Data Mining
Practical Machine Learning Tools and Techniques

Simplicity first

- Simple algorithms often work very well!
- There are many kinds of simple structure, eg:
  - One attribute does all the work
  - All attributes contribute equally & independently
  - A weighted linear combination might do
  - Instance-based: use a few prototypes
  - Use simple logical rules
- Success of method depends on the domain

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 treated as a separate attribute value

Algorithms: The basic methods

- Inferring rudimentary rules
- Statistical modeling
- Constructing decision trees
- Constructing rules
- Association rule learning
- Linear models
- Instance-based learning
- Clustering

Inferring rudimentary rules

1R (Holte, 1993)

- 1R (Holte 1993): learns a 1-level decision tree
- i.e., 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)

Evaluating the weather attributes

For each attribute, classify the class as follows:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Rules</th>
<th>Errors</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>False</td>
</tr>
<tr>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>True</td>
</tr>
<tr>
<td>Overcast</td>
<td>Hot</td>
<td>High</td>
<td>False</td>
</tr>
<tr>
<td>Rainy</td>
<td>Mild</td>
<td>High</td>
<td>False</td>
</tr>
<tr>
<td>Rainy</td>
<td>Cool</td>
<td>Normal</td>
<td>True</td>
</tr>
<tr>
<td>Rainy</td>
<td>Cool</td>
<td>Normal</td>
<td>True</td>
</tr>
</tbody>
</table>

- 1 indicates a tie
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
- Example: temperature from 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>Sunny</td>
<td>85</td>
<td>85</td>
<td>False</td>
<td>No</td>
</tr>
<tr>
<td>Rainy</td>
<td>75</td>
<td>80</td>
<td>False</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<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>80</td>
<td>80</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>
</tbody>
</table>

With overfitting avoidance

- Resulting rule set:

<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>

Discussion of 1R: Hyperpipes

- Another simple technique: build one rule for each class
  - Each rule is a conjunction of tests, one for each attribute
  - For numeric attributes: test checks whether instance's value is inside an interval
    - Interval given by minimum and maximum observed in training data
  - For nominal attributes: test checks whether value is one of a subset of attribute values
    - Subset given by all possible values observed in training data
  - Class with most matching tests is predicted

The problem of overfitting

- This procedure is very sensitive to noise
  - One instance with an incorrect class label will probably produce a separate interval
- Also: time stamp attribute will have zero errors
- Simple solution: enforce minimum number of instances in majority class per interval

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)
    - i.e., knowing the value of one attribute says nothing about the value of another (if the class is known)
- Independence assumption is never correct!
- But ... this scheme works well in practice
### Probabilities for 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>Sunny</td>
<td>2</td>
<td>3</td>
<td>Hot</td>
<td>2</td>
</tr>
<tr>
<td>Overcast</td>
<td>4</td>
<td>0</td>
<td>Cold</td>
<td>3</td>
</tr>
<tr>
<td>Rainy</td>
<td>3</td>
<td>2</td>
<td>Cold</td>
<td>3</td>
</tr>
</tbody>
</table>

<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>3/9 2/5</td>
<td>Hot</td>
<td>2/5</td>
<td>High</td>
</tr>
<tr>
<td>Overcast</td>
<td>4/9 0/5</td>
<td>Mild</td>
<td>2/5</td>
<td>Normal</td>
</tr>
<tr>
<td>Rainy</td>
<td>3/9 2/5</td>
<td>Cool</td>
<td>3/9</td>
<td>1/5</td>
</tr>
</tbody>
</table>

### Bayes's rule

- **Probability of event $H$ given evidence $E$:**
  \[
  Pr[H|E] = \frac{Pr[E|H] \cdot Pr[H]}{Pr[E]}
  \]

- **A priori probability of $H$:** $Pr[H]$
- **Probability of event before evidence is seen**
- **A posteriori probability of $H$:** $Pr[H|E]$
- **Probability of event after evidence is seen**

**Thomas Bayes**

**Born:** 1701 in London, England  
**Died:** 1761 in Tunbridge Wells, Kent, England

---

### Naive Bayes for classification

- **Classification learning:** what’s the probability of the class given an instance?
  - **Evidence $E$ is instance**
  - **Event $H$ = class value for instance**
- **Naive assumption:** evidence splits into parts (i.e. attributes) that are **independent**

\[
Pr[H|E] = \frac{Pr[E_1|H] \cdot Pr[E_2|H] \cdots Pr[E_n|H] \cdot Pr[H]}{Pr[E]}
\]

---

### Weather data example

\[
Pr[yes|E] = Pr[Outlook=Sunny] \cdot Pr[Temperature=Cool] \cdot Pr[Humidity=High] \cdot Pr[Windy=True] \cdot Pr[E] 
= \frac{2 \cdot 3 \cdot 3 \cdot 3 \cdot 9 \cdot 9 \cdot 9 \cdot 9 \cdot 14}{Pr[E]}
\]

---

### The “zero-frequency problem”

- **What if an attribute value does not occur with every class value?**
  - (e.g. “Humidity = high” for class “yes”)
  - **Probability will be zero:** $Pr[Humidity=High|yes] = 0$
  - **A posterior probability will also be zero:** $Pr[yes|E] = 0$
  - (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

\[
\begin{align*}
2+\mu /3 & \quad 4+\mu /3 \quad 3+\mu /3 \\
9+\mu & \quad 9+\mu \quad 9+\mu
\end{align*}
\]

**Sunny** **Overcast** **Rainy**

- Weights don’t need to be equal (but they must sum to 1)

\[
\begin{align*}
2+\mu p_1 & \quad 4+\mu p_3 \quad 3+\mu p_2 \\
9+\mu & \quad 9+\mu \quad 9+\mu
\end{align*}
\]

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>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>True</td>
</tr>
<tr>
<td>Sunny</td>
<td>3</td>
<td>3</td>
<td>70</td>
<td>80</td>
</tr>
<tr>
<td>Overcast</td>
<td>4</td>
<td>0</td>
<td>120</td>
<td>True</td>
</tr>
<tr>
<td>Rainy</td>
<td>2</td>
<td>2</td>
<td>80</td>
<td>95</td>
</tr>
</tbody>
</table>

- Likelihood of “yes” = \(3/9 \times 3/9 \times 3/9 = 9/27 = 0.0438\)
- Likelihood of “no” = \(1/3 \times 1/3 \times 1/3 = 1/27 = 0.0438\)
- \(P(\text{"yes"}) = 0.0438 / (0.0438 + 0.0438) = 50\%\)
- \(P(\text{"no"}) = 0.0438 / (0.0438 + 0.0438) = 50\%\)

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:
  - Sample mean \(\mu = \frac{1}{n} \sum_{i=1}^{n} x_i\)
  - Standard dev \(\sigma = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (x_i - \mu)^2}\)
  - Then the density function

\[
f(x) = \frac{1}{\sqrt{2\pi} \sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}
\]

Statistics for 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>Sunny</td>
<td>66</td>
<td>0.65</td>
<td>70</td>
<td>True</td>
</tr>
<tr>
<td>Overcast</td>
<td>40</td>
<td>0.5</td>
<td>120</td>
<td>True</td>
</tr>
<tr>
<td>Rainy</td>
<td>40</td>
<td>0.2</td>
<td>80</td>
<td>True</td>
</tr>
</tbody>
</table>

- Example density value:

\[
f(\text{temperature}=66|\text{yes}) = \frac{1}{\sqrt{2\pi(0.65)^2}} e^{-\frac{(66-70)^2}{2(0.65)^2}} = 0.0340
\]

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>0.65</td>
<td>70</td>
<td>True</td>
</tr>
</tbody>
</table>

- Likelihood of “yes” = \(0.0340 \times 0.0221 \times 0.7 = 0.000036\)
- Likelihood of “no” = \(0.0181 \times 0.381 \times 0.3 = 0.000108\)
- \(P(\text{"yes"}) = 0.000036 / (0.000036 + 0.000108) = 25\%\)
- \(P(\text{"no"}) = 0.000108 / (0.000036 + 0.000108) = 75\%\)

- Missing values during training are not included in calculation of mean and standard deviation

Probability densities

- Relationship between probability and density:

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

- But: this doesn’t change calculation of a posteriori probabilities because \(\epsilon\) cancels out
- Exact relationship:

\[
Pr\left(a \leq x \leq b\right) = \int_a^b f(t) dt
\]
Multinomial naïve Bayes I

- Version of naïve Bayes used for document classification using bag of words model
- \( n_1, n_2, \ldots, n_k \): number of times word \( i \) occurs in document
- \( P_1, P_2, \ldots, P_k \): probability of obtaining word \( i \) when sampling from documents in class \( H \)
- Probability of observing document \( E \) given class \( H \) (based on multinomial distribution):
  \[ Pr[E|H] \approx M \times \prod_{i=1}^{k} \frac{p_i}{n_i} \]
- Ignores probability of generating a document of the right length (prob. assumed constant for each class)

Naïve Bayes: discussion

- Naïve Bayes works surprisingly well (even if independence assumption is clearly violated)
- Why? Because classification doesn’t require accurate probability estimates 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 (→ kernel density estimators)

Constructing decision trees

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

Which attribute to select?
**Criterion for attribute selection**

ID3: Iterative Dichotomizer 3 (Quinlan, 1986)

- Which is the best attribute?
  - Want to get 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
- Strategy: choose attribute that gives greatest information gain

**Computing information**

- Measure information 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 (can involve fractions of bits!)
- Formula for computing the entropy:
  \[
  \text{entropy}(p_1, p_2, \ldots, p_n) = - \sum_{i=1}^{n} p_i \log_2 p_i
  \]

**Example: attribute Outlook**

- **Outlook = Sunny**:
  \[
  \text{info}(2,3) = \text{entropy}(2/5,3/5) = -2/5 \log_2(2/5) - 3/5 \log_2(3/5) = 0.971 \text{ bits}
  \]
- **Outlook = Overcast**:
  \[
  \text{info}(4,0) = \text{entropy}(1,0) = -1 \log_2(1) - 0 \log_2(0) = 0 \text{ bits}
  \]
- **Outlook = Rainy**:
  \[
  \text{info}(3,2) = \text{entropy}(3/5,2/5) = -3/5 \log_2(3/5) - 2/5 \log_2(2/5) = 0.971 \text{ bits}
  \]
- **Expected information for attribute**:
  \[
  \text{info}(2,3,4,0,3,2) = (5/14) \times 0.971 + (4/14) \times 0 + (5/14) \times 0.971 = 0.693 \text{ bits}
  \]

**Example: attribute Temperature**

- **Temperature = Hot**:
  \[
  \text{info}(2,2) = \text{entropy}(2/4,2/4) = -2/4 \log_2(2/4) - 2/4 \log_2(2/4) = 1.0 \text{ bits}
  \]
- **Outlook = Mild**:
  \[
  \text{info}(4,2) = \text{entropy}(4/6,2/6) = -4/6 \log_2(4/6) - 2/6 \log_2(2/6) = 0.918 \text{ bits}
  \]
- **Outlook = Cool**:
  \[
  \text{info}(3,1) = \text{entropy}(3/4,1/4) = -3/4 \log_2(3/4) - 1/4 \log_2(1/4) = 0.811 \text{ bits}
  \]
- **Expected information for attribute**:
  \[
  \text{info}(2,2,4,2,3,1) = (4/14) \times 1 + (6/14) \times 0.918 + (4/14) \times 0.811 = 0.911 \text{ bits}
  \]

**Example: attribute Humidity**

- **Humidity = High**:
  \[
  \text{info}(3,4) = \text{entropy}(3/7,4/7) = -3/7 \log_2(3/7) - 4/7 \log_2(4/7) = 0.985 \text{ bits}
  \]
- **Humidity = Normal**:
  \[
  \text{info}(6,1) = \text{entropy}(6/7,1/7) = -6/7 \log_2(6/7) - 1/7 \log_2(1/7) = 0.592 \text{ bits}
  \]
- **Expected information for attribute**:
  \[
  \text{info}(3,4,6,1) = (7/14) \times 0.985 + (7/14) \times 0.592 = 0.788 \text{ bits}
  \]

**Example: attribute Windy**

- **Windy = False**:
  \[
  \text{info}(6,2) = \text{entropy}(6/8,2/8) = -6/8 \log_2(6/8) - 2/8 \log_2(2/8) = 0.811 \text{ bits}
  \]
- **Windy = True**:
  \[
  \text{info}(3,3) = \text{entropy}(3/6,3/6) = -3/6 \log_2(3/6) - 3/6 \log_2(3/6) = 1.0 \text{ bits}
  \]
- **Expected information for attribute**:
  \[
  \text{info}(6,2,3,3) = (8/14) \times 0.811 + (6/14) \times 1.0 = 0.892 \text{ bits}
  \]
Computing information gain

- Information gain: information before splitting – information after splitting
  \[ \text{gain(Outlook)} = \text{info}(9.51) - \text{info}([2, 3], [4, 0], [3, 2]) \]
  \[ = 0.940 - 0.693 \]
  \[ = 0.247 \text{ bits} \]

- Information gain for attributes from weather data:
  \[ \text{gain(Outlook)} = 0.247 \text{ bits} \]
  \[ \text{gain(Temperature)} = 0.029 \text{ bits} \]
  \[ \text{gain(Humidity)} = 0.152 \text{ bits} \]
  \[ \text{gain(Windy)} = 0.048 \text{ bits} \]

Splitting on Temperature

\[ \text{gain(Temperature)} = 0.571 \text{ bits} \]

Splitting on Humidity

\[ \text{gain(Humidity)} = 0.971 \text{ bits} \]

Splitting on Windy

\[ \text{gain(Windy)} = 0.020 \text{ bits} \]

Continuing to split

\[ \text{gain(Temperature)} = 0.571 \text{ bits} \]
\[ \text{gain(Humidity)} = 0.971 \text{ bits} \]
\[ \text{gain(Windy)} = 0.020 \text{ bits} \]

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):
    \[
    \text{measure}(\{2,3,4\}) = \text{measure}(\{2,7\}) + \frac{7}{9} \times \text{measure}(\{3,4\})
    \]
  - Entropy is the only function that satisfies all three properties!

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:
  \[
  \text{info}(\{2,3,4\}) = -\frac{2}{9} \times \log_2\left(\frac{2}{9}\right) - \frac{3}{9} \times \log_2\left(\frac{3}{9}\right) - \frac{4}{9} \times \log_2\left(\frac{4}{9}\right)
  = \frac{-2 \times \log_2 2 - 3 \times \log_2 3 - 4 \times \log_2 4 + 9 \times \log_2 9}{9}
  \]

- 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
  \(\Rightarrow\) Information gain is biased towards choosing attributes with a large number of values
  \(\Rightarrow\) This may result in overfitting (selection of an attribute that is non-optimal for prediction)
- Another problem: fragmentation

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>High</td>
<td>False</td>
<td>Yes</td>
<td></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>High</td>
<td>True</td>
<td>Yes</td>
<td></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}(\text{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 \text{ bits)}

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
  \(\Rightarrow\) 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
  \[ \text{info}(1, \ldots, 1) = 14 \times (-1/14 \times \log(1/14)) = 3.807 \text{bits} \]
- 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:
  \[ \text{gain}_\text{ratio}(\text{ID code}) = \frac{0.940}{3.807} = 0.246 \]

Gain ratios for weather data

<table>
<thead>
<tr>
<th>Outlook</th>
<th>Temperature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Info: 0.603</td>
<td>Info: 0.911</td>
</tr>
<tr>
<td>Gain: 0.940-0.693</td>
<td>Gain: 0.940-0.911</td>
</tr>
<tr>
<td>Split info: info([4.5,5])</td>
<td>Split info: info([4.6,4.9])</td>
</tr>
<tr>
<td>Gain ratio: 0.240/1.577</td>
<td>Gain ratio: 0.029/1.577</td>
</tr>
</tbody>
</table>

Humidity

<table>
<thead>
<tr>
<th>Outlook</th>
<th>Temperature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Info: 0.788</td>
<td>Info: 0.892</td>
</tr>
<tr>
<td>Gain: 0.940-0.788</td>
<td>Gain: 0.940-0.892</td>
</tr>
<tr>
<td>Split info: info([7.7])</td>
<td>Split info: info([8.6])</td>
</tr>
<tr>
<td>Gain ratio: 0.152/1</td>
<td>Gain ratio: 0.048/0.985</td>
</tr>
</tbody>
</table>

ID Code

<table>
<thead>
<tr>
<th>Outlook</th>
<th>Temperature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Info: 0.000</td>
<td>Info: 0.940</td>
</tr>
<tr>
<td>Gain: 0.940-0.000</td>
<td>Gain: 0.940</td>
</tr>
<tr>
<td>Split info: info([1,..,1])</td>
<td>Split info: info([1,..,1])</td>
</tr>
<tr>
<td>Gain ratio: 0.240</td>
<td>Gain ratio: 0.3807</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

- Top-down induction of decision trees: ID3, algorithm developed by Ross Quinlan
  - Gain ratio just one modification of this basic algorithm
  - \( \Rightarrow \) C4.5 (Quinlan, 1993)
  - \( \Rightarrow \) C4.5 deals with numeric attributes, missing values, noisy data
- Similar approach: CART (Classification and Regression Trees)
- There are many other attribute selection criteria!
  (But little difference in accuracy of result)

Covering algorithms

- Convert decision tree into a rule set
  - Straightforward, but rule set overly complex
  - More effective conversions are not trivial
- Instead, can generate rule set directly
  - for each class in turn find rule set that covers all instances in it (excluding instances not in the class)
- Called a covering approach:
  - at each stage a rule is identified that “covers” some of the instances

Example: generating a rule

If \( x > 1.2 \) then class = a
- Possible rule set for class “b”:
  - \( x \leq 1.2 \) then class = b
  - \( x > 1.2 \) and \( y \leq 2.6 \) then class = b
- Could add more rules, get “perfect” rule set
### Rules vs. trees

Corresponding decision tree:

```
  x > 1.2?  
  yes  no
   a   b
  y > 2.6?  
  yes  no
   b   a
```

- 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

### 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

  \[ \Rightarrow \text{Select test that maximizes the ratio } \frac{p}{t} \]
  - We are finished when $\frac{p}{t} = 1$ or the set of instances can’t be split any further

### Rule for Recommendation=Hard

- **Rule we seek:** IF ? THEN recommendation = hard

- **Possible tests:**
  - Age = Young
  - Age = Presbyopic
  - Spectacle prescription = Myope
  - Spectacle prescription = Hypermetrope
  - Astigmatism = no
  - Astigmatism = yes
  - Tear production rate = Reduced
  - Tear production rate = Normal

### 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:

### Example: Contact lenses 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>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>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>No</td>
<td>Normal</td>
<td>None</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>Reduced</td>
<td>None</td>
</tr>
<tr>
<td>Pre-presbyopic</td>
<td>Hypermetrope</td>
<td>Yes</td>
<td>Normal</td>
<td>Soft</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>Yes</td>
<td>Normal</td>
<td>None</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>Yes</td>
<td>Normal</td>
<td>Soft</td>
</tr>
</tbody>
</table>

### Instances where 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>No</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>No</td>
<td>Reduced</td>
<td>None</td>
</tr>
<tr>
<td>Pre-presbyopic</td>
<td>Myope</td>
<td>Yes</td>
<td>Normal</td>
<td>Soft</td>
</tr>
<tr>
<td>Presbyopic</td>
<td>Myope</td>
<td>Yes</td>
<td>Normal</td>
<td>None</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>Yes</td>
<td>Normal</td>
<td>Soft</td>
</tr>
</tbody>
</table>

Condereraka (1998)
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>Hypermetropia</td>
<td>Yes</td>
<td>Normal</td>
<td>Hard</td>
</tr>
<tr>
<td>Pre-presbyptic</td>
<td>Hypermetropia</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Pre-presbyptic</td>
<td>Myope</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Presbyptic</td>
<td>Hypermetropia</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Presbyptic</td>
<td>Myope</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
  THEN recommendation = hard

- Possible tests:

<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>Hypermetropia</td>
<td>Yes</td>
<td>Normal</td>
<td>Hard</td>
</tr>
<tr>
<td>Pre-presbyptic</td>
<td>Hypermetropia</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Pre-presbyptic</td>
<td>Myope</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Presbyptic</td>
<td>Hypermetropia</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Presbyptic</td>
<td>Myope</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Pre-presbyptic</td>
<td>Hypermetropia</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Pre-presbyptic</td>
<td>Myope</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Pre-presbyptic</td>
<td>Hypermetropia</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Presbyptic</td>
<td>Hypermetropia</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Presbyptic</td>
<td>Myope</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Pre-presbyptic</td>
<td>Hypermetropia</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Pre-presbyptic</td>
<td>Myope</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Pre-presbyptic</td>
<td>Hypermetropia</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Presbyptic</td>
<td>Hypermetropia</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Presbyptic</td>
<td>Myope</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Pre-presbyptic</td>
<td>Hypermetropia</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Pre-presbyptic</td>
<td>Myope</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Presbyptic</td>
<td>Hypermetropia</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Presbyptic</td>
<td>Myope</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Pre-presbyptic</td>
<td>Hypermetropia</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Pre-presbyptic</td>
<td>Myope</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Presbyptic</td>
<td>Hypermetropia</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Presbyptic</td>
<td>Myope</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
</tbody>
</table>

"Preliminary" result

- Final rule:  
  IF  astigmatism = yes AND
  tear production rate = normal AND
  spectacle prescription = myope
  THEN recommendation = hard

- Unfortunately, the above rule does not cover all instances where recommendation=hard!

- We need to repeat the process on all instances not covered by the above rule ...

Further refinement

- Current state:  
  IF  astigmatism = yes AND
  tear production rate = normal AND
  spectacle prescription = myope
  THEN recommendation = hard

- Possible tests:

<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>Hypermetropia</td>
<td>Yes</td>
<td>Normal</td>
<td>Hard</td>
</tr>
<tr>
<td>Pre-presbyptic</td>
<td>Hypermetropia</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Pre-presbyptic</td>
<td>Myope</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Presbyptic</td>
<td>Hypermetropia</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Presbyptic</td>
<td>Myope</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
</tbody>
</table>

Instances not covered ...

<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>Hypermetropia</td>
<td>Yes</td>
<td>Normal</td>
<td>Hard</td>
</tr>
<tr>
<td>Pre-presbyptic</td>
<td>Hypermetropia</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Pre-presbyptic</td>
<td>Myope</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Presbyptic</td>
<td>Hypermetropia</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Presbyptic</td>
<td>Myope</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Pre-presbyptic</td>
<td>Hypermetropia</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Pre-presbyptic</td>
<td>Myope</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Pre-presbyptic</td>
<td>Hypermetropia</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Presbyptic</td>
<td>Hypermetropia</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Presbyptic</td>
<td>Myope</td>
<td>Yes</td>
<td>Normal</td>
<td>None</td>
</tr>
</tbody>
</table>
Repeat the process...

<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>Myopic</td>
<td>No</td>
<td>Reduced</td>
<td>None</td>
</tr>
<tr>
<td>Young</td>
<td>Myopic</td>
<td>No</td>
<td>Normal</td>
<td>Soft</td>
</tr>
<tr>
<td>Young</td>
<td>Myopic</td>
<td>Yes</td>
<td>Reduced</td>
<td>None</td>
</tr>
<tr>
<td>Young</td>
<td>Hypermetropic</td>
<td>No</td>
<td>Reduced</td>
<td>None</td>
</tr>
<tr>
<td>Young</td>
<td>Hypermetropic</td>
<td>Yes</td>
<td>Normal</td>
<td>Soft</td>
</tr>
<tr>
<td>Pre-presbyopic</td>
<td>Myopic</td>
<td>No</td>
<td>Normal</td>
<td>Soft</td>
</tr>
<tr>
<td>Pre-presbyopic</td>
<td>Myopic</td>
<td>Yes</td>
<td>Reduced</td>
<td>None</td>
</tr>
<tr>
<td>Pre-presbyopic</td>
<td>Hypermetropic</td>
<td>No</td>
<td>Reduced</td>
<td>None</td>
</tr>
<tr>
<td>Pre-presbyopic</td>
<td>Hypermetropic</td>
<td>Yes</td>
<td>Reduced</td>
<td>None</td>
</tr>
<tr>
<td>Presbyopic</td>
<td>Myopic</td>
<td>No</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Presbyopic</td>
<td>Myopic</td>
<td>Yes</td>
<td>Reduced</td>
<td>None</td>
</tr>
<tr>
<td>Presbyopic</td>
<td>Hypermetropic</td>
<td>No</td>
<td>Reduced</td>
<td>None</td>
</tr>
<tr>
<td>Presbyopic</td>
<td>Hypermetropic</td>
<td>Yes</td>
<td>Normal</td>
<td>Soft</td>
</tr>
</tbody>
</table>

What is the second rule for Recommendation=Hard?

Rule #2 for Recommendation=Hard

- Rule we seek: IF ? THEN recommendation = hard
- Possible tests:
  - Age = Young 1/7
  - Age = Pre-presbyopic 0/7
  - Age = Presbyopic 0/7
  - Spectacle prescription = Myopic 0/3
  - Spectacle prescription = Hypermetropic 1/12
  - Astigmatism = no 0/12
  - Astigmatism = yes 1/8
  - Tear production rate = Reduced 0/12
  - Tear production rate = Normal 1/8

Modified rule #2 and resulting data

- Rule with best test added:
  IF age = young 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>Myopic</td>
<td>No</td>
<td>Reduced</td>
<td>None</td>
</tr>
<tr>
<td>Young</td>
<td>Myopic</td>
<td>No</td>
<td>Normal</td>
<td>Soft</td>
</tr>
<tr>
<td>Young</td>
<td>Myopic</td>
<td>Yes</td>
<td>Reduced</td>
<td>None</td>
</tr>
<tr>
<td>Young</td>
<td>Hypermetropic</td>
<td>No</td>
<td>Reduced</td>
<td>None</td>
</tr>
<tr>
<td>Young</td>
<td>Hypermetropic</td>
<td>Yes</td>
<td>Normal</td>
<td>Soft</td>
</tr>
<tr>
<td>Pre-presbyopic</td>
<td>Myopic</td>
<td>No</td>
<td>Normal</td>
<td>Soft</td>
</tr>
<tr>
<td>Pre-presbyopic</td>
<td>Myopic</td>
<td>Yes</td>
<td>Reduced</td>
<td>None</td>
</tr>
<tr>
<td>Pre-presbyopic</td>
<td>Hypermetropic</td>
<td>No</td>
<td>Reduced</td>
<td>None</td>
</tr>
<tr>
<td>Pre-presbyopic</td>
<td>Hypermetropic</td>
<td>Yes</td>
<td>Reduced</td>
<td>None</td>
</tr>
<tr>
<td>Presbyopic</td>
<td>Myopic</td>
<td>No</td>
<td>Normal</td>
<td>None</td>
</tr>
<tr>
<td>Presbyopic</td>
<td>Myopic</td>
<td>Yes</td>
<td>Reduced</td>
<td>None</td>
</tr>
<tr>
<td>Presbyopic</td>
<td>Hypermetropic</td>
<td>No</td>
<td>Reduced</td>
<td>None</td>
</tr>
<tr>
<td>Presbyopic</td>
<td>Hypermetropic</td>
<td>Yes</td>
<td>Normal</td>
<td>Soft</td>
</tr>
</tbody>
</table>

First refinement

- Current state: IF age = young AND ? THEN recommendation = hard
- Possible tests:
  - Spectacle prescription = Myopic 0/3
  - Spectacle prescription = Hypermetropic 1/4
  - Astigmatism = no 0/4
  - Astigmatism = yes 1/8
  - Tear production rate = Reduced 0/4
  - Tear production rate = Normal 1/3

Modified rule #2 and resulting data

- Rule with best test added:
  IF age = young AND astigmatism = 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>Myopic</td>
<td>No</td>
<td>Reduced</td>
<td>None</td>
</tr>
<tr>
<td>Young</td>
<td>Myopic</td>
<td>No</td>
<td>Normal</td>
<td>Soft</td>
</tr>
<tr>
<td>Young</td>
<td>Myopic</td>
<td>Yes</td>
<td>Reduced</td>
<td>None</td>
</tr>
<tr>
<td>Young</td>
<td>Hypermetropic</td>
<td>No</td>
<td>Reduced</td>
<td>None</td>
</tr>
<tr>
<td>Young</td>
<td>Hypermetropic</td>
<td>Yes</td>
<td>Normal</td>
<td>Soft</td>
</tr>
</tbody>
</table>

Further refinement

- Current state: IF age = young AND astigmatism = yes AND ? THEN recommendation = hard
- Possible tests:
  - Spectacle prescription = Myopic 0/1
  - Spectacle prescription = Hypermetropic 1/2
  - Tear production rate = Reduced 0/2
  - Tear production rate = Normal 1/3
**Final result for “Hard”**

- **First 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  
  - Generates 3 “soft”, 2 “hard”, and 4 “none” rules

**Pseudo-code for PRISM**

**PRISM (Cendrowska, 1987)**

For each class C
- Initialize R to the instance set
- While R 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 tie 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, identify a useful rule  
  - Then, separate out all the instances it covers  
  - Finally, “conquer” the remaining instances  
- **Difference to divide-and-conquer methods:**  
  - Subset covered by rule doesn’t need to be explored any further

**Mining association rules**

*Aprieti (Agrawal et al., 1993)*

- **Naive method for finding association rules:**  
  - Use separate-and-conquer method  
  - Treat 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 all tests in the rule (LHS and RHS)  
- **Item:** one test/attribute-value pair  
- **Item set:** all items occurring in a rule  
- **Goal:** only rules that exceed pre-defined support  
  ⇒ Do it by finding all item sets with the given minimum support and generating rules from them
### Generating item sets for the weather data

<table>
<thead>
<tr>
<th>1-item sets</th>
<th>2-item sets</th>
<th>3-item sets</th>
<th>4-item sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wet = Sunny (3)</td>
<td>Wet = Sunny</td>
<td>Wet = Sunny</td>
<td>Wet = Sunny</td>
</tr>
<tr>
<td>Temperature = Hot (2)</td>
<td>Temperature = Hot</td>
<td>Temperature = Hot</td>
<td>Play = No (2)</td>
</tr>
<tr>
<td>Temperature = Cool (4)</td>
<td>Outlook = Sunny</td>
<td>Humidity = High (3)</td>
<td>Outlook = Sunny</td>
</tr>
<tr>
<td>Humidity = High (2)</td>
<td>Wind = False (2)</td>
<td>Wind = False</td>
<td>Play = Yes (2)</td>
</tr>
</tbody>
</table>

- In total: 12 one-item sets, 47 two-item sets, 39 three-item sets, 8 four-item sets and 0 five-item sets (with minimum support of two)

### Rules for weather data

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

<table>
<thead>
<tr>
<th>Association rule</th>
<th>Supp.</th>
<th>Cond.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Humidity=Normal, Windy=False</td>
<td>Play=Yes</td>
<td>4</td>
</tr>
<tr>
<td>Temperature=Hot</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>Outlook=Sunny</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>Temperature=Cold, Play=Yes</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>Outlook=Overcast</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>Temperature=Hot, Humidity=Normal</td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>

- In total: 3 rules with support four 5 with support three 50 with support 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 4/4
    - If Humidity = Normal and Play = Yes then Windy = False 4/6
    - If Windy = False and Play = Yes then Humidity = Normal 4/6
    - If Humidity = Normal then Windy = False and Play = Yes 4/7
    - If Windy = False then Humidity = Normal and Play = Yes 4/8
    - If Play = Yes then Humidity = Normal and Windy = False 4/9
    - If True then Humidity = Normal and Windy = False and Play = Yes 4/12

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

  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)

### 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
  - Compute k-item set by merging (k-1)-item sets

### 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:
  
  \((A \ B \ C \ D) \text{ OK because of } (B \ C D)\)
  
  \((A \ C \ D \ E) \text{ Not OK because of } (C \ D \ E)\)

- 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 \( (2^c - 1) \)
- 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!

Association rules: discussion

- 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 the minimum support)

Other issues

- Standard ARFF format very inefficient for typical market basket data
  - Attributes represent items in a basket and most items are usually missing
  - Data should be represented in sparse format
  - 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: linear regression

- Work most naturally with numeric attributes
- Standard technique for numeric prediction
  - Outcome is linear combination of attributes
  \[ X = W_0 + W_1 a_1 + W_2 a_2 + \cdots + W_k a_k \]
  - Weights are calculated from the training data
  - Predicted value for first training instance \( a^{(1)} \)
  \[ W_0 a^{(1)}_0 + W_1 a^{(1)}_1 + W_2 a^{(1)}_2 + \cdots + W_k a^{(1)}_k = \sum_{j=0}^{k} W_j a^{(1)}_j \]
  (assuming each instance is extended with a constant attribute with value 1)

Minimizing the squared error

- Choose \( k + 1 \) coefficients to minimize the squared error on the training data
- Squared error:
  \[ \sum_{i=1}^{n} (X^{(i)} - \sum_{j=0}^{k} W_j a^{(i)}_j)^2 \]
- Derive coefficients using standard matrix operations
- Can be done if there are more instances than attributes (roughly speaking)
- Minimizing the 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**
- **Problem**: membership values are not in [0,1] range, so aren’t proper probability estimates

Linear models: logistic regression

- Builds a linear model for a transformed target variable
- **Assume** we have two classes
- **Logistic regression** replaces the target
  \[
  \Pr[1|a_1, a_2, \ldots, a_k]
  \]
  by this target
  \[
  \log\left(\frac{P[1|a_1, a_2, \ldots, a_k]}{1-P[1|a_1, a_2, \ldots, a_k]}\right)
  \]
- **Logit transformation** maps [0,1] to (-∞, +∞)

Logit transformation

- **Resulting model**:
  \[
  \Pr[1|a_1, a_2, \ldots, a_k] = \frac{1}{1 + e^{-w_0 - w_1 a_1 - \cdots - w_k a_k}}
  \]

Example logistic regression model

- Model with \(w_0 = 0.5\) and \(w_1 = 1\):

Maximum likelihood

- **Aim**: maximize probability of training data wrt parameters
- Can use logarithms of probabilities and maximize log-likelihood of model:
  \[
  \sum_{i=1}^{n} (1 - x^{(i)}) \log(1 - \Pr[1|a_1^{(i)}, a_2^{(i)}, \ldots, a_k^{(i)})] +
  x^{(i)} \log \Pr[1|a_1^{(i)}, a_2^{(i)}, \ldots, a_k^{(i)}]
  \]
- Where the \(x^{(i)}\) are either 0 or 1
- **Weights** \(w_j\) need to be chosen to maximize log-likelihood (relatively simple method: iteratively re-weighted least squares)

Multiple classes

- **Can perform** logistic regression independently for each class (like multi-response linear regression)
- **Problem**: probability estimates for different classes won’t sum to one
- **Better**: train coupled models by maximizing likelihood over all classes
- **Alternative** that often works well in practice: **pairwise classification**
Pairwise classification

- Idea: build model for each pair of classes, using only training data from those classes
- Problem? Have to solve $k(k-1)/2$ classification problems for $k$-class problem
- Turns out not to be a problem in many cases because training sets become small:
  - Assume data evenly distributed, i.e. $2n/k$ per learning problem for $n$ instances in total
  - Suppose learning algorithm is linear in $n$
  - Then runtime of pairwise classification is proportional to $(k(k-1)/2)(2n/k) = (k-1)n$

Linear models are hyperplanes

- Decision boundary for two-class logistic regression is where probability equals 0.5:
  $$Pr[1 | \mathbf{a}_1, a_2, \cdots, a_k] = \frac{1}{1 + e^{-w_0 a_0 - \cdots - w_k a_k}} = 0.5$$
  which occurs when $$-w_0 a_0 - \cdots - w_k a_k = 0$$
- Thus logistic regression can only separate data that can be separated by a hyperplane
- Multi-response has the same problem.
  Class 1 is assigned if:
  $$w_0^1 + w_1 a_1 + \cdots + w_k a_k > w_0^2 + w_1 a_1 + \cdots + w_k a_k$$
  $$\iff (w_0^1 - w_0^2) + (w_1^1 - w_1^2) a_1 + \cdots + (w_k^1 - w_k^2) a_k > 0$$

Linear models: The Perceptron

- Don’t actually need probability estimates if all we want to do is classification
- Different approach: learn separating hyperplane
- Assumption: data is linearly separable
- Algorithm for learning separating hyperplane: **perceptron learning rule**
- Hyperplane: $$0 = w_0 a_0 + w_1 a_1 + w_2 a_2 + \cdots + w_k a_k$$
  where we again assume that there is a constant attribute with value 1 (bias)
- If sum is greater than zero we predict the first class, otherwise the second class

Perceptron as a neural network

- Another mistake-driven algorithm for finding a separating hyperplane
  - Assumes binary data (i.e. attribute values are either zero or one)
- Difference: multiplicative updates instead of additive updates
  - Weights are multiplied by a user-specified parameter $\alpha > 1$ (or its inverse)
- Another difference: user-specified threshold parameter $\theta$
  - Predict first class if $$w_0 a_0 + w_1 a_1 + w_2 a_2 + \cdots + w_k a_k > \theta$$
The algorithm

while some instances are misclassified
for each instance \( a \) in the training data
classify \( a \) using the current weights
if the predicted class is incorrect
if \( a \) belongs to the first class
for each \( a \), that is \( i \), multiply \( w \) by alpha
(if \( a \) is \( i \), leave \( w \), unchanged)
otherwise
for each \( a \), that is \( i \), divide \( w \) by alpha
(if \( a \) is \( i \), leave \( w \), unchanged)

• Winnow is very effective in homing in on relevant features (it is attribute efficient)
• Can also be used in an on-line setting in which new instances arrive continuously
  (like the perceptron algorithm)

Balanced Winnow

• Winnow doesn’t allow negative weights and this can be a drawback in some applications
• Balanced Winnow maintains two weight vectors, one for each class:
while some instances are misclassified
for each instance \( a \) in the training data
classify \( a \) using the current weights
if the predicted class is incorrect
if \( a \) belongs to the first class
for each \( a \), that is \( i \), multiply \( w_i \) by alpha and divide \( w \) by alpha
(if \( a \) is \( i \), leave \( w_i \) and \( w \), unchanged)
otherwise
for each \( a \), that is \( i \), multiply \( w_i \) by alpha and divide \( w \) by alpha
(if \( a \) is \( i \), leave \( w_i \) and \( w \), unchanged)

• Instance is classified as belonging to the first class (of 2 classes) if:
  \[ w_i^1 - w_i^2 \geq \alpha_i \] and

Instance-based learning

See John (1961)

• Distance function defines what’s learned
• Most instance-based schemes use Euclidean distance:

\[ \sqrt{(a_1^1 - a_1^2)^2 + (a_2^1 - a_2^2)^2 + \cdots + (a_k^1 - a_k^2)^2} \]

\( a^{10} \) and \( a^{20} \): 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

See Flax and Hedges (1995)

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

\[ \bar{a}_i = \frac{v_i - \min v_i}{\max v_i - \min v_i} \]

\( v_i \): 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)

Finding nearest neighbors efficiently

See Flax and Hedges (1995)

• Simplest way of finding nearest neighbour:
  linear scan of the data
  Classification takes time proportional to the product of the number of instances in training and test sets
• Nearest-neighbor search can be done more efficiently using appropriate data structures
• We will discuss two methods that represent training data in a tree structure:
  \( kD \)-trees and ball trees

\( kD \)-tree example

See Flax and Hedges (1995)

Finding nearest neighbors efficiently

• Simplest way of finding nearest neighbour:
  linear scan of the data
  Classification takes time proportional to the product of the number of instances in training and test sets
• Nearest-neighbor search can be done more efficiently using appropriate data structures
• We will discuss two methods that represent training data in a tree structure:
  \( kD \)-trees and ball trees

\( kD \)-tree example

See Flax and Hedges (1995)
Using kD-trees: example

Building trees incrementally
- Big advantage of instance-based learning: classifier can be updated incrementally
  - Just add new training instance!
- Can we do the same with kD-trees?
- Heuristic strategy:
  - Find leaf node containing new instance
  - Place instance into leaf if leaf is empty
  - Otherwise, split leaf according to the longest dimension (to preserve squareness)
- Tree should be re-built occasionally (i.e. if depth grows to twice the optimum depth)

kD-tree example: insertion
- Consider \((7,4),(2,2),(6,7),(3,8)\)

More on kD-trees
- Complexity depends on depth of tree, given by logarithm of number of nodes
- Amount of backtracking required depends on quality of tree ("square" vs. "skinny" nodes)
- How to build a good tree? Need to find good split point and split direction
  - Split direction: direction with greatest variance
  - Split point: median value along that direction
- Using value closest to mean (rather than median) can be better if data is skewed
- Can apply this recursively
### kD-tree example: insertion

**kD-tree (Friedman et al., 1977)**

- Consider \((7,4),(2,2),(6,7),(3,8)\)
- Draw the kD-tree if the original split is on the X-axis (vertical)
- To create a well-balanced tree:
  - Find a good axis for split: Calculate variance of data points along each axis and select the axis with the greatest variance – create a splitting hyperplane perpendicular to this axis
  - Position the hyperplane by locating the median (or point closest to the mean, if the distribution of points is skewed) along this axis

### Ball trees

**Ball trees (Moore, 2000)**

- Problem in kD-trees: corners
- Observation: no need to make sure that regions don’t overlap
- Can use balls (hyperspheres) instead of hyperrectangles
  - A ball tree organizes the data into a tree of k-dimensional hyperspheres
  - Normally allows for a better fit to the data and thus more efficient search

### Using ball trees

- Nearest-neighbor search is done using the same backtracking strategy as in kD-trees
- Ball can be ruled out from consideration if: distance from target to ball’s center exceeds ball’s radius plus current upper bound

### Building ball trees

- Ball trees are built top down (like kD-trees)
- Don’t have to continue until leaf balls contain just two points: can enforce minimum occupancy (same in kD-trees)
- Basic problem: splitting a ball into two
- Simple (linear-time) split selection strategy:
  - Choose point farthest from ball’s center
  - Choose second point farthest from first one
  - Assign each point to these two points
  - Compute cluster centers and radii based on the two subsets to get two balls
**Discussion of nearest-neighbor learning**

- Often very accurate
- Assumes all attributes are equally important
- Remedy: attribute selection or weights
- Possible remedies against noisy instances:
  - Take 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
- kD-trees become inefficient when number of attributes is too large (approximately > 10)
- Ball trees (which are instances of metric trees) work well in higher-dimensional spaces

---

**More discussion**

- Instead of storing all training instances, compress them into regions
- Example: hyperpipes (from discussion of 1R)
- Another simple technique (Voting Feature Intervals):
  - Construct intervals for each attribute
  - Discretize numeric attributes
  - Treat each value of a nominal attribute as an "interval"
  - Count number of times class occurs in interval
  - Prediction is generated by letting intervals vote (those that contain the test instance)

---

**Clustering**

- Clustering techniques apply when there is no class to be predicted
- Aim: divide instances into "natural" groups
- As we've seen clusters can be:
  - disjoint vs. overlapping
  - deterministic vs. probabilistic
  - flat vs. hierarchical
- We'll look at a classic clustering algorithm called $k$-means
  - $k$-means clusters are disjoint, deterministic, and flat

---

**The $k$-means algorithm**

See Hartigan (1975)

To cluster data into $k$ groups:
($k$ is predefined)
1. Choose $k$ cluster centers
   - e.g. at random
2. Assign instances to clusters
   - based on distance to cluster centers
3. Compute centroids of clusters
4. Go to step 1
   - until convergence

---

**Discussion**

- Algorithm minimizes squared distance to cluster centers
- Result can vary significantly
  - based on initial choice of seeds
- Can get trapped in local minimum
  - Example:
  - To increase chance of finding global optimum: restart with different random seeds
- Can be applied recursively with $k = 2$

---

**Faster distance calculations**

See Moneo & Pfaller (2000)

- Can we use $k$D-trees or ball trees to speed up the process? Yes:
  - First, build tree, which remains static, for all the data points
  - At each node, store number of instances and sum of all instances
  - In each iteration, descend tree and find out which cluster each node belongs to
    - Can stop descending as soon as we find out that a node belongs entirely to a particular cluster
    - Use statistics stored at the nodes to compute new cluster centers
Example

Minsky and Papert (1969) showed that linear classifiers have limitations, e.g. can’t learn XOR
But: combinations of them can (→ multi-layer neural nets, which we’ll discuss later)

Comments on basic methods

• Bayes’ rule stems from his “Essay towards solving a problem in the doctrine of chances” (1763)
  • Difficult bit in general: estimating prior probabilities (easy in the case of naïve Bayes)
• Extension of naïve Bayes: Bayesian networks (which we’ll discuss later)
• Algorithm for association rules is called APRIORI (Agrawal et al., 1993)
  • Minsky and Papert (1969) showed that linear classifiers have limitations, e.g. can’t learn XOR
    But: combinations of them can (→ multi-layer neural nets, which we’ll discuss later)