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 few attributes can be captured by a decision tree
  - Use simple logical rules
  - A weighted linear combination might do
  - Instance-based: use a few prototypes
Algorithms: The basic methods

- 1R Algorithm
- Naïve Bayes Classifier
- Constructing decision trees
- PRISM method
- Mining association rules
- Linear models
- $k$-nearest neighbor algorithm
- Clustering: $k$-means method
4.1 1R algorithm
1R algorithm

- An easy way to find very simple classification rule
- 1R: rules that all test one particular attribute
- Basic version
  - One branch for each value
  - Each branch assigns most frequent class
  - Error rate: proportion of instances that don’t belong to the majority class of their corresponding branch
  - Choose attribute with lowest error rate (assumes nominal attributes)
- “Missing” is treated as a separate attribute value
Pseudo-code or 1R Algorithm

For each attribute,

For each value of that attribute, make a rule as follows:
  count how often each class appears
  find the most frequent class
  make the rule assign that class to this attribute-value.

Calculate the error rate of the rules.
Choose the rules with the smallest error rate.
Example: The weather problem

<table>
<thead>
<tr>
<th>Outlook</th>
<th>Temperature</th>
<th>Humidity</th>
<th>Windy</th>
<th>Play</th>
</tr>
</thead>
<tbody>
<tr>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>false</td>
<td>no</td>
</tr>
<tr>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>true</td>
<td>no</td>
</tr>
<tr>
<td>overcast</td>
<td>hot</td>
<td>high</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>rainy</td>
<td>mild</td>
<td>high</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>rainy</td>
<td>cool</td>
<td>normal</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>rainy</td>
<td>cool</td>
<td>normal</td>
<td>true</td>
<td>no</td>
</tr>
<tr>
<td>overcast</td>
<td>cool</td>
<td>normal</td>
<td>true</td>
<td>yes</td>
</tr>
<tr>
<td>sunny</td>
<td>mild</td>
<td>high</td>
<td>false</td>
<td>no</td>
</tr>
<tr>
<td>sunny</td>
<td>cool</td>
<td>normal</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>rainy</td>
<td>mild</td>
<td>normal</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>sunny</td>
<td>mild</td>
<td>normal</td>
<td>true</td>
<td>yes</td>
</tr>
<tr>
<td>overcast</td>
<td>mild</td>
<td>high</td>
<td>true</td>
<td>yes</td>
</tr>
<tr>
<td>overcast</td>
<td>hot</td>
<td>normal</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>rainy</td>
<td>mild</td>
<td>high</td>
<td>true</td>
<td>no</td>
</tr>
</tbody>
</table>
## Evaluating the weather attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Rules</th>
<th>Errors</th>
<th>Total errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 outlook</td>
<td>sunny → no</td>
<td>2/5</td>
<td>4/14</td>
</tr>
<tr>
<td></td>
<td>overcast → yes</td>
<td>0/4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>rainy → yes</td>
<td>2/5</td>
<td></td>
</tr>
<tr>
<td>2 temperature</td>
<td>hot → no*</td>
<td>2/4</td>
<td>5/14</td>
</tr>
<tr>
<td></td>
<td>mild → yes</td>
<td>2/6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>cool → yes</td>
<td>1/4</td>
<td></td>
</tr>
<tr>
<td>3 humidity</td>
<td>high → no</td>
<td>3/7</td>
<td>4/14</td>
</tr>
<tr>
<td></td>
<td>normal → yes</td>
<td>1/7</td>
<td></td>
</tr>
<tr>
<td>4 windy</td>
<td>false → yes</td>
<td>2/8</td>
<td>5/14</td>
</tr>
<tr>
<td></td>
<td>true → no*</td>
<td>3/6</td>
<td></td>
</tr>
</tbody>
</table>
The attribute with the smallest number of errors

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

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

<table>
<thead>
<tr>
<th>Outlook</th>
<th>Temperature</th>
<th>Humidity</th>
<th>Windy</th>
<th>Play</th>
</tr>
</thead>
<tbody>
<tr>
<td>sunny</td>
<td>85</td>
<td>85</td>
<td>false</td>
<td>no</td>
</tr>
<tr>
<td>sunny</td>
<td>80</td>
<td>90</td>
<td>true</td>
<td>no</td>
</tr>
<tr>
<td>overcast</td>
<td>83</td>
<td>86</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>rainy</td>
<td>70</td>
<td>96</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>rainy</td>
<td>68</td>
<td>80</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>rainy</td>
<td>65</td>
<td>70</td>
<td>true</td>
<td>no</td>
</tr>
<tr>
<td>overcast</td>
<td>64</td>
<td>65</td>
<td>true</td>
<td>yes</td>
</tr>
<tr>
<td>sunny</td>
<td>72</td>
<td>95</td>
<td>false</td>
<td>no</td>
</tr>
<tr>
<td>sunny</td>
<td>69</td>
<td>70</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>rainy</td>
<td>75</td>
<td>80</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>sunny</td>
<td>75</td>
<td>70</td>
<td>true</td>
<td>yes</td>
</tr>
<tr>
<td>overcast</td>
<td>72</td>
<td>90</td>
<td>true</td>
<td>yes</td>
</tr>
<tr>
<td>overcast</td>
<td>81</td>
<td>75</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>rainy</td>
<td>71</td>
<td>91</td>
<td>true</td>
<td>no</td>
</tr>
</tbody>
</table>
Example: *temperature* from weather data

<table>
<thead>
<tr>
<th>64</th>
<th>65</th>
<th>68</th>
<th>69</th>
<th>70</th>
<th>71</th>
<th>72</th>
<th>72</th>
<th>75</th>
<th>75</th>
<th>80</th>
<th>81</th>
<th>83</th>
<th>85</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>

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

  yes | no  | yes | yes | yes  | no  | no  | yes | yes | yes  | no  | yes | yes  | no
The problem of overfitting

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

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

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

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

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Rules</th>
<th>Errors</th>
<th>Total errors</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Outlook</strong></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><strong>Temperature</strong></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><strong>Humidity</strong></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><strong>Windy</strong></td>
<td>&gt; 95.5 → Yes</td>
<td>0/1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>False → Yes</td>
<td>2/8</td>
<td>5/14</td>
</tr>
<tr>
<td></td>
<td>True → No*</td>
<td>3/6</td>
<td></td>
</tr>
</tbody>
</table>
4.2 Naïve Bayes Classifier
Naïve Bayes Classifier

- “Opposite” of 1R: use all the attributes
- Two assumptions: Attributes are
  - equally important
  - statistically independent
    - i.e., knowing the value of one attribute says nothing about the value of another
- Equally important & independence assumptions are never correct in real-life datasets
Bayes Theorem

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

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

- $Pr[H]$: A priori probability of $H$
  - Probability of event before evidence is seen
- $Pr[H|E]$: posteriori probability of $H$
  - The probability of $H$ conditional on $E$
- $Pr[E|H]$: Posterior probability of $X$
- $Pr[E]$: A priori probability of $E$
Naïve Bayes for classification

- Classification learning: what’s the probability of the class given an instance?
  - Evidence \( E \) = instance
  - Event \( H \) = class value for instance

- Naïve assumption: evidence splits into parts (i.e. attributes) that are independent

\[
Pr[H/E] = \frac{Pr[E_1/H]Pr[E_2/H]...Pr[E_n/H]Pr[H]}{Pr[E]}
\]
Naïve Bayes classifier

- Hypothesis $H$ is the class.
- $Pr[E]$: can be ignored as it is constant for all classes.

$$Pr(H \mid E) = Pr(H) \prod_{k=1}^{n} Pr(E_k \mid H)$$

- $Pr(H)$ is the ratio of total samples in class $H$ to all samples.
Naïve Bayes classifier

- For Categorical attribute:
  - \( Pr(E_k/H) \) is the frequency of samples having value \( E_k \) in class \( H \).

- For Continuous (numeric) attribute:
  - \( Pr(E_k/H) \) is calculated via a Normal or Gaussian density function.
Having pre-calculated all $Pr(E_k|H)$ to classify an unknown sample $E$:

- Step 1: For all classes calculate $P(H|E)$.
- Step 2: Assign sample $E$ to the class with the highest $Pr(H|E)$. 
Naïve Bayes classifier

- E.g. \( \Pr(\text{outlook}=\text{sunny} \mid \text{play}=\text{yes}) = \frac{2}{9} \)
  \[ \Pr(\text{windy}=\text{true} \mid \text{play}=\text{No}) = \frac{3}{9} \]
Probabilities for weather data

- A new day:
  
<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>cool</td>
<td>high</td>
<td>true</td>
<td>?</td>
</tr>
</tbody>
</table>

  likelihood of *yes* = \( \frac{2}{9} \times \frac{3}{9} \times \frac{3}{9} \times \frac{3}{9} \times \frac{9}{14} = 0.0053 \).

  likelihood of *no* = \( \frac{3}{5} \times \frac{1}{5} \times \frac{4}{5} \times \frac{3}{5} \times \frac{5}{14} = 0.0206 \).

- Conversion into a probability by normalization:
  
  Probability of *yes* = \( \frac{0.0053}{0.0053 + 0.0206} = 20.5\% \),

  Probability of *no* = \( \frac{0.0206}{0.0053 + 0.0206} = 79.5\% \).
Bayes’s rule

- The hypothesis $H$ (class) is that $\text{play}$ will be ‘yes’ $\Pr[H|E]$ is 20.5%.
- The evidence $E$ is the particular combination of attribute values for the new day:
  - $\text{outlook} = \text{sunny}$
  - $\text{temperature} = \text{cool}$
  - $\text{humidity} = \text{high}$
  - $\text{windy} = \text{true}$
Weather data example

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

\[
\Pr [\text{yes} | E] = \frac{2}{9} \times \frac{3}{9} \times \frac{3}{9} \times \frac{3}{9} \times \frac{9}{14}
\]
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!
    
    \[ Pr[\text{Humidity}=\text{High} \mid \text{yes}]=0 \]
  
  - A posteriori probability will also be zero!
    \[ Pr[\text{yes} \mid E]=0 \]
  
  - (No matter how likely the other values are!)

- Correction: add 1 to the count for every attribute value-class combination (Laplace estimator)

- Result: probabilities will never be zero!
Modified probability estimates

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

\[
\begin{align*}
\text{sunny:} & \quad \frac{2 + \mu}{3} \quad \frac{4 + \mu}{3} \\
\text{overcast:} & \quad \frac{3 + \mu}{3} \\
\text{rainy:} & \quad \frac{3 + \mu}{3}
\end{align*}
\]

- Weights don’t need to be equal but they must sum to 1 (\(p_1, p_2, \text{ and } p_3\)) sum to 1.

\[
\begin{align*}
\text{sunny:} & \quad \frac{2 + \mu p_1}{9 + \mu} \\
\text{overcast:} & \quad \frac{4 + \mu p_2}{9 + \mu} \\
\text{rainy:} & \quad \frac{3 + \mu p_3}{9 + \mu}
\end{align*}
\]
Missing values

- Training: instance is not included in frequency count for attribute value-class combination
- Classification: attribute will be omitted from calculation
- Example: if the value of *outlook* were missing in the example

<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>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” = $\frac{1}{5} \times \frac{4}{5} \times \frac{3}{5} \times \frac{5}{14} = 0.0343$
- $P(\text{“yes”}) = \frac{0.0238}{(0.0238 + 0.0343)} = 41\%$
- $P(\text{“no”}) = \frac{0.0343}{(0.0238 + 0.0343)} = 59\%$
Numeric attributes

- Usual assumption: attributes have a *normal* or *Gaussian* probability distribution
- The *probability density function* for the normal distribution is defined by two parameters:
  - **Sample mean** $\mu$
  $$\mu = \frac{1}{n} \sum_{i=1}^{n} x_i$$
  - **Standard deviation** $\sigma$
  $$\sigma = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (x_i - \mu)^2}$$
  - Then the density function $f(x)$ is:
  $$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$
## 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></td>
<td>yes/ no</td>
<td>yes/ no</td>
<td>yes/ no</td>
<td>yes/ no</td>
</tr>
<tr>
<td>sunny</td>
<td>2/3</td>
<td>83/85</td>
<td>86/85</td>
<td>false/6/2/9/5</td>
</tr>
<tr>
<td>overcast</td>
<td>4/0</td>
<td>70/80</td>
<td>96/90</td>
<td>true/3/3</td>
</tr>
<tr>
<td>rainy</td>
<td>3/2</td>
<td>68/65</td>
<td>80/70</td>
<td></td>
</tr>
<tr>
<td></td>
<td>64/72</td>
<td>65/95</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>69/71</td>
<td>70/91</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>75</td>
<td>80</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>75</td>
<td>70</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>72</td>
<td>90</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>81</td>
<td>75</td>
<td></td>
<td></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></td>
<td>mean std. dev.</td>
<td>mean std. dev.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sunny</td>
<td>73/6.2</td>
<td>74.6/7.9</td>
<td>79.1/10.2</td>
<td>false/6/9/2/5/9/14/5/14</td>
</tr>
<tr>
<td>overcast</td>
<td>4/9</td>
<td>0/5</td>
<td>79.1/10.2</td>
<td>false/6/9/2/5/9/14/5/14</td>
</tr>
<tr>
<td>rainy</td>
<td>3/9</td>
<td>2/5</td>
<td>79.1/10.2</td>
<td>false/6/9/2/5/9/14/5/14</td>
</tr>
</tbody>
</table>
Example density value

- If we are considering a yes outcome when temperature has a value of 66
- We just need to plug $x = 66$, $\mu = 73$, and $\sigma = 6.2$ into the formula
- The value of the probability density function is:

$$f(temperature = 66 \mid yes) = \frac{1}{\sqrt{2\pi} \cdot 6.2} e^{-\frac{(66-73)^2}{2\cdot 6.2^2}} = 0.0340$$
### Classifying a new day

**A new day:**

<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>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* = \( \frac{3}{5} \times 0.0221 \times 0.0381 \times \frac{3}{5} \times \frac{5}{14} = 0.000108 \)

Probability of *yes* = \( \frac{0.000036}{0.000036 + 0.000108} = 25.0\% \)

Probability of *no* = \( \frac{0.000108}{0.000036 + 0.000108} = 75.0\% \)
Missing values

- Missing values during training are not included in calculation of mean and standard deviation.
4.3 Constructing decision trees
Constructing decision trees

- Strategy: top down
- Recursive *divide-and-conquer*
  - First: select attribute for root node
    Create branch for each possible attribute value
  - This splits instances into subsets
    One for each branch extending from the node
  - Then: repeat recursively for each branch, using only instances that reach the branch
- Stop if all instances at a node have the same class
Which attribute to select?
Chapter 4: Algorithms: The Basic Methods

Which attribute to select?
Criterion for attribute selection

- 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
  - It is measured in bits
- Strategy: choose attribute that gives greatest information gain
Criterion for attribute selection

- Nodes with **homogeneous** class distribution are preferred

- Need a measure of node impurity:

<table>
<thead>
<tr>
<th></th>
<th>C0: 5</th>
<th>C1: 5</th>
<th></th>
<th>C0: 9</th>
<th>C1: 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-homogeneous,</td>
<td></td>
<td></td>
<td>Homogeneous,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>High degree of impurity</td>
<td></td>
<td></td>
<td>Low degree of impurity</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
How to Find the Best Split

Before Splitting:

<table>
<thead>
<tr>
<th></th>
<th>C0</th>
<th>N00</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>N01</td>
<td></td>
</tr>
</tbody>
</table>

A?
- Yes: Node N1
- No: Node N2

Gain = M0 – M12 vs M0 – M34

B?
- Yes: Node N3
- No: Node N4
Computing information

- 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:

\[
\text{entropy}(p_1, p_2, \ldots, p_n) = -p_1 \log p_1 - p_2 \log p_2 \ldots - p_n \log p_n
\]

- “High Entropy” means X is from a uniform (boring) distribution.
- “Low Entropy” means X is from a varied (peaks and valleys) distribution.
Example: attribute \textit{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:**
  \[
  \text{info}([2,3]) = \text{entropy}(3/5,2/5) = -3/5 \log(3/5) - 2/5 \log(2/5) = 0.971 \text{bits}
  \]

- **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 information gain

- Information gain: information before splitting – information after splitting:

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

- Information gain for attributes from weather data:

  \[
  \begin{align*}
  \text{gain}(\text{Outlook }) &= 0.247 \text{ bits} \\
  \text{gain}(\text{Temperature }) &= 0.029 \text{ bits} \\
  \text{gain}(\text{Humidity }) &= 0.152 \text{ bits} \\
  \text{gain}(\text{Windy }) &= 0.048 \text{ bits}
  \end{align*}
  \]
Continuing to split
gain(temperature) = 0.571 bits
gain(humidity) = 0.971 bits
gain(windy) = 0.020 bits
Final decision tree

- Splitting stops when data can’t be split any further
Wish list 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
Highly-branching attributes

- Problem: 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 selection of an attribute that is non-optimal for prediction
- Another problem: fragmentation
<table>
<thead>
<tr>
<th>ID code</th>
<th>Outlook</th>
<th>Temperature</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 ‘*ID Code*’:
  \[
  \text{info}([0,1]) + \text{info}([0,1]) + \text{info}([1,0]) + \ldots + \text{info}([1,0]) + \text{info}([0,1])
  \]

- Information gain is maximal for ID code (namely 0.940 bits)
Gain ratio

- **Gain ratio**: a modification of the information gain
- 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
Computing the gain ratio

- Example: intrinsic information for *Outlook* split:
  \[ \text{info}([5, 4, 5]) = 1.577 \]

- Value of attribute decreases as intrinsic information gets larger

- Gain ratio attribute = gain attribute / intrinsic info attribute

- Gain ratio ID code = 0.247 bits / 1.577 bits = 1.157
## Gain ratios for weather data

<table>
<thead>
<tr>
<th>Outlook</th>
<th>Temperature</th>
<th>Humidity</th>
<th>Windy</th>
</tr>
</thead>
<tbody>
<tr>
<td>info:</td>
<td>0.693</td>
<td>0.911</td>
<td>0.788</td>
</tr>
<tr>
<td>gain: 0.940–0.693</td>
<td>0.247</td>
<td>0.029</td>
<td>0.152</td>
</tr>
<tr>
<td></td>
<td></td>
<td>gain: 0.940–0.788</td>
<td>gain: 0.940–0.892</td>
</tr>
<tr>
<td>split info:</td>
<td>1.577</td>
<td>1.557</td>
<td>1.000</td>
</tr>
<tr>
<td>info([5,4,5])</td>
<td></td>
<td>split info:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>info([4,6,4])</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>split info:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>info ([7,7])</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gain ratio:</td>
<td>0.157</td>
<td>0.019</td>
<td>0.152</td>
</tr>
<tr>
<td>0.247/1.577</td>
<td></td>
<td>0.029/1.557</td>
<td>0.152/1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4.4 PRISM method
Covering algorithms

- Convert decision tree into a rule set
  - Straightforward, but rule set very complex
- 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

- Possible rule set for class “a”:
  
  \( \text{if true then class = a} \)
Example: generating a rule

- Possible rule set for class “a”:

  If $x > 1.2$ then class = a
Example: generating a rule

- Possible rule set for class “a”:

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

- Corresponding decision tree: (produces exactly the same predictions)
Rules vs. trees

- Both methods might first split the dataset using the $x$ attribute and would probably end up splitting it at the same place ($x = 1.2$)
- But: rule sets *can* be more clear when decision trees suffer from replicated subtrees
- Also: in multiclass situations, covering algorithm concentrates on one class at a time whereas decision tree learner takes all classes into account
A simple covering algorithm

- It is called **PRISM method** for constructing rules
- Generates a rule by adding tests that maximize rule’s accuracy
- Divide-and-conquer algorithms choose an attribute to maximize the information gain
- But: the covering algorithm chooses an attribute–value pair to maximize the probability of the desired classification
A simple covering algorithm

- Each new test reduces rule's coverage:

![Diagram showing the space of examples, rule so far, and rule after adding a new term]
Selecting a test

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

- We are finished when \( p/t = 1 \) or the set of instances can’t be split any further
### Example: contact lens data

<table>
<thead>
<tr>
<th>Age</th>
<th>Spectacle Prescription</th>
<th>Astigmatism</th>
<th>Tear Production Rate</th>
<th>Recommended Lenses</th>
</tr>
</thead>
<tbody>
<tr>
<td>young</td>
<td>myope</td>
<td>no</td>
<td>reduced</td>
<td>none</td>
</tr>
<tr>
<td>young</td>
<td>myope</td>
<td>no</td>
<td>normal</td>
<td>soft</td>
</tr>
<tr>
<td>young</td>
<td>myope</td>
<td>yes</td>
<td>reduced</td>
<td>none</td>
</tr>
<tr>
<td>young</td>
<td>hypermetrope</td>
<td>no</td>
<td>reduced</td>
<td>none</td>
</tr>
<tr>
<td>young</td>
<td>hypermetrope</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>soft</td>
</tr>
<tr>
<td>pre-presbyopic</td>
<td>myope</td>
<td>no</td>
<td>reduced</td>
<td>none</td>
</tr>
<tr>
<td>pre-presbyopic</td>
<td>myope</td>
<td>no</td>
<td>normal</td>
<td>soft</td>
</tr>
<tr>
<td>pre-presbyopic</td>
<td>myope</td>
<td>yes</td>
<td>reduced</td>
<td>none</td>
</tr>
<tr>
<td>pre-presbyopic</td>
<td>myope</td>
<td>yes</td>
<td>normal</td>
<td>hard</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>no</td>
<td>normal</td>
<td>soft</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>no</td>
<td>reduced</td>
<td>none</td>
</tr>
<tr>
<td>presbyopic</td>
<td>myope</td>
<td>no</td>
<td>normal</td>
<td>none</td>
</tr>
<tr>
<td>presbyopic</td>
<td>myope</td>
<td>yes</td>
<td>reduced</td>
<td>none</td>
</tr>
<tr>
<td>presbyopic</td>
<td>myope</td>
<td>yes</td>
<td>normal</td>
<td>hard</td>
</tr>
<tr>
<td>presbyopic</td>
<td>hypermetrope</td>
<td>no</td>
<td>reduced</td>
<td>none</td>
</tr>
<tr>
<td>presbyopic</td>
<td>hypermetrope</td>
<td>no</td>
<td>normal</td>
<td>soft</td>
</tr>
<tr>
<td>presbyopic</td>
<td>hypermetrope</td>
<td>yes</td>
<td>reduced</td>
<td>none</td>
</tr>
<tr>
<td>presbyopic</td>
<td>hypermetrope</td>
<td>yes</td>
<td>normal</td>
<td>none</td>
</tr>
</tbody>
</table>
Example: contact lens data

To begin, we seek a rule:

If ? then recommendation = hard

Possible tests:

- age = young 2/8
- age = pre-presbyopic 1/8
- age = presbyopic 1/8
- spectacle prescription = myope 3/12
- spectacle prescription = hypermetrope 1/12
- astigmatism = no 0/12
- astigmatism = yes 4/12
- tear production rate = reduced 0/12
- tear production rate = normal 4/12
Create the rule

- Rule with best test added and covered instances:

If astigmatism = yes then recommendation = hard

<table>
<thead>
<tr>
<th>Age</th>
<th>Spectacle prescription</th>
<th>Astigmatism</th>
<th>Tear production rate</th>
<th>Recommended lenses</th>
</tr>
</thead>
<tbody>
<tr>
<td>young</td>
<td>myope</td>
<td>yes</td>
<td>reduced</td>
<td>none</td>
</tr>
<tr>
<td>young</td>
<td>myope</td>
<td>yes</td>
<td>normal</td>
<td>hard</td>
</tr>
<tr>
<td>young</td>
<td>hypermetrope</td>
<td>yes</td>
<td>reduced</td>
<td>none</td>
</tr>
<tr>
<td>young</td>
<td>hypermetrope</td>
<td>yes</td>
<td>normal</td>
<td>hard</td>
</tr>
<tr>
<td>pre-presbyopic</td>
<td>myope</td>
<td>yes</td>
<td>reduced</td>
<td>none</td>
</tr>
<tr>
<td>pre-presbyopic</td>
<td>myope</td>
<td>yes</td>
<td>normal</td>
<td>hard</td>
</tr>
<tr>
<td>pre-presbyopic</td>
<td>hypermetrope</td>
<td>yes</td>
<td>reduced</td>
<td>none</td>
</tr>
<tr>
<td>pre-presbyopic</td>
<td>hypermetrope</td>
<td>yes</td>
<td>normal</td>
<td>none</td>
</tr>
<tr>
<td>presbyopic</td>
<td>myope</td>
<td>yes</td>
<td>reduced</td>
<td>none</td>
</tr>
<tr>
<td>presbyopic</td>
<td>myope</td>
<td>yes</td>
<td>normal</td>
<td>hard</td>
</tr>
<tr>
<td>presbyopic</td>
<td>hypermetrope</td>
<td>yes</td>
<td>reduced</td>
<td>none</td>
</tr>
<tr>
<td>presbyopic</td>
<td>hypermetrope</td>
<td>yes</td>
<td>normal</td>
<td>none</td>
</tr>
</tbody>
</table>
Further refinement

- **Current state:**
  
  If astigmatism = yes and ? then recommendation = hard

- **Possible tests:**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>age = young</td>
<td>2/4</td>
</tr>
<tr>
<td>age = pre-presbyopic</td>
<td>1/4</td>
</tr>
<tr>
<td>age = presbyopic</td>
<td>1/4</td>
</tr>
<tr>
<td>spectacle prescription = myope</td>
<td>3/6</td>
</tr>
<tr>
<td>spectacle prescription = hypermetrope</td>
<td>1/6</td>
</tr>
<tr>
<td>tear production rate = reduced</td>
<td>0/6</td>
</tr>
<tr>
<td>tear production rate = normal</td>
<td>4/6</td>
</tr>
</tbody>
</table>
Modified rule and resulting data

- **Rule with best test added:**

  If astigmatism = yes and tear production rate = normal
  then recommendation = hard

- **Instances covered by modified rule:**

<table>
<thead>
<tr>
<th>Age</th>
<th>Spectacle prescription</th>
<th>Astigmatism</th>
<th>Tear production rate</th>
<th>Recommended lenses</th>
</tr>
</thead>
<tbody>
<tr>
<td>young</td>
<td>myope</td>
<td>yes</td>
<td>normal</td>
<td>hard</td>
</tr>
<tr>
<td>young</td>
<td>hypermetrope</td>
<td>yes</td>
<td>normal</td>
<td>hard</td>
</tr>
<tr>
<td>pre-presbyopic</td>
<td>myope</td>
<td>yes</td>
<td>normal</td>
<td>hard</td>
</tr>
<tr>
<td>pre-presbyopic</td>
<td>hypermetrope</td>
<td>yes</td>
<td>normal</td>
<td>none</td>
</tr>
<tr>
<td>presbyopic</td>
<td>myope</td>
<td>yes</td>
<td>normal</td>
<td>hard</td>
</tr>
<tr>
<td>presbyopic</td>
<td>hypermetrope</td>
<td>yes</td>
<td>normal</td>
<td>none</td>
</tr>
</tbody>
</table>
Further refinement

- **Current state:**

  If astigmatism = yes and tear production rate = normal and ? then recommendation = hard

- **Possible tests:**

  age = young 2/2
  age = pre-presbyopic 1/2
  age = presbyopic 1/2
  spectacle prescription = myope 3/3
  spectacle prescription = hypermetrope 1/3

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

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

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

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

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

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

- Methods like PRISM (for dealing with one class) are *separate-and-conquer* algorithms:
  - First, identify a useful rule
  - Then, separate out all the instances it covers
  - Finally, “conquer” the remaining instances

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

- Given a set of transactions, find rules that will predict the occurrence of an item based on the occurrences of other items in the transaction

- Broad applications
  - Basket data analysis, cross-marketing, catalog design, sale campaign analysis
  - Web log (click stream) analysis, DNA sequence analysis, etc.
Market basket analysis

Which items are frequently purchased together by my customers?

Shopping Baskets

Customer 1: milk, bread, cereal
Customer 2: milk, bread, sugar, eggs
Customer 3: milk, bread, butter
Customer n: sugar, eggs

Market Analyst
Market basket analysis

- Market-Basket transactions

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bread, Milk</td>
</tr>
<tr>
<td>2</td>
<td>Bread, Diaper, Beer, Eggs</td>
</tr>
<tr>
<td>3</td>
<td>Milk, Diaper, Beer, Coke</td>
</tr>
<tr>
<td>4</td>
<td>Bread, Milk, Diaper, Beer</td>
</tr>
<tr>
<td>5</td>
<td>Bread, Milk, Diaper, Coke</td>
</tr>
</tbody>
</table>

- Example of Association Rules

{Diaper} → {Beer},
{Milk, Bread} → {Eggs, Coke},
{Beer, Bread} → {Milk},
Definitions: Item set

- **Item**: one test/attribute-value pair (e.g. Milk, Bread)
- **Item set**: A collection of one or more items (e.g. \{Milk, Bread, Diaper\})
- **k-itemset**: An itemset that contains k items
- **Support count**: Frequency of occurrence of an itemset
- **Frequent Itemset**: An itemset whose support count is greater than or equal to a minsup


Definition: Association Rule

- **Association Rule**
  - An implication expression of the form $X \rightarrow Y$, where $X$ and $Y$ are itemsets
  - Example: \{Milk, Diaper\} $\rightarrow$ \{Beer\}

- **Rule Evaluation Metrics**
  - Support ($s$): Fraction of transactions that contain both $X$ and $Y$
  - Confidence ($c$): Measures how often items in $Y$ appear in transactions that contain $X$

\[
\text{support}(A \Rightarrow B) = P(A \cup B) \\
\text{confidence}(A \Rightarrow B) = P(B|A)
\]

\[
\text{confidence}(A \Rightarrow B) = P(B|A) = \frac{\text{support}(A \cup B)}{\text{support}(A)} = \frac{\text{support}_\text{count}(A \cup B)}{\text{support}_\text{count}(A)}
\]
Definition: Association Rule

- Example:

\[ \{\text{Milk, Diaper}\} \Rightarrow \text{Beer} \]

\[ s = \frac{2}{5} = 0.4 \]

\[ c = \frac{2}{3} = 0.67 \]

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bread, Milk</td>
</tr>
<tr>
<td>2</td>
<td>Bread, Diaper, Beer, Eggs</td>
</tr>
<tr>
<td>3</td>
<td>Milk, Diaper, Beer, Coke</td>
</tr>
<tr>
<td>4</td>
<td>Bread, Milk, Diaper, Beer</td>
</tr>
<tr>
<td>5</td>
<td>Bread, Milk, Diaper, Coke</td>
</tr>
</tbody>
</table>
Association Rules

- Itemset $X = \{x_1, \ldots, x_k\}$
- Find all the rules $X \Rightarrow Y$ with min confidence and support
  - Support, $s$, probability that a transaction contains $X \cup Y$
  - Confidence, $c$, conditional probability that a transaction having $X$ also contains $Y$.

\[
\begin{align*}
\text{Customer buys both} & \quad \text{Customer buys diaper} \\
\text{Customer buys beer} & \quad \{\text{Diaper}\} \Rightarrow \text{Beer}
\end{align*}
\]
Example

Let \( \text{min\_support} = 50\% \), \( \text{min\_conf} = 50\% \):
- \( A \rightarrow C \) (50\%, 66.7\%)
- \( C \rightarrow A \) (50\%, 100\%)

<table>
<thead>
<tr>
<th>Transaction-id</th>
<th>Items bought</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>A, B, C</td>
</tr>
<tr>
<td>20</td>
<td>A, C</td>
</tr>
<tr>
<td>30</td>
<td>A, D</td>
</tr>
<tr>
<td>40</td>
<td>B, E, F</td>
</tr>
</tbody>
</table>
Association Rule Mining Task

- Given a set of transactions T, the goal of association rule mining is to find all rules having
  - support ≥ \textit{minsup} threshold
  - confidence ≥ \textit{minconf} threshold

- Brute-force approach:
  - List all possible association rules
  - Compute the support and confidence for each rule
  - Prune rules that fail the \textit{minsup} and \textit{minconf} thresholds

⇒ Problem: Computational complexity!
Frequent Itemset Generation

Given \( d \) items, there are \( 2^d \) possible candidate itemsets.
Given $d$ unique items:

- Total number of itemsets = $2^d$
- Total number of possible association rules:

$$R = \sum_{k=1}^{d-1} \binom{d}{k} \times \sum_{j=1}^{d-k} \binom{d-k}{j}$$

$$= 3^d - 2^{d+1} + 1$$

If $d=6$, $R = 602$ rules
Apriori Algorithm

- Let \( k = 1 \)
- Generate frequent itemsets of length 1
- Repeat until no new frequent itemsets are identified
  - Generate length \((k+1)\) candidate itemsets from length \(k\) frequent itemsets
  - Prune candidate itemsets containing subsets of length \(k\) that are infrequent
  - Count the support of each candidate by scanning the dataset
  - Eliminate candidates that are infrequent, leaving only those that are frequent
The Apriori Algorithm—An Example

Database TDB

<table>
<thead>
<tr>
<th>Tid</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>A, C, D</td>
</tr>
<tr>
<td>20</td>
<td>B, C, E</td>
</tr>
<tr>
<td>30</td>
<td>A, B, C, E</td>
</tr>
<tr>
<td>40</td>
<td>B, E</td>
</tr>
</tbody>
</table>

Sup_{min} = 2

1^{st} scan

C_1

\begin{array}{|c|c|}
\hline
\text{Itemset} & \text{sup} \\
\hline
\{A\} & 2 \\
\{B\} & 3 \\
\{C\} & 3 \\
\{D\} & 1 \\
\{E\} & 3 \\
\hline
\end{array}

L_1

\begin{array}{|c|c|}
\hline
\text{Itemset} & \text{sup} \\
\hline
\{A\} & 2 \\
\{B\} & 3 \\
\{C\} & 3 \\
\{E\} & 3 \\
\hline
\end{array}

2^{nd} scan

C_2

\begin{array}{|c|c|}
\hline
\text{Itemset} & \text{sup} \\
\hline
\{A, B\} & 1 \\
\{A, C\} & 2 \\
\{A, E\} & 1 \\
\{B, C\} & 2 \\
\{B, E\} & 3 \\
\{C, E\} & 2 \\
\hline
\end{array}

L_2

\begin{array}{|c|c|}
\hline
\text{Itemset} & \text{sup} \\
\hline
\{A, B\} & 1 \\
\{A, C\} & 2 \\
\{A, E\} & 1 \\
\{B, C\} & 2 \\
\{B, E\} & 3 \\
\{C, E\} & 2 \\
\hline
\end{array}

3^{rd} scan

C_3

\begin{array}{|c|c|}
\hline
\text{Itemset} & \\
\hline
\{B, C, E\} & \\
\hline
\end{array}

L_3

\begin{array}{|c|c|}
\hline
\text{Itemset} & \text{sup} \\
\hline
\{B, C, E\} & 2 \\
\hline
\end{array}
The Apriori Algorithm

- **Pseudo-code:**
  
  \[ C_k: \text{Candidate itemset of size } k \]
  \[ L_k: \text{frequent itemset of size } k \]

  \[ L_1 = \{\text{frequent items}\}; \]

  \[ \text{for } (k = 1; L_k \neq \emptyset; k++) \text{ do begin} \]
  \[ C_{k+1} = \text{candidates generated from } L_k; \]
  \[ \text{for each transaction } t \text{ in database do} \]
  \[ \text{increment the count of all candidates in } C_{k+1} \]
  \[ \text{that are contained in } t \]
  \[ L_{k+1} = \text{candidates in } C_{k+1} \text{ with min\_support} \]
  \[ \text{end} \]

  \[ \text{return } \bigcup_k L_k; \]
Important Details of Apriori

- How to generate candidates?
  - Step 1: self-joining $L_k$
  - Step 2: pruning
- How to count supports of candidates?
- Example of Candidate-generation
  - $L_3=\{abc, abd, acd, ace, bcd\}$
  - Self-joining: $L_3 \times L_3$
    - $abcd$ from $abc$ and $abd$
    - $acde$ from $acd$ and $ace$
  - Pruning:
    - $acde$ is removed because $ade$ is not in $L_3$
  - $C_4=\{abcd\}$
## 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>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>
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 temperature = mild (2)</td>
<td>outlook = sunny temperature = hot humidity = high (2)</td>
<td>outlook = sunny temperature = hot humidity = high play = no (2)</td>
</tr>
<tr>
<td>outlook = overcast (4)</td>
<td>outlook = sunny temperature = hot (2)</td>
<td>outlook = sunny temperature = hot play = no (2)</td>
<td>outlook = sunny temperature = hot windy = false play = no (2)</td>
</tr>
<tr>
<td>outlook = rainy (5)</td>
<td>outlook = sunny humidity = normal (2)</td>
<td>outlook = sunny humidity = normal play = yes (2)</td>
<td>outlook = overcast temperature = hot windy = false play = yes (2)</td>
</tr>
<tr>
<td>.....</td>
<td>.....</td>
<td>.....</td>
<td>.....</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)
Once all item sets with minimum support have been generated, we can turn them into rules:

humidity = normal, windy = false, play = yes

Seven potential rules:

1. If humidity = normal and windy = false then play = yes
2. If humidity = normal and play = yes then windy = false
3. If windy = false and play = yes then humidity = normal
4. If humidity = normal then windy = false and play = yes
5. If windy = false then humidity = normal and play = yes
6. If play = yes then humidity = normal and windy = false
7. If - then humidity = normal and windy = false and play = yes
Rules for weather data

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

<table>
<thead>
<tr>
<th>Association rule</th>
<th>Coverage</th>
<th>Accuracy</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 = cool play = yes ⇒ humidity = normal</td>
<td>3</td>
<td>100%</td>
</tr>
<tr>
<td>5 outlook = rainy windy = false ⇒ play = yes</td>
<td>3</td>
<td>100%</td>
</tr>
<tr>
<td>6 outlook = rainy play = yes ⇒ windy = false</td>
<td>3</td>
<td>100%</td>
</tr>
<tr>
<td>7 outlook = sunny humidity = high ⇒ play = no</td>
<td>3</td>
<td>100%</td>
</tr>
<tr>
<td>8 outlook = sunny play = no ⇒ humidity = high</td>
<td>3</td>
<td>100%</td>
</tr>
<tr>
<td>9 temperature = cool windy = false ⇒ humidity = normal ⇒ play = yes</td>
<td>2</td>
<td>100%</td>
</tr>
</tbody>
</table>

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

- **Item set:**

  \[
  \text{temperature} = \text{cool}, \ \text{humidity} = \text{normal}, \ \text{windy} = \text{false}, \ \text{play} = \text{yes}
  \]

- **Resulting rules (all with 100% confidence):**

  \[
  \begin{align*}
  \text{temperature} = \text{cool} \ \text{windy} = \text{false} & \implies \text{humidity} = \text{normal} \\
  \text{temperature} = \text{cool} \ \text{humidity} = \text{normal} \ \text{windy} = \text{false} & \implies \text{play} = \text{yes} \\
  \text{temperature} = \text{cool} \ \text{windy} = \text{false} \ \text{play} = \text{yes} & \implies \text{humidity} = \text{normal}
  \end{align*}
  \]

- **Three subsets of this item set also have coverage 2:**

  \[
  \begin{align*}
  \text{temperature} = \text{cool}, \ \text{windy} = \text{false} \\
  \text{temperature} = \text{cool}, \ \text{humidity} = \text{normal}, \ \text{windy} = \text{false} \\
  \text{temperature} = \text{cool}, \ \text{windy} = \text{false}, \ \text{play} = \text{yes}
  \end{align*}
  \]
Generating rules efficiently

- We are looking for all high-confidence rules
  - But: rough 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 humidity = high and windy = false and play = no then outlook = sunny
If outlook = sunny and windy = false and play = no then humidity = high

● Corresponding 2 consequent rule:

If windy = false and play = no then outlook = sunny
and humidity = high
4.6 Linear models
Linear regression

- Work most naturally with numeric attributes
- Standard technique for numeric prediction
- **Linear regression**: Data are modeled to fit a straight line
- Linear regression involves a response variable $y$ and a single predictor variable $x$

  $$y = w_0 + w_1 x$$

  - where $w_0$ (y-intercept) and $w_1$ (slope) are regression coefficients
  - Two regression coefficients, $w$ and $b$, specify the line
Linear regression

- **Method of least squares**: estimates the best-fitting straight line

\[ w_1 = \frac{\sum_{i=1}^{|D|} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{|D|} (x_i - \bar{x})^2} \]

\[ w_0 = \bar{y} - w_1 \bar{x} \]

- \( D \): a training set consisting of values of predictor variable
- \(|D|\) data points of the form \((x_1, y_1), (x_2, y_2), \ldots, (x_{|D|}, y_{|D|})\).
- where \( x \) is the mean value of \( x_1, x_2, \ldots, x_{|D|} \), and \( y \) is the mean value of \( y_1, y_2, \ldots, y_{|D|} \).
Example: Salary data

<table>
<thead>
<tr>
<th>$x$ years experience</th>
<th>$y$ salary (in $1000s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>8</td>
<td>57</td>
</tr>
<tr>
<td>9</td>
<td>64</td>
</tr>
<tr>
<td>13</td>
<td>72</td>
</tr>
<tr>
<td>3</td>
<td>36</td>
</tr>
<tr>
<td>6</td>
<td>43</td>
</tr>
<tr>
<td>11</td>
<td>59</td>
</tr>
<tr>
<td>21</td>
<td>90</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>16</td>
<td>83</td>
</tr>
</tbody>
</table>

$\bar{x} = 9.1$ and $\bar{y} = 55.4$

\[ w_1 = \frac{(3 - 9.1)(30 - 55.4) + (8 - 9.1)(57 - 55.4) + \cdots + (16 - 9.1)(83 - 55.4)}{(3 - 9.1)^2 + (8 - 9.1)^2 + \cdots + (16 - 9.1)^2} = 3.5 \]

\[ w_0 = 55.4 - (3.5)(9.1) = 23.6 \]

$y = 23.6 + 3.5x$
Example: Salary data

Linear Regression: $Y = 3.5X + 23.2$

![Graph showing linear regression with equation $Y = 3.5X + 23.2$.]
Multiple linear regression

- Multiple linear regression involves more than one predictor variable.
- Training data is of the form \((X_1, y_1), (X_2, y_2), \ldots, (X_{|D|}, y_{|D|})\)
- Where the \(X_i\) are the \(n\)-dimensional training data with associated class labels, \(y_i\).
- An example of a multiple linear regression model based on two predictor attributes:

\[ y = w_0 + w_1x_1 + w_2x_2 \]
### Linear Regression: CPU performance data

<table>
<thead>
<tr>
<th>Cycle time (ns)</th>
<th>Main memory (KB)</th>
<th>Cache (KB)</th>
<th>Channels</th>
<th>Performance PRP</th>
</tr>
</thead>
<tbody>
<tr>
<td>MYCT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>125</td>
<td>256</td>
<td>6000</td>
<td>256</td>
</tr>
<tr>
<td>2</td>
<td>29</td>
<td>8000</td>
<td>32000</td>
<td>32</td>
</tr>
<tr>
<td>3</td>
<td>29</td>
<td>8000</td>
<td>32000</td>
<td>32</td>
</tr>
<tr>
<td>4</td>
<td>29</td>
<td>8000</td>
<td>32000</td>
<td>32</td>
</tr>
<tr>
<td>5</td>
<td>29</td>
<td>8000</td>
<td>16000</td>
<td>32</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>207</td>
<td>125</td>
<td>2000</td>
<td>8000</td>
<td>0</td>
</tr>
<tr>
<td>208</td>
<td>480</td>
<td>512</td>
<td>8000</td>
<td>32</td>
</tr>
<tr>
<td>209</td>
<td>480</td>
<td>1000</td>
<td>4000</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
PRP = -55.9 + 0.0489 \text{MYCT} + 0.0153 \text{MMIN} + 0.0056 \text{MMAX} + 0.6410 \text{CACH} - 0.2700 \text{CHMIN} + 1.480 \text{CHMAX}.
\]
4.7 k-nearest neighbor algorithm
Example Problem: Face Recognition

- We have a database of (say) 1 million face images
- We are given a new image and want to find the most similar images in the database
- Represent faces by (relatively) invariant values, e.g., ratio of nose width to eye width
- Each image represented by a large number of numerical features

**Problem:** given the features of a new face, find those in the DB that are close in at least $\frac{3}{4}$ (say) of the features
k-nearest neighbor algorithm

- *k*-Nearest neighbor is an example of *instance-based learning*
- Distance function defines what’s learned
- A classification for a new unclassified record may be found simply by comparing it to the most similar records in the training set
- **Example:**
  - We are interested in classifying the type of drug a patient should be prescribed
  - Based on the age of the patient and the patient’s sodium/potassium ratio (Na/K)
  - Dataset includes 200 patients
On the scatter plot; light gray points indicate drug Y; medium gray points indicate drug A or X; dark gray points indicate drug B or C.
Close-up of neighbors to new patient 2

- $k=1 \Rightarrow$ drugs B and C (dark gray)
- $k=2 \Rightarrow$ ?
- $K=3 \Rightarrow$ drugs A and X (medium gray)

Main questions:
- How many neighbors should we consider? That is, what is $k$?
- How do we measure distance?
- Should all points be weighted equally, or should some points have more influence than others?
Instance-based learning

- 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: Manhattan or *city-block metric*
  - Taking absolute differences value without squaring them
Normalization and other issues

- Different attributes are measured on different scales, 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 \( i \)
- All attribute values lie between 0 and 1
- Nominal attributes: distance either 0 or 1
- Common policy for missing values: assumed to be maximally distant (given normalized attributes)
Finding nearest neighbors efficiently

- Simplest way of finding nearest neighbor: 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
- There are two methods that represent training data in a tree structure:
  - $kD$-trees ($k$-dimensional trees)
  - Ball trees
$kD$-tree example
Using $k$D-trees: example
More on \( kD \)-trees

- Complexity depends on depth of tree, given by base 2 logarithm of number of nodes.
- Amount of backtracking required depends on quality of tree.
- 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 or value closest to mean along that direction.
- Can apply this recursively.
Building trees incrementally

- Big advantage of instance-based learning: classifier can be updated incrementally
  - Just add new training instance!
- We can do the same with $k$D-trees
- Heuristic strategy:
  - Find leaf node containing new instance
  - Place instance into leaf if leaf is empty
  - Otherwise, split leaf
- Tree should be rebuilt occasionally
Ball trees

- Problem in $k$D-trees: corners
- Can use balls (hyperspheres) instead of hyperrectangles
  - no need to make sure that regions don't overlap
  - 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
Ball tree for 16 training instances
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 minimum radius based on the two subsets to get two balls
4.8 Clustering: k-means method
Example: Clustering Documents

- Represent a document by a vector \((x_1, x_2, \ldots, x_k)\), where \(x_i = 1\) if the \(i\)th word (in some order) appears in the document.

- Documents with similar sets of words may be about the same topic.
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
Examples of Clustering Applications

- **Marketing**: Help marketers discover distinct groups in their customer bases, and then use this knowledge to develop targeted marketing programs.
- **Land use**: Identification of areas of similar land use in an earth observation database.
- **Insurance**: Identifying groups of motor insurance policy holders with a high average claim cost.
- **City-planning**: Identifying groups of houses according to their house type, value, and geographical location.
- **Documenting**: with similar sets of words may be about the same topic.
The $k$-means algorithm

To cluster data into $k$ groups:

($k$ is predefined)

1. Choose $k$ cluster centers
   - e.g. first time at random, then mean point
2. Assign instances to clusters
   - based on distance to cluster centers with the nearest point
3. Compute *centroids or mean* of clusters and they are taken to be new center values
4. Go to step 1
   - until convergence
Example: The *K*-Means Clustering Method

1. Arbitrarily choose *K* objects as initial cluster centers.
2. Assign each object to the most similar center.
3. Update the cluster means.
4. Reassign objects to the nearest new cluster.
5. Update the cluster means.
6. Repeat steps 3-5 until convergence.
The criterion function

- The square-error criterion

\[ E = \sum_{i=1}^{k} \sum_{p \in C_i} |p - m_i|^2 \]

- where \( E \) is the sum of the square error for all objects in the data set;
- \( p \) is the point in space representing a given object; and
- \( m_i \) is the mean of cluster \( C_i \) (both \( p \) and \( m_i \) are multidimensional)
Weakness of K-means method

- Often terminate at a local optimum, The *global optimum* may be found using techniques such as: *deterministic annealing* and *genetic algorithms*.

- Applicable only when *mean* is defined, then what about categorical data?

- Need to specify $k$, the *number* of clusters, in advance.

- Unable to handle noisy data and *outliers*.
The end of
Chapter 4: Algorithms:
The Basic Methods