A Decision Stump

mpg values: bad good

root
22 18
pvalue = 0.001

Split on
One feature

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
Recursion Step

Take the Original Dataset.
And partition it according to the value of the attribute we split on.

Build tree from These examples.

Build tree from These examples.

Build tree from These examples.

Build tree from These examples.

©Carlos Guestrin 2005-2013
Second level of tree

Recurisely build a tree from the seven records in which there are four cylinders and the maker was based in Asia

(Similar recursion in the other cases)

The final tree
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 on **subset of data** consistent with each leaf

Choosing a good attribute

<table>
<thead>
<tr>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$Y$</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>

©Carlos Guestrin 2005-2013
Measuring uncertainty

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

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

\[
\begin{align*}
P(Y=A) &= 1/4 \\
P(Y=B) &= 1/4 \\
P(Y=C) &= 1/4 \\
P(Y=D) &= 1/4
\end{align*}
\]

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
  \[ H(Y) = -\sum_{i=1}^{n} P(Y = y_i) \log_2 P(Y = y_i) \]
- Entropy after split
  - Weight by probability of following each branch, i.e., normalized number of records
  \[ H(Y | x_i) = \sum_{j=1}^{n} P(x = x_j) \sum_{i=1}^{m} P(Y = y_i | X = x_j) \log_2 P(Y = y_i | X = x_j) \]
- Information gain is difference
  \[ IG(X_i) = H(Y) - H(Y | X_i) \]

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

When do I stop?
1. when information gain is small?
2. entropy \( 0 \) in leaf, correct class
3. nothing to split on (used all features)
Suppose we want to predict MPG

Look at all the information gains...

A Decision Stump
**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:

The resulting bad decision tree:

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

Predict 0
If we omit Base Case 3:

\[
y = a \text{ XOR } b
\]

The resulting decision tree:

The low info gain is not a good stopping criterion.

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.
The test set error is much worse than the training set error...

...why?
## Decision trees & Learning Bias

### Suppose "no label noise":

\[
\begin{align*}
&X', Y' = X, Y \\
&X' = X^2, \text{ but } Y' \neq Y^2,
\end{align*}
\]

What can we say about simple decision trees:

\[\implies \phi \text{ train error} \implies \text{can lead to overfitting} \implies \text{"bias"} \implies \text{variance high}\]

### Decision trees will overfit

- Standard decision trees are 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: 1, 2, 3, ...
  - Fixed number of leaves
  - Or something smarter…

©Carlos Guestrin 2005-2013
Consider this split

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?
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 you way up until there are no more prunable nodes

MaxPchance is a magic parameter you must specify to the decision tree, indicating your willingness to risk fitting noise.
Pruning example

- With 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.

Increasing MaxPchance leads to a decrease in expected true error by cross validation, moving from high bias to 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>91</td>
<td>2265</td>
<td>18.2</td>
<td>71</td>
<td>asia</td>
<td>good</td>
</tr>
<tr>
<td>bad</td>
<td>6</td>
<td>90</td>
<td>2648</td>
<td>15</td>
<td>70</td>
<td>america</td>
<td>bad</td>
</tr>
<tr>
<td>bad</td>
<td>4</td>
<td>121</td>
<td>2860</td>
<td>12.5</td>
<td>74</td>
<td>europe</td>
<td>bad</td>
</tr>
<tr>
<td>bad</td>
<td>8</td>
<td>350</td>
<td>4100</td>
<td>13</td>
<td>73</td>
<td>america</td>
<td>bad</td>
</tr>
<tr>
<td>bad</td>
<td>6</td>
<td>106</td>
<td>3100</td>
<td>16.5</td>
<td>74</td>
<td>america</td>
<td>bad</td>
</tr>
<tr>
<td>bad</td>
<td>4</td>
<td>108</td>
<td>2379</td>
<td>16.5</td>
<td>73</td>
<td>asia</td>
<td>bad</td>
</tr>
<tr>
<td>bad</td>
<td>4</td>
<td>113</td>
<td>2220</td>
<td>14</td>
<td>71</td>
<td>asia</td>
<td>bad</td>
</tr>
<tr>
<td>bad</td>
<td>8</td>
<td>302</td>
<td>3570</td>
<td>12.8</td>
<td>78</td>
<td>america</td>
<td>bad</td>
</tr>
<tr>
<td>good</td>
<td>4</td>
<td>120</td>
<td>2625</td>
<td>18.6</td>
<td>82</td>
<td>america</td>
<td>good</td>
</tr>
<tr>
<td>bad</td>
<td>8</td>
<td>465</td>
<td>4425</td>
<td>10</td>
<td>70</td>
<td>america</td>
<td>bad</td>
</tr>
<tr>
<td>good</td>
<td>4</td>
<td>107</td>
<td>2464</td>
<td>16.5</td>
<td>76</td>
<td>europe</td>
<td>good</td>
</tr>
<tr>
<td>bad</td>
<td>5</td>
<td>131</td>
<td>2830</td>
<td>15.9</td>
<td>78</td>
<td>europe</td>
<td>bad</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 $X$ into $\{x_1, \ldots, x_m\}$
  - Consider split points of the form $x_i + (x_{i+1} - x_i)/2$
A better idea: thresholded splits

- Suppose $X$ is real valued
- Define $IG(Y|X:t)$ as $H(Y) - H(Y|X:t)$
- Define $H(Y|X:t) = H(Y|X < t) \frac{P(X < t)}{P(X)} + H(Y|X \geq t) \frac{P(X \geq t)}{P(X)}$

$IG(Y|X:t)$ is the information gain for predicting $Y$ if all you know is whether $X$ is greater than or less than $t$

- Then define $IG^*(Y|X) = \max_t IG(Y|X:t)$
- For each real-valued attribute, use $IG^*(Y|X)$ for assessing its suitability as a split

- Note, may split on an attribute multiple times, with different thresholds

---

Example with MPG

<table>
<thead>
<tr>
<th>Input Attribute</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>0.48286</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>0.423055</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>0.48286</td>
</tr>
<tr>
<td></td>
<td>&gt;= 94</td>
<td>red</td>
<td></td>
</tr>
<tr>
<td>weight</td>
<td>&lt; 2700</td>
<td>blue</td>
<td>0.379471</td>
</tr>
<tr>
<td></td>
<td>&gt;= 2700</td>
<td>red</td>
<td></td>
</tr>
<tr>
<td>acceleration</td>
<td>&lt; 18.2</td>
<td>blue</td>
<td>0.159862</td>
</tr>
<tr>
<td></td>
<td>&gt;= 18.2</td>
<td>red</td>
<td></td>
</tr>
<tr>
<td>modelyear</td>
<td>&lt; 81</td>
<td>blue</td>
<td>0.319163</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>red</td>
<td></td>
</tr>
</tbody>
</table>

©Carlos Guestrin 2005-2013
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
Why not just use Linear Regression?

Using data to predict new data
Nearest neighbor

Given data points \((x^1, y^1) (x^2, y^2) \ldots (x^N, y^N)\), where we assume \(y^i = f(x^i)\) for some unknown function \(f\).

Given query point \(x^q\), your job is to predict

1. Find the closest \(x^i\) in our set of data points
   \[ j(nn) = \arg\min_j |x^j - x^q| \]

2. Predict \(\hat{y} = y^{(nn)}\)

Here’s a dataset with one input, one output and four data points.
1-Nearest Neighbor is an example of… 
Instance-based learning

A function approximator that has been around since about 1910.

To make a prediction, search database for similar datapoints, and fit with the local points.

Four things make a memory based learner:
- A distance metric
- How many nearby neighbors to look at?
- A weighting function (optional)
- How to fit with the local points?

1-Nearest Neighbor

Four things make a memory based learner:

1. A distance metric
   Euclidian (and many more)
2. How many nearby neighbors to look at?
   One
3. A weighting function (optional)
   Unused
4. How to fit with the local points?
   Just predict the same output as the nearest neighbor.
   \[ i = \arg \min_j \| y^i - x^j \| \]
   \[ \hat{y} = y^i \]
Multivariate 1-NN examples

Classification

Regression

Multivariate distance metrics

Suppose the input vectors \( x^1, x^2, \ldots, x^N \) are two dimensional:

\[ x^1 = (x^1_1, x^1_2), x^2 = (x^2_1, x^2_2), \ldots, x^N = (x^N_1, x^N_2). \]

One can draw the nearest-neighbor regions in input space.

\[ \text{Dist}(x_i, x_j) = (x_i^1 - x_j^1)^2 + (x_i^2 - x_j^2)^2 \]

The relative scalings in the distance metric affect region shapes.
Euclidean distance metric

\[ D(x, x') = \sqrt{\sum_i \sigma_i^2 (x_i - x'_i)^2} \]

Or equivalently,

\[ D(x, x') = \sqrt{(x - x')^T \sum (x - x')} \]

where

\[ \sum = \begin{bmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_N^2 \end{bmatrix} \]

Other Metrics…
- Mahalanobis, Rank-based, Correlation-based,…

Notable distance metrics (and their level sets)

- Scaled Euclidean (L_2)
- Mahalanobis
  - (here, \( \Sigma \) on the previous slide is not necessarily diagonal, but is symmetric)

\( \Sigma \) distances in \( L_1 \) are more important

\( \Sigma \) distances in this direction are more important

Learn \( \Sigma \) from data:
- distance metric learning

\( L_1 \) norm (absolute)

\( L_1 \) (max) norm
Consistency of 1-NN

- Consider an estimator \( f_n \) trained on \( n \) examples
  - e.g., 1-NN, neural nets, regression,...
- Estimator is consistent if true error goes to zero as amount of data increases
  - e.g., for no noise data, consistent if:
    \[
    \lim_{n \to \infty} MSE(f_n) = 0
    \]
- Regression is not consistent!
  - Representation bias
- 1-NN is consistent (under some mild fineprint)

What about variance???

1-NN overfits?

- too much flexibility, want smoother function
k-Nearest Neighbor

Four things make a memory based learner:

1. A distance metric
   - Euclidian (and many more)
2. How many nearby neighbors to look at?
   - $k$
3. A weighting function (optional)
   - Unused
4. How to fit with the local points?
   - Just predict the average output among the $k$ nearest neighbors.

K-nearest neighbor for function fitting smoothes away noise, but there are clear deficiencies.

What can we do about all the discontinuities that k-NN gives us?
Weighted k-NNs

- Neighbors are not all the same

\[ y \approx \frac{\pi_1 y_1 + \pi_2 y_2 + \pi_3 y_3}{\pi_1 + \pi_2 + \pi_3} \]

\[ \pi_i \text{ is some weight} \]

E.g., \[ \pi_i = \frac{1}{\|\mathbf{x}^j - \mathbf{x}^i\|^2} \]

Kernel regression

Four things make a memory based learner:

1. A distance metric
   Euclidian (and many more)

2. How many nearby neighbors to look at?
   All of them

3. A weighting function (optional)
   \[ \pi^i = \exp(-D(x^i, \text{query})^2 / \rho^2) \]
   Nearby points to the query are weighted strongly, far points weakly. The \( \rho \) parameter is the Kernel Width. Very important.

4. How to fit with the local points?
   Predict the weighted average of the outputs:
   \[ \text{predict} = \frac{\Sigma \pi^i y^i}{\Sigma \pi^i} \]
Weighting functions

\[ \pi' = \exp(-D(x^i, \text{query})^2 / \rho^2) \]

Typically optimize \( \rho \) using gradient descent (Our examples use Gaussian)

Increasing the kernel width \( \rho \) means further away points get an opportunity to influence you.

As \( \rho \to \infty \), the prediction tends to the global average.
Kernel regression on our test cases

Choosing a good $\rho$ is important. Not just for Kernel Regression, but for all the locally weighted learners we’re about to see.

Kernel regression can look bad

Time to try something more powerful…
Locally weighted regression

Kernel regression:
Take a very very conservative function approximator called AVERAGING. Locally weight it.

Locally weighted regression:
Take a conservative function approximator called LINEAR REGRESSION. Locally weight it.

Four things make a memory based learner:
- A distance metric
  Any
- How many nearby neighbors to look at?
  All of them
- A weighting function (optional)
  Kernels
  \[ \pi^i = \exp(-D(x^i, \text{query})^2 / \rho^2) \]
- How to fit with the local points?

General weighted regression:
\[ \hat{w}^q = \arg\min_w \sum_{k=1}^{N} \pi^k \bigl( y^k - w^T x^k \bigr)^2 \]
\[ \hat{y}^q = \hat{w}^q \cdot x^q \]
How LWR works

Linear regression
- Same parameters for all queries
\[ \hat{w} = (X^T X)^{-1} X^T Y \]

Locally weighted regression
- Solve weighted linear regression for each query
\[ \hat{w}^q = \left( \Pi X \right)^T \left( \Pi X \right)^{-1} \Pi Y \]

Another view of LWR

Kernel too wide - includes nonlinear region
Kernel just right
Kernel too narrow - excludes some of linear region

LWR on our test cases

\[ \rho = \frac{1}{16} \text{ of } x\text{-axis width.} \quad \rho = \frac{1}{32} \text{ of } x\text{-axis width.} \quad \rho = \frac{1}{8} \text{ of } x\text{-axis width.} \]

Locally weighted polynomial regression

Kernel Regression
Kernel width \( \rho \) at optimal level.
\[ \rho = \frac{1}{100} \text{ x-axis} \]

LW Linear Regression
Kernel width \( \rho \) at optimal level.
\[ \rho = \frac{1}{40} \text{ x-axis} \]

LW Quadratic Regression
Kernel width \( \rho \) at optimal level.
\[ \rho = \frac{1}{15} \text{ x-axis} \]

Local quadratic regression is easy: just add quadratic terms to the X matrix. As the regression degree increases, the kernel width can increase without introducing bias.
Curse of dimensionality for instance-based learning

- Must store and retrieve all data!
  - Most real work done during testing
  - For every test sample, must search through all dataset – very slow!
  - There are (sometimes) fast methods for dealing with large datasets
- Instance-based learning often poor with noisy or irrelevant features

Curse of the irrelevant feature
What you need to know about instance-based learning

- **k-NN**
  - Simplest learning algorithm
  - With sufficient data, very hard to beat “strawman” approach
  - Picking k?

- **Kernel regression**
  - Set k to n (number of data points) and optimize weights by gradient descent
  - Smoother than k-NN

- **Locally weighted regression**
  - Generalizes kernel regression, not just local average

- **Curse of dimensionality**
  - Must remember (very large) dataset for prediction
  - Irrelevant features often killers for instance-based approaches

Acknowledgment

- This lecture contains some material from Andrew Moore’s excellent collection of ML tutorials:
  - [http://www.cs.cmu.edu/~awm/tutorials](http://www.cs.cmu.edu/~awm/tutorials)