</td>
<td>72</td>
<td>Yes</td>
</tr>
<tr>
<td>72</td>
<td>Yes</td>
<td>75</td>
<td>Yes</td>
<td>75</td>
<td>Yes</td>
<td>80</td>
<td>Yes</td>
<td>81</td>
<td>Yes</td>
<td>83</td>
<td>Yes</td>
<td>85</td>
<td>Yes</td>
</tr>
<tr>
<td>85</td>
<td>No</td>
<td>72</td>
<td>Yes</td>
<td>72</td>
<td>Yes</td>
<td>75</td>
<td>Yes</td>
<td>75</td>
<td>Yes</td>
<td>80</td>
<td>Yes</td>
<td>81</td>
<td>Yes</td>
</tr>
<tr>
<td>83</td>
<td>Yes</td>
<td>83</td>
<td>Yes</td>
<td>83</td>
<td>Yes</td>
<td>85</td>
<td>Yes</td>
<td>85</td>
<td>Yes</td>
<td>85</td>
<td>Yes</td>
<td>85</td>
<td>Yes</td>
</tr>
<tr>
<td>85</td>
<td>No</td>
<td>72</td>
<td>Yes</td>
<td>72</td>
<td>Yes</td>
<td>75</td>
<td>Yes</td>
<td>75</td>
<td>Yes</td>
<td>80</td>
<td>Yes</td>
<td>81</td>
<td>Yes</td>
</tr>
<tr>
<td>83</td>
<td>Yes</td>
<td>83</td>
<td>Yes</td>
<td>83</td>
<td>Yes</td>
<td>85</td>
<td>Yes</td>
<td>85</td>
<td>Yes</td>
<td>85</td>
<td>Yes</td>
<td>85</td>
<td>Yes</td>
</tr>
</tbody>
</table>

10/25/2000
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>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>
**Result of overfitting avoidance**

- **Final result for 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:**

10/25/2000
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!
Statistical modeling

- “Opposite” of 1R: use all the attributes
- Two assumptions: Attributes are
  - equally important
  - statistically independent (given the class value)
    - This means that knowledge about the value of a particular attribute doesn’t tell us anything about the value of another attribute (if the class is known)
- Although based on assumptions that are almost never correct, this scheme works well in practice!
## Probabilities for the weather data

<table>
<thead>
<tr>
<th>Outlook</th>
<th>Temperature</th>
<th>Humidity</th>
<th>Windy</th>
<th>Play</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Sunny</td>
<td>2</td>
<td>3</td>
<td>Hot</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Overcast</td>
<td>4</td>
<td>0</td>
<td>Mild</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Rainy</td>
<td>3</td>
<td>2</td>
<td>Cool</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Outlook</th>
<th>Temp.</th>
<th>Humidity</th>
<th>windy</th>
<th>Play</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Sunny</td>
<td>Cool</td>
<td>High</td>
<td>True</td>
<td>?</td>
<td></td>
</tr>
</tbody>
</table>

### A new day:

- **Likelihood of the two classes**
  - For “yes” = $2/9 \times 3/9 \times 3/9 \times 9/14 = 0.0053$
  - For “no” = $3/5 \times 1/5 \times 4/5 \times 3/5 \times 5/14 = 0.0206$

Conversion into a probability by normalization:
- $P(“yes”) = 0.0053 / (0.0053 + 0.0206) = 0.205$
- $P(“no”) = 0.0206 / (0.0053 + 0.0206) = 0.795$
Bayes’s rule

- Probability of event $H$ given evidence $E$:

$$\Pr[H | E] = \frac{\Pr[E | H] \Pr[H]}{\Pr[E]}$$

- A priori probability of $H$: $\Pr[H]$
  - Probability of event before evidence has been seen
- A posteriori probability of $H$: $\Pr[H | E]$
  - Probability of event after evidence has been seen
Naïve Bayes for classification

- Classification learning: what’s the probability of the class given an instance?
  - Evidence $E = \text{instance}$
  - Event $H = \text{class value for instance}$

- Naïve Bayes assumption: evidence can be split into independent parts (i.e. attributes of instance!)

$$\text{Pr}[H \mid E] = \frac{\text{Pr}[E_1 \mid H] \text{Pr}[E_1 \mid H] \ldots \text{Pr}[E_n \mid H] \text{Pr}[H]}{\text{Pr}[E]}$$
The weather data example

<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>Cool</td>
<td>High</td>
<td>True</td>
<td>?</td>
</tr>
</tbody>
</table>

\[
\Pr[\text{yes} \mid E] = \frac{\Pr[\text{Outlook} = \text{Sunny} \mid \text{yes}] \times \Pr[\text{Temperature} = \text{Cool} \mid \text{yes}] \times \Pr[\text{Humidity} = \text{High} \mid \text{yes}] \times \Pr[\text{Windy} = \text{True} \mid \text{yes}] \times \frac{\Pr[\text{yes}]}{\Pr[E]}}{\Pr[E]}
\]

\[
= \frac{2/9 \times 3/9 \times 3/9 \times 3/9 \times 3/9 \times 9/14}{\Pr[E]}
\]
The “zero-frequency problem”

- What if an attribute value doesn’t occur with every class value (e.g. “Humidity = high” for class “yes”)?
  - Probability will be zero! \( \Pr[\text{Humidity} = \text{High} | \text{yes}] = 0 \)
  - A posteriori probability will also be zero! \( \Pr[\text{yes} | E] = 0 \)
    (No matter how likely the other values are!)
- Remedy: add 1 to the count for every attribute value-class combination (Laplace estimator)
- Result: probabilities will never be zero! (also: stabilizes probability estimates)
Modified probability estimates

- In some cases adding a constant different from 1 might be more appropriate
- Example: attribute *outlook* for class *yes*

\[
\begin{align*}
\text{Sunny: } & \frac{2 + \mu}{3} \frac{4 + \mu}{3} \frac{3 + \mu}{3} \\
\text{Overcast: } & \frac{9 + \mu}{9 + \mu} \frac{9 + \mu}{9 + \mu} \frac{9 + \mu}{9 + \mu} \\
\text{Rainy: } & \frac{9 + \mu}{9 + \mu} \frac{9 + \mu}{9 + \mu} \frac{9 + \mu}{9 + \mu}
\end{align*}
\]

- Weights don’t need to be equal (if they sum to 1)
Missing values

- Training: instance is not included in frequency count for attribute value-class combination
- Classification: attribute will be omitted from calculation
- Example:

<table>
<thead>
<tr>
<th>Outlook</th>
<th>Temp.</th>
<th>Humidity</th>
<th>Windy</th>
<th>Play</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>Cool</td>
<td>High</td>
<td>True</td>
<td>?</td>
</tr>
</tbody>
</table>

Likelihood of “yes” = \(\frac{3}{9} \times \frac{3}{9} \times \frac{3}{9} \times \frac{9}{14} = 0.0238\)

Likelihood of “no” = \(\frac{1}{5} \times \frac{4}{5} \times \frac{3}{5} \times \frac{5}{14} = 0.0343\)

\(P(“yes”) = \frac{0.0238}{0.0238 + 0.0343} = 41\%\)

\(P(“no”) = \frac{0.0343}{0.0238 + 0.0343} = 59\%\)
Dealing with numeric attributes

- Usual assumption: attributes have a normal or Gaussian probability distribution (given the class)
- The probability density function for the normal distribution is defined by two parameters:
  - The sample mean $\mu$: $\mu = \frac{1}{n} \sum_{i=1}^{n} x_i$
  - The standard deviation $\sigma$:
  - The density function $f(x)$:
    
    $f(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$
## Statistics for the weather data

<table>
<thead>
<tr>
<th>Outlook</th>
<th>Temperature</th>
<th>Humidity</th>
<th>Windy</th>
<th>Play</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Sunny</td>
<td>2</td>
<td>3</td>
<td>83</td>
<td>85</td>
</tr>
<tr>
<td>Overcast</td>
<td>4</td>
<td>0</td>
<td>70</td>
<td>80</td>
</tr>
<tr>
<td>Rainy</td>
<td>3</td>
<td>2</td>
<td>68</td>
<td>65</td>
</tr>
</tbody>
</table>

| Sunny      | 2/9 | 3/5 | mean | 73   | 74.6 | mean | 79.1 | 86.2 | False | 6/9 | 2/5 | 9/14 | 5/14 |
| Overcast   | 4/9 | 0/5 | std dev | 6.2 | 7.9  | std dev | 10.2 | 9.7  | True  | 3/9 | 3/5 |     |     |
| Rainy      | 3/9 | 2/5 |     |     |     |     |     |     |     |     |     |     |     |

- Example density value:

\[
f(\text{temperature} = 66 \mid \text{yes}) = \frac{1}{\sqrt{2\pi \cdot 6.2}} e^{-\frac{(66-73)^2}{2\cdot6.2^2}} = 0.0340
\]
Classifying a new day

A new day:

<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>66</td>
<td>90</td>
<td>true</td>
<td>?</td>
</tr>
</tbody>
</table>

Likelihood of “yes” = \( \frac{2}{9} \times 0.0340 \times 0.0221 \times \frac{3}{9} \times \frac{9}{14} = 0.000036 \)

Likelihood of “no” = \( \frac{3}{5} \times 0.0291 \times 0.0380 \times \frac{3}{5} \times \frac{5}{14} = 0.000136 \)

\[ P(“yes”) = \frac{0.000036}{0.000036 + 0.000136} = 20.9\% \]

\[ P(“no”) = \frac{0.000136}{0.000036 + 0.000136} = 79.1\% \]

Missing values during training: not included in calculation of mean and standard deviation
Probability densities

- Relationship between probability and density:

\[
\Pr\left[ c - \frac{\varepsilon}{2} < x < c + \frac{\varepsilon}{2} \right] \approx \varepsilon \ast f(c)
\]

- But: this doesn’t change calculation of *a posteriori* probabilities because \( \varepsilon \) cancels out

- Exact relationship:

\[
\Pr[a \leq x \leq b] = \int_{a}^{b} f(t) \, dt
\]
Discussion of Naïve Bayes

- Naïve Bayes works surprisingly well (even if independence assumption is clearly violated).
- Why? Because classification doesn’t require accurate probability estimates \(\textit{as long as maximum probability is assigned to correct class}\).
- However: adding too many redundant attributes will cause problems (e.g. identical attributes).
- Note also: many numeric attributes are not normally distributed (→ \textit{kernel density estimators}).
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”:
  \[
  \text{info}([2,3]) = \text{entropy}(2/5, 3/5) = -2/5 \log(2/5) - 3/5 \log(3/5) = 0.971 \text{ bits}
  \]

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

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

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

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

gain("Temperature") = 0.571 bits

gain("Humidity") = 0.971 bits

gain("Windy") = 0.020 bits
The final decision tree

- Note: not all leaves need to be pure; sometimes identical instances have different classes
  ⇒ 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):
    \[ \text{measure}([23,4]) = \text{measure}([27]) + \left(\frac{7}{9}\right) \times \text{measure}([34]) \]

- Entropy is the only function that satisfies all three properties!
Some properties of the entropy

- The multistage property:

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

- 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:

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

\[
\Rightarrow \text{Information gain is maximal for ID code (namely 0.940 bits)}
\]
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
  \[ \text{info}(1,1,\ldots,1) = 14 \times (-1/14 \times \log_{14} 1/14) = 3.807 \text{bits} \]

- Value of attribute decreases as intrinsic information gets larger

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

- 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>Info:</td>
<td>0.693</td>
</tr>
<tr>
<td>Gain: 0.940-0.693</td>
<td>0.247</td>
</tr>
<tr>
<td>Split info: info([5,4,5])</td>
<td>1.577</td>
</tr>
<tr>
<td>Gain ratio: 0.247/1.577</td>
<td>0.156</td>
</tr>
<tr>
<td>Humidity</td>
<td>Windy</td>
</tr>
<tr>
<td>Info:</td>
<td>0.788</td>
</tr>
<tr>
<td>Gain: 0.940-0.788</td>
<td>0.152</td>
</tr>
<tr>
<td>Split info: info([7,7])</td>
<td>1.000</td>
</tr>
<tr>
<td>Gain ratio: 0.152/1</td>
<td>0.152</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 true then class = a

If x > 1.2 then class = a

If x > 1.2 and y > 2.6 then class = a

Possible rule set for class “b”:

If x ≤ 1.2 then class = b
If x > 1.2 and y ≤ 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
  \[ \Rightarrow \] 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:
  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 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
Mining association rules

- Naïve method for finding association rules:
  - Using the standard separate-and-conquer method, treating every possible combination of attribute values as a separate class

- Two problems:
  - Computational complexity
  - Resulting number of rules (which would have to be pruned on the basis of support and confidence)

- But: we can look for high support rules directly!
**Item sets**

- Support: number of instances correctly covered by association rule
  - The same as the number of instances covered by \textit{all} tests in the rule (LHS and RHS!)
- \textit{Item}: one test/attribute-value pair
- \textit{Item set}: all items occurring in a rule
- Goal: only rules that exceed pre-defined support
  \[\Rightarrow\] We can do it by finding all item sets with the given minimum support and generating rules from them!
## Item sets for weather data

<table>
<thead>
<tr>
<th>One-item sets</th>
<th>Two-item sets</th>
<th>Three-item sets</th>
<th>Four-item sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outlook = Sunny (5)</td>
<td>Outlook = Sunny</td>
<td>Outlook = Sunny</td>
<td>Outlook = Sunny</td>
</tr>
<tr>
<td>Temperature = Mild (2)</td>
<td>Temperature = Hot</td>
<td>Humidity = High (2)</td>
<td>Temperature = Hot</td>
</tr>
<tr>
<td></td>
<td>Humidity = High</td>
<td>Play = No (2)</td>
<td>Humidity = High</td>
</tr>
<tr>
<td></td>
<td>Windy = False (2)</td>
<td></td>
<td>Play = False</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Play = Yes (2)</td>
</tr>
<tr>
<td>Temperature = Cool (4)</td>
<td>Outlook = Sunny</td>
<td>Outlook = Sunny</td>
<td>Outlook = Rainy</td>
</tr>
<tr>
<td>Humidity = High (3)</td>
<td>Temperature = High</td>
<td>Humidity = High</td>
<td>Temperature = Mild</td>
</tr>
<tr>
<td></td>
<td>Windy = False (2)</td>
<td></td>
<td>Windy = False</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Play = False</td>
</tr>
</tbody>
</table>

- **In total:** 12 one-item sets, 47 two-item sets, 39 three-item sets, 6 four-item sets and 0 five-item sets (with minimum support of two)
Generating rules from an item set

- Once all item sets with minimum support have been generated, we can turn them into rules
- Example: Humidity = Normal, Windy = False, Play = Yes (4)
- Seven \(2^{N-1}\) potential rules:

  - If Humidity = Normal and Windy = False then Play = Yes
  - If Humidity = Normal and Play = Yes then Windy = False
  - If Windy = False and Play = Yes then Humidity = Normal
  - If Humidity = Normal then Windy = False and Play = Yes
  - If Windy = False then Humidity = Normal and Play = Yes
  - If Play = Yes then Humidity = Normal and Windy = False
  - If True then Humidity = Normal and Windy = False and Play = Yes
Rules for the weather data

- Rules with support > 1 and confidence = 100%:

<table>
<thead>
<tr>
<th>Association rule</th>
<th>Sup.</th>
<th>Conf.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Humidity=Normal Windy=False ⇒Play=Yes</td>
<td>4</td>
<td>100%</td>
</tr>
<tr>
<td>2 Temperature=Cool ⇒Humidity=Normal</td>
<td>4</td>
<td>100%</td>
</tr>
<tr>
<td>3 Outlook=Overcast ⇒Play=Yes</td>
<td>4</td>
<td>100%</td>
</tr>
<tr>
<td>4 Temperature=Cold Play=Yes ⇒Humidity=Normal</td>
<td>3</td>
<td>100%</td>
</tr>
<tr>
<td>... ... ...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>58 Outlook=Sunny Temperature=Hot ⇒Humidity=High</td>
<td>2</td>
<td>100%</td>
</tr>
</tbody>
</table>

- In total: 3 rules with support four, 5 with support three, and 50 with support two
Example rules from the same set

- Item set:

  Temperature = Cool, Humidity = Normal, Windy = False, Play = Yes (2)

- Resulting rules (all with 100% confidence):

  Temperature = Cool, Windy = False ⇒ Humidity = Normal, Play = Yes
  Temperature = Cool, Windy = False, Humidity = Normal ⇒ Play = Yes
  Temperature = Cool, Windy = False, Play = Yes ⇒ Humidity = Normal

  due to the following “frequent” item sets:

  Temperature = Cool, Windy = False (2)
  Temperature = Cool, Humidity = Normal, Windy = False (2)
  Temperature = Cool, Windy = False, Play = Yes (2)

10/25/2000
Generating item sets efficiently

- How can we efficiently find all frequent item sets?
- Finding one-item sets easy
- Idea: use one-item sets to generate two-item sets, two-item sets to generate three-item sets, ...
  - If (A B) is frequent item set, then (A) and (B) have to be frequent item sets as well!
  - In general: if X is frequent k-item set, then all (k-1)-item subsets of X are also frequent
    \[\Rightarrow\] Compute k-item set by merging (k-1)-item sets
An example

- Given: five three-item sets
  \[(A \ B \ C), (A \ B \ D), (A \ C \ D), (A \ C \ E), (B \ C \ D)\]
- Lexicographically ordered!
- Candidate four-item sets:
  \[(A \ B \ C \ D) \quad OK \ because \ of \ (B \ C \ D)\]
  \[(A \ C \ D \ E) \quad Not \ OK \ because \ of \ (C \ D \ E)\]
- Final check by counting instances in dataset!
- \((k-1)\)-item sets are stored in hash table
Generating rules efficiently

- We are looking for all high-confidence rules
  - Support of antecedent obtained from hash table
  - But: brute-force method is \((2^N-1)\)
- Better way: building \((c + 1)\)-consequent rules from \(c\)-consequent ones
  - Observation: \((c + 1)\)-consequent rule can only hold if all corresponding \(c\)-consequent rules also hold
- Resulting algorithm similar to procedure for large item sets
Example

- 1-consequent rules:
  
  If Outlook = Sunny and Windy = False and Play = No  
  then Humidity = High (2/2)

  If Humidity = High and Windy = False and Play = No  
  then Outlook = Sunny (2/2)

- Corresponding 2-consequent rule:
  
  If Windy = False and Play = No  
  then Outlook = Sunny and Humidity = High (2/2)

- Final check of antecedent against hash table!
Discussion of association rules

- Above method makes one pass through the data for each different size item set
  - Other possibility: generate \((k+2)\)-item sets just after \((k+1)\)-item sets have been generated
  - Result: more \((k+2)\)-item sets than necessary will be considered but less passes through the data
  - Makes sense if data too large for main memory
- Practical issue: generating a certain number of rules (e.g. by incrementally reducing min. support)
Other issues

- ARFF format very inefficient for typical *market basket data*
  - Attributes represent items in a basket and most items are usually missing
- Instances are also called *transactions*
- Confidence is not necessarily the best measure
  - Example: milk occurs in almost every supermarket transaction
  - Other measures have been devised (e.g. lift)
Linear models

- Work most naturally with numeric attributes
- Standard technique for numeric prediction: linear regression
  - Outcome is linear combination of attributes
    \[ x = w_0 + w_1 a_1 + w_2 a_2 + \ldots + w_k a_k \]
- Weights are calculated from the training data
- Predicted value for first training instance \( a^{(1)} \)
  \[ w_0 a_0^{(1)} + w_1 a_1^{(1)} + w_2 a_2^{(1)} + \ldots + w_k a_k^{(1)} = \sum_{j=0}^{k} w_j a_j^{(1)} \]
Minimizing the squared error

- $k+1$ coefficients are chosen so that the squared error on the training data is minimized.

- Squared error: $\sum_{i=1}^{n} \left( x^{(i)} - \sum_{j=0}^{k} w_j a_j^{(i)} \right)$

- Coefficient can be derived using standard matrix operations.

- Can be done if there are more instances than attributes (roughly speaking).

- Minimization of absolute error is more difficult!
Classification

- Any regression technique can be used for classification
  - Training: perform a regression for each class, setting the output to 1 for training instances that belong to class, and 0 for those that don’t
  - Prediction: predict class corresponding to model with largest output value (membership value)
- For linear regression this is known as multi-response linear regression
Theoretical justification

Model Instance

Observed target value (either 0 or 1)

True class probability

The scheme minimizes this

\[ E_y \{(f(X) - Y)^2 \mid X = x} \]

\[ = E_y \{(f(X) - P(Y = 1 \mid X = x))^2 + 2(f(X) - P(Y = 1 \mid X = x)) \times 
\]

\[ E_y \{P(Y = 1 \mid X = x) - Y \mid X = x\} + E_y \{(P(Y = 1 \mid X = x) - Y)^2 \mid X = x} \]

\[ = (f(X) - P(Y = 1 \mid X = x))^2 + 2(f(X) - P(Y = 1 \mid X = x)) \times 
\]

\[ (P(Y = 1 \mid X = x) - E_y \{Y \mid X = x\}) + E_y \{(P(Y = 1 \mid X = x) - Y)^2 \mid X = x} \]

\[ = (f(X) - P(Y = 1 \mid X = x))^2 + E_y \{(P(Y = 1 \mid X = x) - Y)^2 \mid X = x} \]

We want to minimize this

Constant

10/25/2000
Pairwise regression

- Another way of using regression for classification:
  - A regression function for every pair of classes, using only instances from these two classes
  - An output of +1 is assigned to one member of the pair, an output of –1 to the other
- Prediction is done by voting
  - Class that receives most votes is predicted
  - Alternative: “don’t know” if there is no agreement
- More likely to be accurate but more expensive
Logistic regression

- Problem: some assumptions violated when linear regression is applied to classification problems
- **Logistic** regression: alternative to linear regression
  - Designed for classification problems
  - Tries to estimate class probabilities directly
    - Does this using the *maximum likelihood* method
  - Uses the following linear model:

\[
\log(P/(1-P)) = w_0 a_0 + w_1 a_1 + w_2 a_2 + \ldots + w_k a_k
\]

*Class probability*
Discussion of linear models

- Not appropriate if data exhibits non-linear dependencies
- But: can serve as building blocks for more complex schemes (i.e. model trees)
- Example: multi-response linear regression defines a *hyperplane* for any two given classes:

\[(w_0^{(1)} - w_0^{(2)})a_0 + (w_1^{(1)} - w_1^{(2)})a_1 + (w_2^{(1)} - w_2^{(2)})a_2 + \ldots + (w_k^{(1)} - w_k^{(2)})a_k > 0\]

- Obviously the same for pairwise linear regression
Instance-based learning

- Distance function defines what’s learned
- Most instance-based schemes use Euclidean distance:
  \[ \sqrt{(a_1^{(1)} - a_1^{(2)})^2 + (a_2^{(1)} - a_2^{(2)})^2 + \ldots + (a_k^{(1)} - a_k^{(2)})^2} \]

- \(a^{(1)}\) and \(a^{(2)}\): two instances with \(k\) attributes
- Taking the square root is not required when comparing distances
- Other popular metric: city-block metric
  - Adds differences without squaring them
Normalization and other issues

- Different attributes are measured on different scales ⇒ they need to be normalized:

\[ a_i = \frac{v_i - \min v_i}{\max v_i - \min v_i} \]

\(v_i\): the actual value of attribute \(i\)

- Nominal attributes: distance either 0 or 1
- Common policy for missing values: assumed to be maximally distant (given normalized attributes)
Discussion of 1-NN

- Often very accurate but also slow: simple version scans entire training data to derive a prediction
- Assumes all attributes are equally important
  - Remedy: attribute selection or weights
- Possible remedies against noisy instances:
  - Taking a majority vote over the $k$ nearest neighbors
  - Removing noisy instances from dataset (difficult!)
- Statisticians have used $k$-NN since early 1950s
  - If $n \to \infty$ and $k/n \to 0$, error approaches minimum
Comments on basic methods

- Bayes’ rule stems from his “Essay towards solving a problem in the doctrine of chances” (1763)
  - Difficult bit: estimating prior probabilities
  - Prior-free analysis generates confidence intervals
- Extension of Naïve Bayes: Bayesian Networks
- Algorithm for association rules is called APRIORI
- Minsky and Papert (1969) showed that linear classifiers have limitations, e.g. can’t learn XOR
  - But: combinations of them can (→ Neural Networks)