Linear separability

- A dataset is **linearly separable** iff there exists a separating hyperplane:
  - Exists \( w \), such that:
    - \( w_0 + \sum_i w_i x_i > 0; \) if \( x = \{x_1, \ldots, x_k\} \) is a positive example
    - \( w_0 + \sum_i w_i x_i < 0; \) if \( x = \{x_1, \ldots, x_k\} \) is a negative example
Not linearly separable data

- Some datasets are **not linearly separable!**

Addressing non-linearly separable data – Option 1, non-linear features

- Choose non-linear features, e.g.,
  - Typical linear features: $w_0 + \sum_i w_i x_i$
  - Example of non-linear features:
    - Degree 2 polynomials, $w_0 + \sum_i w_i x_i + \sum_{ij} w_{ij} x_i x_j$
- Classifier $h_w(x)$ still linear in parameters $w$
  - As easy to learn
  - Data is linearly separable in higher dimensional spaces
  - More discussion later this quarter
Addressing non-linearly separable data – Option 2, non-linear classifier

- Choose a classifier \( h_w(x) \) that is non-linear in parameters \( w \), e.g.,
  - Decision trees, boosting, nearest neighbor, neural networks...

- More general than linear classifiers

- But, can often be harder to learn (non-convex/concave optimization required)

- But, but, often very useful

- (BTW. Later this quarter, we'll see that these options are not that different)

A small dataset: Miles Per Gallon

Suppose we want to predict MPG

From the UCI repository (thanks to Ross Quinlan)
A Decision Stump

Recursion Step

Take the Original Dataset...

And partition it according to the value of the attribute we split on

Examples in which cylinders = 4

Examples in which cylinders = 5

Examples in which cylinders = 6

Examples in which cylinders = 8
Recursion Step

Build tree from These examples..

Second level of tree

Recursively build a tree from the seven records in which there are four cylinders and the maker was based in Asia
Classification of a new example

Classifying a test example – traverse tree and report leaf label
Are all decision trees equal?

- Many trees can represent the same concept
- But, not all trees will have the same size!
  - e.g., $\phi = A \land B \lor \neg A \land C$ ((A and B) or (not A and C))

Learning decision trees is hard!!!

- Learning the simplest (smallest) decision tree is an NP-complete problem [Hyafil & Rivest '76]
- Resort to a greedy heuristic:
  - Start from empty decision tree
  - Split on next best attribute (feature)
  - Recurse
Choosing a good attribute

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

Measuring uncertainty

- Good split if we are more certain about classification after split
  - Deterministic good (all true or all false)
  - Uniform distribution bad

\[
\begin{array}{l}
P(Y=A) = 1/2 \\
P(Y=B) = 1/4 \\
P(Y=C) = 1/8 \\
P(Y=D) = 1/8 \\
\end{array}
\]

\[
\begin{array}{l}
P(Y=A) = 1/4 \\
P(Y=B) = 1/4 \\
P(Y=C) = 1/4 \\
P(Y=D) = 1/4 \\
\end{array}
\]
Entropy

Entropy $H(Y)$ of a random variable $Y$

$$H(Y) = - \sum_{i=1}^{k} P(Y = y_i) \log_2 P(Y = y_i)$$

More uncertainty, more entropy!

Information Theory interpretation: $H(Y)$ is the expected number of bits needed to encode a randomly drawn value of $Y$ (under most efficient code)

Information gain

Advantage of attribute – decrease in uncertainty

- Entropy of $Y$ before you split
- Entropy after split
  - Weight by probability of following each branch, i.e., normalized number of records

$$H(Y \mid X) = - \sum_{j=1}^{n} P(X = x_j) \sum_{i=1}^{k} P(Y = y_i \mid X = x_j) \log_2 P(Y = y_i \mid X = x_j)$$

Information gain is difference

$$IG(X) = H(Y) - H(Y \mid X)$$
Learning decision trees

- Start from empty decision tree
- Split on **next best attribute (feature)**
  - Use, for example, information gain to select attribute
  - Split on $\arg \max_i IG(X_i) = \arg \max_i H(Y) - H(Y | X_i)$
- Recurse

Suppose we want to predict MPG

Look at all the information gains...
A Decision Stump

mpg values: bad good

root

22 18

pchange = 0.001

cylinders = 3
cylinders = 4
cylinders = 5
cylinders = 6
cylinders = 8

0 0
4 17
1 0
8 0
9 1

Predict bad
Predict good
Predict bad
Predict bad
Predict bad

Base Case One

Don’t split a node if all matching records have the same output value
Base Case Two:

Don't split a node if none of the attributes can create multiple non-empty children.

Base Case Two:

No attributes can distinguish.
Base Cases

- Base Case One: If all records in current data subset have the same output then don’t recurse
- Base Case Two: If all records have exactly the same set of input attributes then don’t recurse

Base Cases: An idea

- Base Case One: If all records in current data subset have the same output then don’t recurse
- Base Case Two: If all records have exactly the same set of input attributes then don’t recurse

Proposed Base Case 3:
If all attributes have zero information gain then don’t recurse

•Is this a good idea?
The problem with Base Case 3:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Y = A XOR B

The information gains:

<table>
<thead>
<tr>
<th>Input</th>
<th>Value</th>
<th>Distribution</th>
<th>Info Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>Red</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Red</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>Red</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Red</td>
<td>0</td>
</tr>
</tbody>
</table>

The resulting bad decision tree:

y values: 0 1

If we omit Base Case 3:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

y = a XOR b

The resulting decision tree:

y values: 0 1

Predict 0
Basic Decision Tree Building
Summarized

BuildTree(DataSet, Output)

- If all output values are the same in DataSet, return a leaf node that says "predict this unique output"
- If all input values are the same, return a leaf node that says "predict the majority output"
- Else find attribute X with highest Info Gain
  - Suppose X has n_x distinct values (i.e. X has arity n_x).
    - Create and return a non-leaf node with n_x children.
    - The i'th child should be built by calling
      BuildTree(DS_i, Output)
      Where DS_i built consists of all those records in DataSet for which X = i'th distinct value of X.

MPG Test set error

<table>
<thead>
<tr>
<th></th>
<th>Num Errors</th>
<th>Set Size</th>
<th>Percent Wrong</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training Set</td>
<td>1</td>
<td>40</td>
<td>2.50</td>
</tr>
<tr>
<td>Test Set</td>
<td>74</td>
<td>352</td>
<td>21.02</td>
</tr>
</tbody>
</table>

©Carlos Guestrin 2005-2013
The test set error is much worse than the training set error…

...why?
Decision trees will overfit

- Standard decision trees have no learning bias
  - Training set error is always zero!
    - (If there is no label noise)
  - Lots of variance
  - Will definitely overfit!!
  - Must bias towards simpler trees

- Many strategies for picking simpler trees:
  - Fixed depth
  - Fixed number of leaves
  - Or something smarter…
A chi-square test

Suppose that MPG was completely uncorrelated with maker.
What is the chance we’d have seen data of at least this apparent level of association anyway?

By using a particular kind of chi-square test, the answer is 7.2%

(Such simple hypothesis tests are very easy to compute, unfortunately, not enough time to cover in the lecture, but see readings...)
Using Chi-squared to avoid overfitting

- Build the full decision tree as before
- But when you can grow it no more, start to prune:
  - Beginning at the bottom of the tree, delete splits in which $p_{\text{chance}} > \text{MaxPchance}$
  - Continue working your way up until there are no more prunable nodes

$\text{MaxPchance}$ is a magic parameter you must specify to the decision tree, indicating your willingness to risk fitting noise

Pruning example

- With $\text{MaxPchance} = 0.1$, you will see the following MPG decision tree:

Note the improved test set accuracy compared with the unpruned tree

<table>
<thead>
<tr>
<th>Num Errors</th>
<th>Set Size</th>
<th>Percent Wrong</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training Set</td>
<td>5</td>
<td>40</td>
</tr>
<tr>
<td>Test Set</td>
<td>56</td>
<td>352</td>
</tr>
</tbody>
</table>
MaxPchance

- Technical note MaxPchance is a regularization parameter that helps us bias towards simpler models.

Expected True Error

- High Bias
- Increasing MaxPchance
- Decreasing MaxPchance
- High Variance

Real-Valued inputs

- What should we do if some of the inputs are real-valued?

<table>
<thead>
<tr>
<th>mpg</th>
<th>cylinders</th>
<th>displacement</th>
<th>horsepower</th>
<th>weight</th>
<th>acceleration</th>
<th>modelyear</th>
<th>maker</th>
</tr>
</thead>
<tbody>
<tr>
<td>good</td>
<td>4</td>
<td>90</td>
<td>75</td>
<td>2265</td>
<td>18.2</td>
<td>77 asia</td>
<td></td>
</tr>
<tr>
<td>bad</td>
<td>6</td>
<td>106</td>
<td>90</td>
<td>2496</td>
<td>16</td>
<td>70 america</td>
<td></td>
</tr>
<tr>
<td>bad</td>
<td>4</td>
<td>121</td>
<td>110</td>
<td>2600</td>
<td>12.8</td>
<td>77 europe</td>
<td></td>
</tr>
<tr>
<td>bad</td>
<td>6</td>
<td>350</td>
<td>175</td>
<td>4100</td>
<td>13</td>
<td>73 america</td>
<td></td>
</tr>
<tr>
<td>bad</td>
<td>4</td>
<td>108</td>
<td>95</td>
<td>3100</td>
<td>16.5</td>
<td>74 america</td>
<td></td>
</tr>
<tr>
<td>bad</td>
<td>4</td>
<td>113</td>
<td>95</td>
<td>2228</td>
<td>16</td>
<td>73 asia</td>
<td></td>
</tr>
<tr>
<td>bad</td>
<td>8</td>
<td>302</td>
<td>139</td>
<td>3570</td>
<td>12.8</td>
<td>78 america</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>good</td>
<td>4</td>
<td>120</td>
<td>79</td>
<td>2625</td>
<td>18.6</td>
<td>80 america</td>
<td></td>
</tr>
<tr>
<td>bad</td>
<td>8</td>
<td>405</td>
<td>225</td>
<td>4425</td>
<td>10</td>
<td>70 america</td>
<td></td>
</tr>
<tr>
<td>good</td>
<td>4</td>
<td>107</td>
<td>85</td>
<td>2644</td>
<td>15.5</td>
<td>76 europe</td>
<td></td>
</tr>
<tr>
<td>bad</td>
<td>5</td>
<td>131</td>
<td>102</td>
<td>2850</td>
<td>15.9</td>
<td>78 europe</td>
<td></td>
</tr>
</tbody>
</table>

Infinite number of possible split values!!!

Finite dataset, only finite number of relevant splits!

Idea One: Branch on each possible real value
“One branch for each numeric value” idea:

Hopeless: with such high branching factor will shatter the dataset and overfit

Threshold splits

- Binary tree, split on attribute $X_i$
  - One branch: $X_i < t$
  - Other branch: $X_i \geq t$
Choosing threshold split

- Binary tree, split on attribute $X_i$
  - One branch: $X_i < t$
  - Other branch: $X_i \geq t$
- Search through possible values of $t$
  - Seems hard!!!
- But only finite number of $t$'s are important
  - Sort data according to $X_i$ into $\{x_1, \ldots, x_m\}$
  - Consider split points of the form $x_a + (x_{a+1} - x_a)/2$

A better idea: thresholded splits

- Suppose $X_i$ is real valued
- Define $IG(Y|X_i:t)$ as $H(Y) - H(Y|X_i:t)$
- Define $H(Y|X_i:t) = H(Y|X_i < t) P(X_i < t) + H(Y|X_i \geq t) P(X_i \geq t)$
  - $IG(Y|X_i:t)$ is the information gain for predicting $Y$ if all you know is whether $X_i$ is greater than or less than $t$
- Then define $IG^*(Y|X_i) = \max_t IG(Y|X_i:t)$
- For each real-valued attribute, use $IG^*(Y|X_i)$ for assessing its suitability as a split
- Note, may split on an attribute multiple times, with different thresholds
Example with MPG

Information gains using the training set (40 records)

<table>
<thead>
<tr>
<th>Input</th>
<th>Value</th>
<th>Distribution</th>
<th>Info Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>cylinders</td>
<td>&lt; 5</td>
<td>blue</td>
<td>0.48268</td>
</tr>
<tr>
<td></td>
<td>&gt;= 5</td>
<td>red</td>
<td></td>
</tr>
<tr>
<td>displacement</td>
<td>&lt; 106</td>
<td>blue</td>
<td>0.42605</td>
</tr>
<tr>
<td></td>
<td>&gt;= 106</td>
<td>red</td>
<td></td>
</tr>
<tr>
<td>horsepower</td>
<td>&lt; 94</td>
<td>blue</td>
<td>0.48268</td>
</tr>
<tr>
<td></td>
<td>&gt;= 94</td>
<td>red</td>
<td></td>
</tr>
<tr>
<td>weight</td>
<td>&lt; 2709</td>
<td>blue</td>
<td>0.279471</td>
</tr>
<tr>
<td></td>
<td>&gt;= 2709</td>
<td>red</td>
<td></td>
</tr>
<tr>
<td>acceleration</td>
<td>&lt; 19.2</td>
<td>blue</td>
<td>0.159862</td>
</tr>
<tr>
<td></td>
<td>&gt;= 19.2</td>
<td>red</td>
<td></td>
</tr>
<tr>
<td>modelyear</td>
<td>&lt; 81</td>
<td>blue</td>
<td>0.318103</td>
</tr>
<tr>
<td></td>
<td>&gt;= 81</td>
<td>red</td>
<td></td>
</tr>
<tr>
<td>maker</td>
<td>america</td>
<td>blue</td>
<td>0.0437265</td>
</tr>
<tr>
<td></td>
<td>asia</td>
<td>red</td>
<td></td>
</tr>
<tr>
<td></td>
<td>europe</td>
<td>blue</td>
<td></td>
</tr>
</tbody>
</table>

Example tree using reals
What you need to know about decision trees

- Decision trees are one of the most popular data mining tools
  - Easy to understand
  - Easy to implement
  - Easy to use
  - Computationally cheap (to solve heuristically)
- Information gain to select attributes (ID3, C4.5,...)
- Presented for classification, can be used for regression and density estimation too
- Decision trees will overfit!!!
  - Zero bias classifier! Lots of variance
  - Must use tricks to find “simple trees”, e.g.,
    - Fixed depth/Early stopping
    - Pruning
    - Hypothesis testing

Acknowledgements

- Some of the material in the decision trees presentation is courtesy of Andrew Moore, from his excellent collection of ML tutorials:
  - [http://www.cs.cmu.edu/~awm/tutorials](http://www.cs.cmu.edu/~awm/tutorials)