Introduction
- Decision Trees
- TDIDT: Top-Down Induction of Decision Trees

ID3
- Attribute selection
- Entropy, Information, Information Gain
- Gain Ratio

C4.5
- Numeric Values
- Missing Values
- Pruning
- Regression and Model Trees

Acknowledgements:
Many slides based on Frank & Witten, a few on Kan, Steinbach & Kumar
Decision Trees

- a decision tree consists of
  - **Nodes:**
    - test for the value of a certain attribute
  - **Edges:**
    - correspond to the outcome of a test
    - connect to the next node or leaf
  - **Leaves:**
    - terminal nodes that predict the outcome

to classify an example:

1. start at the root
2. perform the test
3. follow the edge corresponding to outcome
4. goto 2. unless leaf
5. predict that outcome associated with the leaf
In **Decision Tree Learning**, a new example is classified by submitting it to a series of tests that determine the class label of the example. These tests are organized in a hierarchical structure called a *decision tree*. The training examples are used for choosing appropriate tests in the decision tree. Typically, a tree is built from top to bottom, where tests that maximize the information gain about the classification are selected first.
A Sample Task

<table>
<thead>
<tr>
<th>Day</th>
<th>Temperature</th>
<th>Outlook</th>
<th>Humidity</th>
<th>Windy</th>
<th>Play Golf?</th>
</tr>
</thead>
<tbody>
<tr>
<td>07-05</td>
<td>hot</td>
<td>sunny</td>
<td>high</td>
<td>false</td>
<td>no</td>
</tr>
<tr>
<td>07-06</td>
<td>hot</td>
<td>sunny</td>
<td>high</td>
<td>true</td>
<td>no</td>
</tr>
<tr>
<td>07-07</td>
<td>hot</td>
<td>overcast</td>
<td>high</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>07-09</td>
<td>cool</td>
<td>rain</td>
<td>normal</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>07-10</td>
<td>cool</td>
<td>overcast</td>
<td>normal</td>
<td>true</td>
<td>yes</td>
</tr>
<tr>
<td>07-12</td>
<td>mild</td>
<td>sunny</td>
<td>high</td>
<td>false</td>
<td>no</td>
</tr>
<tr>
<td>07-14</td>
<td>cool</td>
<td>sunny</td>
<td>normal</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>07-15</td>
<td>mild</td>
<td>rain</td>
<td>normal</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>07-20</td>
<td>mild</td>
<td>sunny</td>
<td>normal</td>
<td>true</td>
<td>yes</td>
</tr>
<tr>
<td>07-21</td>
<td>mild</td>
<td>overcast</td>
<td>high</td>
<td>true</td>
<td>yes</td>
</tr>
<tr>
<td>07-22</td>
<td>hot</td>
<td>overcast</td>
<td>normal</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>07-23</td>
<td>mild</td>
<td>rain</td>
<td>high</td>
<td>true</td>
<td>no</td>
</tr>
<tr>
<td>07-26</td>
<td>cool</td>
<td>rain</td>
<td>normal</td>
<td>true</td>
<td>no</td>
</tr>
<tr>
<td>07-30</td>
<td>mild</td>
<td>rain</td>
<td>high</td>
<td>false</td>
<td>yes</td>
</tr>
</tbody>
</table>

| today    | cool        | sunny   | normal   | false | ?          |
| tomorrow | mild        | sunny   | normal   | false | ?          |
Decision Tree Learning
Divide-And-Conquer Algorithms

- Family of decision tree learning algorithms
  - TDIDT: Top-Down Induction of Decision Trees
- Learn trees in a Top-Down fashion:
  - divide the problem in subproblems
  - solve each problem

Basic Divide-And-Conquer Algorithm:

1. select a test for root node
   Create branch for each possible outcome of the test
2. split instances into subsets
   One for each branch extending from the node
3. repeat recursively for each branch, using only instances that reach the branch
4. stop recursion for a branch if all its instances have the same class
ID3 Algorithm

Function ID3

- **Input:** Example set $S$
- **Output:** Decision Tree $DT$

- If all examples in $S$ belong to the same class $c$
  - return a new leaf and label it with $c$
- Else
  - i. Select an attribute $A$ according to some heuristic function
  - ii. Generate a new node $DT$ with $A$ as test
  - iii. For each Value $v_i$ of $A$
    - (a) Let $S_i =$ all examples in $S$ with $A = v_i$
    - (b) Use ID3 to construct a decision tree $DT_i$ for example set $S_i$
    - (c) Generate an edge that connects $DT$ and $DT_i$
A Different Decision Tree

- also explains all of the training data
- will it generalize well to new data?
Which attribute to select as the root?
What is a good Attribute?

- We want to grow a simple tree
  → a good heuristic prefers attributes that split the data so that each successor node is as *pure* as possible
    - i.e., the distribution of examples in each node is so that it mostly contains examples of a single class

- In other words:
  - We want a measure that prefers attributes that have a high degree of „order“:
    - Maximum order: All examples are of the same class
    - Minimum order: All classes are equally likely
  → *Entropy* is a measure for (un-)orderedness

- Another interpretation:
  - Entropy is the amount of information that is contained in the node
  - all examples of the same class → no information
Entropy (for two classes)

- $S$ is a set of examples
- $p_{\oplus}$ is the proportion of examples in class $\oplus$
- $p_{\ominus} = 1 - p_{\oplus}$ is the proportion of examples in class $\ominus$

Entropy:

$$E(S) = -p_{\oplus} \cdot \log_2 p_{\oplus} - p_{\ominus} \cdot \log_2 p_{\ominus}$$

- Interpretation:
  - amount of unorderedness in the class distribution of $S$

maximal value at equal class distribution

minimal value if only one class left in $S$
Example: Attribute Outlook

- **Outlook = sunny:** 2 examples yes, 3 examples no
  \[ E(\text{Outlook} = \text{sunny}) = -\frac{2}{5} \log_2 \left( \frac{2}{5} \right) - \frac{3}{5} \log_2 \left( \frac{3}{5} \right) = 0.971 \]

- **Outlook = overcast:** 4 examples yes, 0 examples no
  \[ E(\text{Outlook} = \text{overcast}) = -1 \cdot \log_2 (1) - 0 \cdot \log_2 (0) = 0 \]

- **Outlook = rainy:** 3 examples yes, 2 examples no
  \[ E(\text{Outlook} = \text{rainy}) = -\frac{3}{5} \log_2 \left( \frac{3}{5} \right) - \frac{2}{5} \log_2 \left( \frac{2}{5} \right) = 0.971 \]

*Note:* this is normally undefined. Here: = 0
Entropy (for more classes)

Entropy can be easily generalized for \( n > 2 \) classes

- \( p_i \) is the proportion of examples in \( S \) that belong to the \( i \)-th class
  \[
  E(S) = -p_1 \log_2 p_1 - p_2 \log_2 p_2 \ldots - p_n \log_2 p_n = -\sum_{i=1}^{n} p_i \log_2 p_i
  \]

- Calculation can be simplified using absolute counts \( c_i \) of examples in class \( i \) instead of fractions
  - If \( p_i = \frac{c_i}{|S|} \):
    \[
    E(S) = -\sum_{i=1}^{n} p_i \log_2 p_i = -\frac{1}{|S|} \left( \sum_{i=1}^{n} c_i \log_2 c_i - |S| \cdot \log_2 |S| \right)
    \]

- Example:
  \[
  E([2,3,4]) = -\frac{2}{9} \cdot \log_2 \left( \frac{2}{9} \right) - \frac{3}{9} \cdot \log_2 \left( \frac{3}{9} \right) - \frac{4}{9} \cdot \log_2 \left( \frac{4}{9} \right) \\
  = -\frac{1}{9} \left( 2 \cdot \log_2 (2) + 3 \cdot \log_2 (3) + 4 \cdot \log_2 (4) - 9 \cdot \log_2 (9) \right)
  \]
**Average Entropy / Information**

**Problem:**
- Entropy only computes the quality of a single (sub-)set of examples
  - corresponds to a single value
- How can we compute the quality of the entire split?
  - corresponds to an entire attribute

**Solution:**
- Compute the weighted average over all sets resulting from the split
  - weighted by their size

\[
I(S, A) = \sum_i \frac{|S_i|}{|S|} \cdot E(S_i)
\]

**Example:**
- Average entropy for attribute *Outlook*:

\[
I(\text{Outlook}) = \frac{5}{14} \cdot 0.971 + \frac{4}{14} \cdot 0 + \frac{5}{14} \cdot 0.971 = 0.693
\]
Information Gain

- When an attribute $A$ splits the set $S$ into subsets $S_i$
  - we compute the average entropy
  - and compare the sum to the entropy of the original set $S$

Information Gain for Attribute $A$

$$Gain(S, A) = E(S) - I(S, A) = E(S) - \sum_i \frac{|S_i|}{|S|} \cdot E(S_i)$$

- The attribute that maximizes the difference is selected
  - i.e., the attribute that reduces the unorderedness most!

Note:
- maximizing information gain is equivalent to minimizing average entropy, because $E(S)$ is constant for all attributes $A$
Example

\[ S: [9+,5-] \]
\[ E = 0.940 \]

\[ \text{Humidity} \]

\[ \text{High} \]
\[ [3+,4-] \]
\[ E = 0.985 \]

\[ \text{Normal} \]
\[ [6+,1-] \]
\[ E = 0.592 \]

\[ S: [9+,5-] \]
\[ E = 0.940 \]

\[ \text{Wind} \]

\[ \text{Weak} \]
\[ [6+,2-] \]
\[ E = 0.811 \]

\[ \text{Strong} \]
\[ [3+,3-] \]
\[ E = 1.00 \]

\[ \text{Gain} (S, \text{Humidity}) = 0.246 \]
\[ = 0.940 - (7/14) \cdot 0.985 - (7/14) \cdot 0.592 \]
\[ = 0.151 \]

\[ \text{Gain} (S, \text{Temperature}) = 0.029 \]
\[ = 0.940 - (8/14) \cdot 0.811 - (6/14) \cdot 1.0 \]
\[ = 0.048 \]
Example (Ctd.)

Outlook is selected as the root node

Outlook = overcast contains only examples of class yes

Outlook = overcast contains only examples of class yes

Further splitting necessary
Example (Ctd.)

Gain(\textit{Temperature}) = 0.571 \text{ bits}
Gain(\textit{Humidity}) = 0.971 \text{ bits}
Gain(\textit{Windy}) = 0.020 \text{ bits}

\textbf{Humidity is selected}
Example (Ctd.)

Humidity is selected

Further splitting necessary

Pure leaves
→ No further expansion necessary
Final decision tree

- Outlook:
  - Sunny
  - Overcast
  - Rainy

- Humidity:
  - High
  - Normal

- Windy:
  - False
  - True

- Final decision:
  - Yes
  - No
Properties of Entropy

Entropy is the only function that satisfies all of the following three properties

1. When node is pure, measure should be zero
2. When impurity is maximal (i.e. all classes equally likely), measure should be maximal
3. Measure should obey multistage property:
   - \( p, q, r \) are classes in set \( S \), and \( T \) are examples of class \( t = q \lor r \)
   \[
   E_{p,q,r}(S) = E_{p,t}(S) + \frac{|T|}{|S|} \cdot E_{q,r}(T)
   \]
   → decisions can be made in several stages
   - For example:
     \[
     E([2,3,4]) = E([2,7]) + \frac{7}{9} \cdot E([3,4])
     \]
Highly-branching attributes

Problematic: attributes with a large number of values

- extreme case: each example has its own value
  - e.g. example ID; Day attribute in weather data

- Subsets are more likely to be pure if there is a large number of different attribute values
  - Information gain is biased towards choosing attributes with a large number of values

- This may cause several problems:
  - **Overfitting**
    - selection of an attribute that is non-optimal for prediction
  - **Fragmentation**
    - data are fragmented into (too) many small sets
Decision Tree for Day attribute

- Entropy of split:

\[ I(\text{Day}) = \frac{1}{14} \left( E(\left[0,1\right]) + E(\left[0,1\right]) + \ldots + E(\left[0,1\right]) \right) = 0 \]

- Information gain is maximal for Day (0.940 bits)
Intrinsic Information of an Attribute

- Intrinsic information of a split
  - entropy of distribution of instances into branches
  - i.e. how much information do we need to tell which branch an instance belongs to

\[
IntI(S, A) = - \sum_i \frac{|S_i|}{|S|} \log_2 \left( \frac{|S_i|}{|S|} \right)
\]

- Example:
  - Intrinsic information of Day attribute:

\[
IntI(Day) = 14 \times \left( -\frac{1}{14} \cdot \log_2 \left( \frac{1}{14} \right) \right) = 3.807
\]

- Observation:
  - Attributes with higher intrinsic information are less useful
Gain Ratio

- modification of the information gain that reduces its bias towards multi-valued attributes
- takes number and size of branches into account when choosing an attribute
  - corrects the information gain by taking the *intrinsic information* of a split into account
- Definition of Gain Ratio:

  \[ GR(S, A) = \frac{Gain(S, A)}{IntI(S, A)} \]

- Example:
  - Gain Ratio of Day attribute

  \[ GR(Day) = \frac{0.940}{3,807} = 0.246 \]
### Gain ratios for weather data

<table>
<thead>
<tr>
<th>Outlook</th>
<th>Temperature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Info:</td>
<td>0.693</td>
</tr>
<tr>
<td>Gain: 0.940-0.693</td>
<td>0.247</td>
</tr>
<tr>
<td>Split info: info([5,4,5])</td>
<td>1.577</td>
</tr>
<tr>
<td>Gain ratio: 0.247/1.577</td>
<td>0.157</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Humidity</th>
<th>Windy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Info:</td>
<td>0.788</td>
</tr>
<tr>
<td>Gain: 0.940-0.788</td>
<td>0.152</td>
</tr>
<tr>
<td>Split info: info([7,7])</td>
<td>1.000</td>
</tr>
<tr>
<td>Gain ratio: 0.152/1</td>
<td>0.152</td>
</tr>
</tbody>
</table>

- Day attribute would still win...
  - one has to be careful which attributes to add...
- Nevertheless: Gain ratio is more reliable than Information Gain
Gini Index

- Many alternative measures to Information Gain
- Most popular alternative: Gini index
  - used in e.g., in CART (Classification And Regression Trees)
  - impurity measure (instead of entropy)

\[
Gini(S) = \sum_i p_i (1 - p_i) = 1 - \sum_i p_i^2
\]

- average Gini index (instead of average entropy / information)

\[
Gini(S, A) = \sum_i \frac{|S_i|}{|S|} \cdot Gini(S_i)
\]

- Gini Gain
  - could be defined analogously to information gain
  - but typically average Gini index is minimized instead of maximizing Gini gain
Comparison of Splitting Criteria

For a 2-class problem:

- **Entropy**
- **2*Gini**
- **2*Error**
Why not use Error as a Splitting Criterion?

- **Reason:**
  - The bias towards pure leaves is not strong enough
  - **Example 1:** Data set with 160 Examples A, 40 Examples B
    - Error rate without splitting is 20%

For each of the two splits, the total error after splitting is also $(0% + 40%)/2 = 20%$ → no improvement

However, together both splits would give a perfect classifier.

Based on a slide by Richard Lawton
Why not use Error as a Splitting Criterion?

- **Reason:**
  - The bias towards pure leaves is not strong enough
- **Example 2:**
  - Dataset with 400 examples of class A and 400 examples of class B

```
400 of A
400 of B

200 of A
400 of B

200 of A
0 of B

Error rate = 25%
```

```
400 of A
400 of B

300 of A
100 of B

100 of A
300 of B

Error rate = 25%
```

Based on a slide by Richard Lawton
Industrial-strength algorithms

- For an algorithm to be useful in a wide range of real-world applications it must:
  - Permit numeric attributes
  - Allow missing values
  - Be robust in the presence of noise
  - Be able to approximate arbitrary concept descriptions (at least in principle)

→ ID3 needs to be extended to be able to deal with real-world data

- Result: **C4.5**
  - Best-known and (probably) most widely-used learning algorithm
    - original C-implementation at [http://www.rulequest.com/Personal/](http://www.rulequest.com/Personal/)
    - Re-implementation of C4.5 Release 8 in Weka: J4.8
    - Commercial successor: C5.0
Missing values

- Examples are classified as usual
  - if we are lucky, attributes with missing values are not tested by the tree
- If an attribute with a missing value needs to be tested:
  - split the instance into fractional instances (pieces)
  - one piece for each outgoing branch of the node
  - a piece going down a branch receives a weight proportional to the popularity of the branch
  - weights sum to 1
- Info gain or gain ratio work with fractional instances
  - use sums of weights instead of counts
- during classification, split the instance in the same way
  - Merge probability distribution using weights of fractional instances
Numeric attributes

- Standard method: binary splits
  - E.g. temp < 45
- Unlike nominal attributes, every attribute has many possible split points
- Solution is straightforward extension:
  - Evaluate info gain (or other measure) for every possible split point of attribute
  - Choose “best” split point
  - Info gain for best split point is info gain for attribute
→ Computationally more demanding than splits on discrete attributes
Example

- Assume a numerical attribute for Temperature
- First step:
  - Sort all examples according to the value of this attribute
  - Could look like this:

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

Temperature < 71.5: yes/4, no/2  
Temperature ≥ 71.5: yes/5, no/3

\[
I(\text{Temperature @ 71.5}) = \frac{6}{14} \cdot E(\text{Temperature < 71.5}) + \frac{8}{14} E(\text{Temperature ≥ 71.5}) = 0.939
\]

- Split points can be placed between values or directly at values
- Has to be computed for all pairs of neighboring values
Efficient Computation

- Efficient computation needs only one scan through the values!
  - Linearly scan the sorted values, each time updating the count matrix and computing the evaluation measure
  - Choose the split position that has the best value

### Cheat

<table>
<thead>
<tr>
<th></th>
<th>No</th>
<th>No</th>
<th>No</th>
<th>Yes</th>
<th>Yes</th>
<th>Yes</th>
<th>No</th>
<th>No</th>
<th>No</th>
<th>No</th>
</tr>
</thead>
</table>

### Taxable Income

<table>
<thead>
<tr>
<th></th>
<th>60</th>
<th>70</th>
<th>75</th>
<th>85</th>
<th>90</th>
<th>95</th>
<th>100</th>
<th>120</th>
<th>125</th>
<th>220</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;= &gt;</td>
<td>&lt;= &gt;</td>
<td>&lt;= &gt;</td>
<td>&lt;= &gt;</td>
<td>&lt;= &gt;</td>
<td>&lt;= &gt;</td>
<td>&lt;= &gt;</td>
<td>&lt;= &gt;</td>
<td>&lt;= &gt;</td>
<td>&lt;= &gt;</td>
<td>&lt;= &gt;</td>
</tr>
</tbody>
</table>

### Sorted Values

<table>
<thead>
<tr>
<th></th>
<th>55</th>
<th>65</th>
<th>72</th>
<th>80</th>
<th>87</th>
<th>92</th>
<th>97</th>
<th>110</th>
<th>122</th>
<th>125</th>
<th>172</th>
<th>230</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>No</td>
<td>0</td>
<td>7</td>
<td>1</td>
<td>6</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Gini</td>
<td>0.420</td>
<td>0.400</td>
<td>0.375</td>
<td>0.343</td>
<td>0.417</td>
<td>0.400</td>
<td>0.300</td>
<td>0.343</td>
<td>0.375</td>
<td>0.400</td>
<td>0.420</td>
<td></td>
</tr>
</tbody>
</table>
Binary vs. Multiway Splits

- Splitting (multi-way) on a nominal attribute exhausts all information in that attribute
  - Nominal attribute is tested (at most) once on any path in the tree
- Not so for binary splits on numeric attributes!
  - Numeric attribute may be tested several times along a path in the tree
- Disadvantage: tree is hard to read

Remedy:
- pre-discretize numeric attributes (→ discretization), or
- use multi-way splits instead of binary ones
  - can, e.g., be computed by building a subtree using a single numerical attribute.
  - subtree can be flattened into a multiway split
  - other methods possible (dynamic programming, greedy...)

Overfitting and Pruning

- The smaller the complexity of a concept, the less danger that it overfits the data
  - A polynomial of degree $n$ can always fit $n+1$ points
- Thus, learning algorithms try to keep the learned concepts simple
  - Note a „perfect“ fit on the training data can always be found for a decision tree! (except when data are contradictory)

**Pre-Pruning:**
- stop growing a branch when information becomes unreliable

**Post-Pruning:**
- grow a decision tree that correctly classifies all training data
- simplify it later by replacing some nodes with leaves

- Postpruning preferred in practice—prepruning can “stop early”
Prepruning

- Based on statistical significance test
  - Stop growing the tree when there is no *statistically significant* association between any attribute and the class at a particular node
- Most popular test: *chi-squared test*
- ID3 used chi-squared test in addition to information gain
  - Only statistically significant attributes were allowed to be selected by information gain procedure
- C4.5 uses a simpler strategy
  - but combines it with → post-pruning
  - parameter -m: (default value m=2)
    - each node above a leave must have
      - at least two successors
      - that contain at least m examples
Early Stopping

- Pre-pruning may stop the growth process prematurely: *early stopping*

- Classic example: XOR/Parity-problem
  - No individual attribute exhibits any significant association to the class
  
  → In a dataset that contains XOR attributes a and b, and several irrelevant (e.g., random) attributes, ID3 cannot distinguish between relevant and irrelevant attributes
  
  → Pre-pruning won’t expand the root node
  - Structure is only visible in fully expanded tree

- But:
  - XOR-type problems rare in practice
  - Pre-pruning is faster than postpruning
Post-Pruning

- **basic idea**
  - first grow a full tree to capture all possible attribute interactions
  - later remove those that are due to chance

1. **learn a complete and consistent decision tree that classifies all examples in the training set correctly**
2. **as long as the performance increases**
   - try **simplification operators** on the tree
   - evaluate the resulting trees
   - make the replacement that results in the **best estimated performance**
3. return the resulting decision tree
Postpruning

- Two subtree simplification operators
  - Subtree replacement
  - Subtree raising

- Possible performance evaluation strategies
  - error estimation
    - on separate pruning set ("reduced error pruning")
    - with confidence intervals (C4.5's method)
  - significance testing
  - MDL principle
Subtree Replacement

Proceeds Bottom-up:
- consider replacing a tree only after considering all its subtrees
- may make a difference for complexity-based heuristics
Subtree Raising

- Delete node B
- Redistribute instances of leaves 4 and 5 into C
Estimating Error Rates

- Prune only if it does not increase the estimated error
  - Error on the training data is NOT a useful estimator (would result in almost no pruning)

- Reduced Error Pruning
  - Use hold-out set for pruning
  - Essentially the same as in rule learning
    - only pruning operators differ (subtree replacement)

- C4.5’s method
  - Derive confidence interval from training data
    - with a user-provided confidence level
  - Assume that the true error is on the upper bound of this confidence interval (pessimistic error estimate)
Reduced Error Pruning

Basic Idea

- optimize the accuracy of a decision tree on a separate pruning set

1. split training data into a growing and a pruning set
2. learn a complete and consistent decision tree that classifies all examples in the growing set correctly
3. as long as the error on the pruning set does not increase
   - try to replace each node by a leaf (predicting the majority class)
   - evaluate the resulting (sub-)tree on the pruning set
   - make the replacement that results in the maximum error reduction
4. return the resulting decision tree
Pessimistic Error Rates

- Consider classifying \( E \) examples incorrectly out of \( N \) examples as observing \( E \) events in \( N \) trials in the binomial distribution.
- For a given confidence level \( CF \), the upper limit on the error rate over the whole population is \( U_{CF}(E, N) \) with \( CF\% \) confidence.
- Example:
  - 100 examples in a leaf
  - 6 examples misclassified
  - How large is the true error assuming a pessimistic estimate with a confidence of 25%?
- Note:
  - this is only a heuristic!
  - but one that works well
C4.5’s method

- **Pessimistic error estimate for a node**

\[
e = \frac{f + \frac{z^2}{2N} + z\sqrt{\frac{f}{N} - \frac{f^2}{N^2} + \frac{z^2}{4N^2}}}{1 + \frac{z^2}{N}}
\]

- \(z\) is derived from the desired confidence value
  - If \(c = 25\%\) then \(z = 0.69\) (from normal distribution)
  - \(f\) is the error on the training data
  - \(N\) is the number of instances covered by the leaf

- **Error estimate for subtree** is weighted sum of error estimates for all its leaves

→ A node is pruned if error estimate of subtree is lower than error estimate of the node
Example

Combined using ratios 6:2:6 gives 0.51
C4.5: choices and options

- C4.5 has several parameters
  - \texttt{-c} Confidence value (default 25%): lower values incur heavier pruning
  - \texttt{-m} Minimum number of instances in the two most popular branches (default 2)
  - Others for, e.g., having only two-way splits (also on symbolic attributes), etc.
Sample Experimental Evaluation

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Tree Size</th>
<th>Purity</th>
<th>Predictive Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Pruning (C4.5-m1)</td>
<td>547</td>
<td>99.7%</td>
<td>60.3% (± 4.8)</td>
</tr>
<tr>
<td>C4.5 -m2</td>
<td>314</td>
<td>91.8%</td>
<td>60.1% (± 3.3)</td>
</tr>
<tr>
<td>C4.5 -m5</td>
<td>170</td>
<td>82.3%</td>
<td>60.4% (± 5.7)</td>
</tr>
<tr>
<td>C4.5 -m10</td>
<td>90</td>
<td>76.6%</td>
<td>60.0% (± 5.2)</td>
</tr>
<tr>
<td>C4.5 -m15</td>
<td>62</td>
<td>74.1%</td>
<td>61.6% (± 4.7)</td>
</tr>
<tr>
<td>C4.5 -m20</td>
<td>47</td>
<td>71.9%</td>
<td>62.7% (± 2.0)</td>
</tr>
<tr>
<td>C4.5 -m25</td>
<td>37</td>
<td>71.3%</td>
<td>63.0% (± 2.2)</td>
</tr>
<tr>
<td>C4.5 -m30</td>
<td>26</td>
<td>70.1%</td>
<td>65.1% (± 2.5)</td>
</tr>
<tr>
<td>C4.5 -m35</td>
<td>22</td>
<td>69.9%</td>
<td>65.0% (± 4.2)</td>
</tr>
<tr>
<td>C4.5 -m40</td>
<td>20</td>
<td>69.2%</td>
<td>64.8% (± 2.6)</td>
</tr>
<tr>
<td>C4.5 -m50</td>
<td>24</td>
<td>69.1%</td>
<td>64.5% (± 3.5)</td>
</tr>
<tr>
<td>C4.5 -c75</td>
<td>524</td>
<td>99.7%</td>
<td>61.0% (± 4.5)</td>
</tr>
<tr>
<td>C4.5 -c50</td>
<td>357</td>
<td>95.3%</td>
<td>60.2% (± 3.6)</td>
</tr>
<tr>
<td>C4.5 -c25</td>
<td>257</td>
<td>91.2%</td>
<td>62.3% (± 4.4)</td>
</tr>
<tr>
<td>C4.5 -c15</td>
<td>137</td>
<td>81.8%</td>
<td>64.8% (± 4.6)</td>
</tr>
<tr>
<td>C4.5 -c10</td>
<td>75</td>
<td>76.9%</td>
<td>65.9% (± 4.9)</td>
</tr>
<tr>
<td>C4.5 -c5</td>
<td>53</td>
<td>74.7%</td>
<td>63.8% (± 6.0)</td>
</tr>
<tr>
<td>C4.5 -c1</td>
<td>27</td>
<td>70.2%</td>
<td>63.4% (± 5.8)</td>
</tr>
<tr>
<td>C4.5 Default</td>
<td>173</td>
<td>86.2%</td>
<td>62.5% (± 5.2)</td>
</tr>
<tr>
<td>C4.5 -m30 -c10</td>
<td>20</td>
<td>69.6%</td>
<td>66.7% (± 3.7)</td>
</tr>
<tr>
<td>Mode Prediction</td>
<td>1</td>
<td>56.8%</td>
<td>56.8%</td>
</tr>
</tbody>
</table>

Typical behavior with growing $m$ and decreasing $c$

- tree size and training accuracy (= purity)
  - always decrease
- predictive accuracy
  - first increases (overfitting avoidance)
  - then decreases (over-generalization)
- ideal value on this data set near
  - $m = 30$
  - $c = 10$
Complexity of tree induction

- Assume
  - $m$ attributes, $n$ training instances
  - tree depth $O(\log n)$
  - tree has $O(n)$ nodes ($\leq$ one leaf per example)

Costs for
- Building a tree $O(m \cdot n \cdot \log n)$
- Subtree replacement $O(n)$
  - counts of covered instances can be reused from training
- Subtree raising $O(n \cdot (\log n)^2)$
  - Every instance may have to be redistributed at every node between its leaf and the root
  - Cost for redistribution (on average): $O(\log n)$

$\rightarrow$ Total cost: $O(m \cdot n \cdot \log n) + O(n \cdot (\log n)^2)$
Error-Complexity Measure

- CART uses the following measure

\[ R_\alpha(T) = f(T) + \alpha \cdot L(T) \]

- \( f \) is the error rate on the training data
- \( L \) is the number leaves in the tree
- \( \alpha \) is a parameter that trades off tree complexity vs. test error

- Different values of \( \alpha \) prefer different trees
  - smaller values of \( \alpha \) prefer different trees with low training error
  - larger values of \( \alpha \) prefer different trees with low training error
  - would nowadays be called a \textit{regularization} parameter
Error-Complexity Pruning
(CART, Breiman et al. 1984)

- Generate a sequence of trees with decreasing complexity
  \[ T_0 \rightarrow T_1 \rightarrow \ldots \rightarrow T_m \]
  - \( T_0 \) is the full tree, \( T_m \) is the tree that consists only of the root node
  - Each tree is generated from its predecessor by replacing a subtree with a node.
    - It selects the node which results in the smallest increase in the error function, weighted by the number of leaves in the subtree
  - this sequence of trees optimizes \( R_\alpha \) for successive ranges of \( \alpha \)-values
    - \( T_0 \) is optimal for the range \([0, \alpha_1]\)
    - \( T_1 \) is optimal for the range \([\alpha_1, \alpha_2]\)
    - ...
  - Optimal values for \( \alpha \) are then determined with cross-validation on the training data
Simple way:
- one rule for each leaf

- yes :- Outlook = overcast.
- yes :- Outlook = sunny,
  Humidity = normal.
- no :- Outlook = rain,
  Windy = true.
- no :- Outlook = rainy,
  Humidity = high.
- yes :- Outlook = rain,
  Windy = false.
C4.5rules and successors

- **C4.5rules:**
  - greedily prune conditions from each rule if this reduces its estimated error
    - Can produce duplicate rules
    - Check for this at the end
  - Then look at each class in turn
    - consider the rules for that class
    - find a “good” subset (guided by MDL)
    - rank the subsets to avoid conflicts
  - Finally, remove rules (greedily) if this decreases error on the training data
- C4.5rules slow for large and noisy datasets
- Commercial version C5.0rules uses a different technique
  - Much faster and a bit more accurate
Decision Lists and Decision Graphs

- **Decision Lists**
  - An ordered list of rules
  - the first rule that fires makes the prediction
  - can be learned with a covering approach

- **Decision Graphs**
  - Similar to decision trees, but nodes may have multiple predecessors
  - DAGs: Directed, acyclic graphs
  - there are a few algorithms that can learn DAGs
    - learn much smaller structures
    - but in general not very successful

- **Special case:**
  - a decision list may be viewed as a special case of a DAG
Example

- A decision list for a rule set with rules
  - with 4, 2, 2, 1 conditions, respectively
  - drawn as a decision graph

![Decision Tree Diagram]

Rule 1
Rule 2
Rule 3
Rule 4
Rules vs. Trees

- Each decision tree can be converted into a rule set
  → Rule sets are at least as expressive as decision trees
    - a decision tree can be viewed as a set of non-overlapping rules
    - typically learned via *divide-and-conquer* algorithms (recursive partitioning)
- Transformation of rule sets / decision lists into trees is less trivial
  - Many concepts have a shorter description as a rule set
  - low complexity decision lists are more expressive than low complexity decision trees (Rivest, 1987)
  - exceptions: if one or more attributes are relevant for the classification of all examples (e.g., parity)
- Learning strategies:
  - Separate-and-Conquer vs. Divide-and-Conquer
Discussion TDIDT

- The most extensively studied method of machine learning used in data mining
- Different criteria for attribute/test selection rarely make a large difference
- Different pruning methods mainly change the size of the resulting pruned tree
- C4.5 builds univariate decision trees
- Some TDIDT systems can build multivariate trees (e.g. CART)
  - multi-variate: a split is not based on a single attribute but on a function defined on multiple attributes
Regression Problems

- **Regression Task**
  - the target variable \( y = f(x) \) is numerical instead of discrete
  - Various error functions, e.g., **Mean-squared error**:
    \[
    L(f, \hat{f}) = \sum_x (f(x) - \hat{f}(x))^2
    \]

Two principal approaches

- Discretize the numerical target variable
  - e.g., equal-width intervals, or equal-frequency
  - and use a classification learning algorithm

- Adapt the classification algorithm to regression data → Regression Trees and Model Trees
Regression Trees

Differences to Decision Trees (Classification Trees)

- **Leaf Nodes:**
  - Predict the average value of all instances in this leaf

- **Splitting criterion:**
  - Minimize the variance of the values in each subset $S_i$
  - Standard deviation reduction
    \[
    SDR(A, S) = SD(S) - \sum_i \frac{|S_i|}{|S|} SD(S_i)
    \]

- **Termination criteria:**
  - Very important! (otherwise only single points in each leaf)
    - lower bound on standard deviation in a node
    - lower bound on number of examples in a node

- **Pruning criterion:**
  - Numeric error measures, e.g. Mean-Squared Error
**CART** (Breiman et al. 1984)

- Algorithm for learning **Classification And Regression Trees**
  - Quite similar to ID3/C4.5, but developed independently in the statistics community
- Splitting criterion:
  - Gini-Index for Classification
  - Sum-of-Squares for Regression
- Pruning:
  - Cost-Complexity Pruning
Regression Tree Example

- Task:
  understand how computer performance is related to a number of variables which describe the features of a PC

- Data:
  - the size of the cache,
  - the cycle time of the computer,
  - the memory size
  - the number of channels (both the last two were not measured but minimum and maximum values obtained).

Based on a slide by Richard Lawton
Regression Tree

Based on a slide by Richard Lawton
Estimated Relative Error over Tree Complexity

Based on a slide by Richard Lawton
Model Trees

- In a Leaf node
  - Classification Trees predict a class value
  - Regression Trees predict the average value of all instances in the model
  - Model Trees use a linear model for making the predictions
    - growing of the tree is as with Regression Trees

- Linear Model:
  - $LM(x) = \sum_i w_i v_i(x)$ where $v_i(x)$ is the value of attribute $A_i$
    - for example $x$ and $w_i$ is a weight
  - The attributes that have been used in the path of the tree can be ignored
  - Weights can be fitted with standard math packages
    - Minimize the Mean Squared Error $MSE = \frac{1}{n} \sum_j (y_j - r_j)^2$
Summary

- **Classification Problems** require the prediction of a discrete target value
  - can be solved using decision tree learning
  - iteratively select the best attribute and split up the values according to this attribute

- **Regression Problems** require the prediction of a numerical target value
  - can be solved with regression trees and model trees
  - difference is in the models that are used at the leafs
  - are grown like decision trees, but with different splitting criteria

- **Overfitting** is a serious problem!
  - simpler, seemingly less accurate trees are often preferable
  - evaluation has to be done on separate test sets