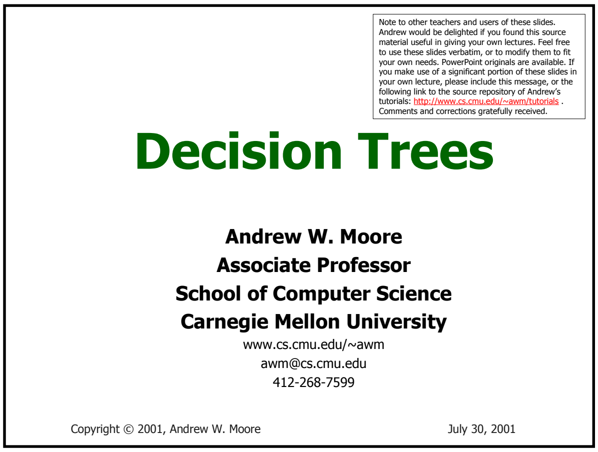
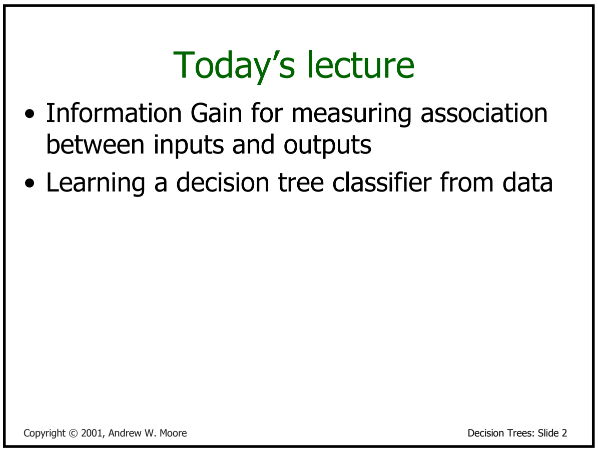


Decision Trees

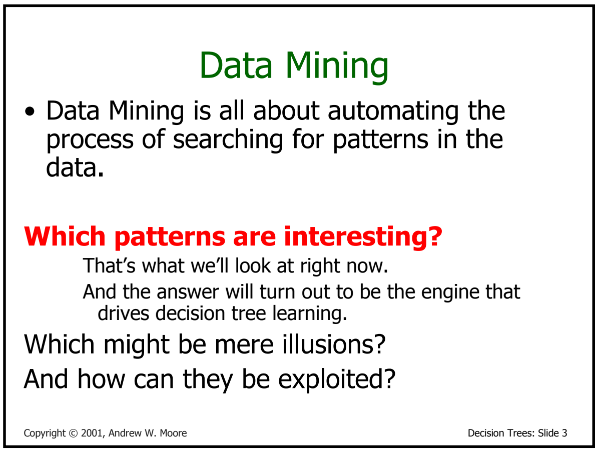
Andrew W. Moore
Associate Professor
School of Computer Science
Carnegie Mellon University
www.cs.cmu.edu/~awm
awm@cs.cmu.edu
412-268-7599

July 30, 2001



Today's lecture

- Information Gain for measuring association between inputs and outputs
- Learning a decision tree classifier from data



Data Mining

- Data Mining is all about automating the process of searching for patterns in the data.

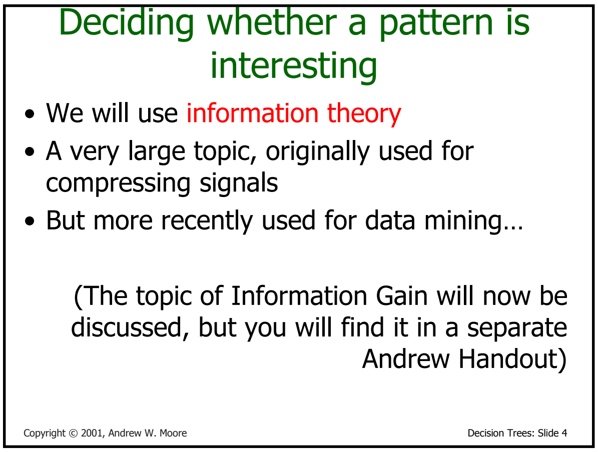
Which patterns are interesting?

That's what we'll look at right now.

And the answer will turn out to be the engine that drives decision tree learning.

Which might be mere illusions?

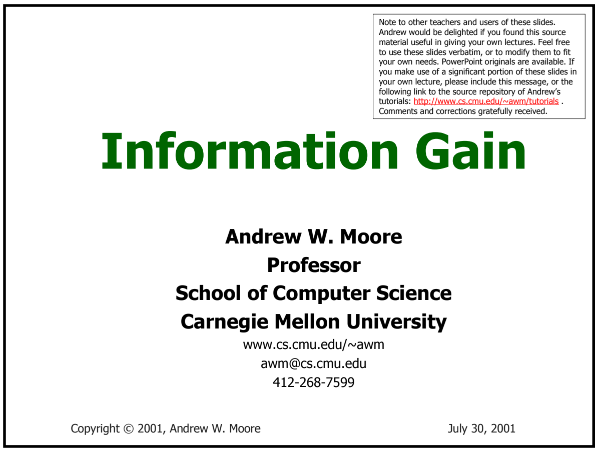
And how can they be exploited?



Deciding whether a pattern is interesting

- We will use information theory
- A very large topic, originally used for compressing signals
- But more recently used for data mining…

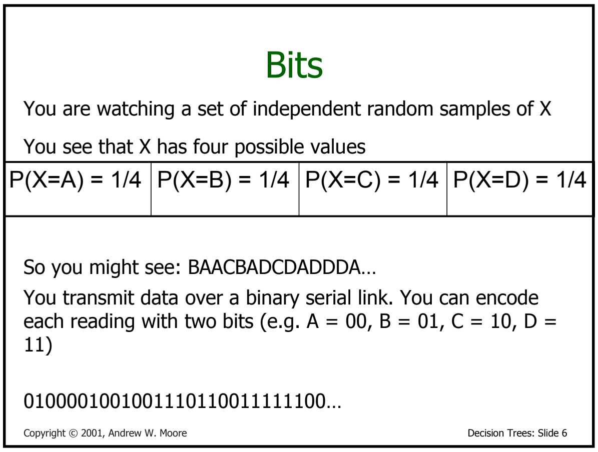
(The topic of Information Gain will now be discussed, but you will find it in a separate Andrew Handout)



Information Gain

Andrew W. Moore
Professor
School of Computer Science
Carnegie Mellon University
www.cs.cmu.edu/~awm
awm@cs.cmu.edu
412-268-7599

July 30, 2001



Bits

You are watching a set of independent random samples of X

You see that X has four possible values

P(X=A) = 1/4	P(X=B) = 1/4	P(X=C) = 1/4	P(X=D) = 1/4

So you might see: BAACBADCDADDDA…

You transmit data over a binary serial link. You can encode each reading with two bits (e.g. A = 00, B = 01, C = 10, D = 11)

0100001001001110110011111100…

Fewer Bits

Someone tells you that the probabilities are not equal

P(X=A) = 1/2	P(X=B) = 1/4	P(X=C) = 1/8	P(X=D) = 1/8

It's possible…

…to invent a coding for your transmission that only uses 1.75 bits on average per symbol. How?

Fewer Bits

Someone tells you that the probabilities are not equal

P(X=A) = 1/2	P(X=B) = 1/4	P(X=C) = 1/8	P(X=D) = 1/8

It's possible…

…to invent a coding for your transmission that only uses 1.75 bits on average per symbol. How?

A	0
B	10
C	110
D	111

(This is just one of several ways)

Fewer Bits

Suppose there are three equally likely values…

P(X=B) = 1/3	P(X=C) = 1/3	P(X=D) = 1/3

Here's a naïve coding, costing 2 bits per symbol

A	00
B	01
C	10

Can you think of a coding that would need only 1.6 bits per symbol on average?

In theory, it can in fact be done with 1.58496 bits per symbol.

General Case

Suppose X can have one of *m* values… $V_1, V_2, \dots V_m$

$P(X=V_1) = p_1$	$P(X=V_2) = p_2$	….	$P(X=V_m) = p_m$

What's the smallest possible number of bits, on average, per symbol, needed to transmit a stream of symbols drawn from X's distribution? It's

$$H(X) = -p_1 \log_2 p_1 - p_2 \log_2 p_2 - \dots - p_m \log_2 p_m$$

$$= -\sum_{j=1}^{m} p_j \log_2 p_j$$

H(X) = The entropy of X

- "High Entropy" means X is from a uniform (boring) distribution
- "Low Entropy" means X is from varied (peaks and valleys) distribution

General Case

Suppose X can have one of *m* values… $V_1, V_2, \dots V_m$

$P(X=V_1) = p_1$	$P(X=V_2) = p_2$	….	$P(X=V_m) = p_m$

What's the smallest possible number of symbol, tream m X's distri

$H(X)$ $p_2 - \dots$

$= -$ $p_j \log_2 p_j$

A histogram of the frequency distribution of values of X would be flat

A histogram of the frequency distribution of values of X would have many lows and one or two highs

H(X) = The entropy of X

- "High Entropy" means X is from a uniform (boring) distribution
- "Low Entropy" means X is from varied (peaks and valleys) distribution

General Case

Suppose X can have one of *m* values… $V_1, V_2, \dots V_m$

$P(X=V_1) = p_1$	$P(X=V_2) = p_2$	….	$P(X=V_m) = p_m$

What's the smallest possible number of symbol, tream m X's distri

$H(X)$ $p_2 - \dots$

A histogram of the frequency distribution of values of X would be flat

A histogram of the frequency distribution of values of X would have many lows and one or two highs

..and so the values sampled from it would be all over the place

..and so the values sampled from it would be more predictable

H(X) = The entropy of X

- "High Entropy" means X is from a uniform (boring) distribution
- "Low Entropy" means X is from varied (peaks and valleys) distribution

Entropy in a nut-shell

Low Entropy

High Entropy

Entropy in a nut-shell

Low Entropy

..the values (locations of soup) sampled entirely from within the soup bowl

High Entropy

..the values (locations of soup) unpredictable... almost uniformly sampled throughout our dining room

Specific Conditional Entropy H(Y|X=v)

Suppose I'm trying to predict output Y and I have input X

X = College Major

Y = Likes "Gladiator"

X	Y
Math	Yes
History	No
CS	Yes
Math	No
Math	No
CS	Yes
History	No
Math	Yes

Let's assume this reflects the true probabilities

E.G. From this data we estimate

- *P(LikeG = Yes) = 0.5*
- *P(Major = Math & LikeG = No) = 0.25*
- *P(Major = Math) = 0.5*
- *P(LikeG = Yes | Major = History) = 0*

Note:

- *H(X) = 1.5*
- *H(Y) = 1*

Specific Conditional Entropy H(Y|X=v)

X = College Major

Y = Likes "Gladiator"

X	Y
Math	Yes
History	No
CS	Yes
Math	No
Math	No
CS	Yes
History	No
Math	Yes

Definition of Specific Conditional Entropy:

H(Y |X=v) = **The entropy of** *Y* **among only those records in which** *X* **has value** *v*

Specific Conditional Entropy H(Y|X=v)

X = College Major

Y = Likes "Gladiator"

X	Y
Math	Yes
History	No
CS	Yes
Math	No
Math	No
CS	Yes
History	No
Math	Yes

Definition of Specific Conditional Entropy:

H(Y |X=v) = **The entropy of** *Y* **among only those records in which** *X* **has value** *v*

Example:

- *H(Y|X=Math) = 1*
- *H(Y|X=History) = 0*
- *H(Y|X=CS) = 0*

Conditional Entropy H(Y|X)

X = College Major

Y = Likes "Gladiator"

X	Y
Math	Yes
History	No
CS	Yes
Math	No
Math	No
CS	Yes
History	No
Math	Yes

Definition of Conditional Entropy:

H(Y |X) = The average specific conditional entropy of *Y*

= if you choose a record at random what will be the conditional entropy of *Y*, conditioned on that row's value of *X*

= Expected number of bits to transmit *Y* if both sides will know the value of *X*

$= \Sigma_j Prob(X=v_j)\ H(Y \mid X = v_j)$

Conditional Entropy

X = College Major

Y = Likes "Gladiator"

X	Y
Math	Yes
History	No
CS	Yes
Math	No
Math	No
CS	Yes
History	No
Math	Yes

Definition of Conditional Entropy:

$H(Y|X)$ = The average conditional entropy of Y

$= \Sigma_j Prob(X=v_j)\ H(Y \mid X = v_j)$

Example:

v_j	$Prob(X=v_j)$	$H(Y \mid X = v_j)$
Math	0.5	1
History	0.25	0
CS	0.25	0

$H(Y|X) = 0.5 * 1 + 0.25 * 0 + 0.25 * 0 = 0.5$

Information Gain

X = College Major

Y = Likes "Gladiator"

X	Y
Math	Yes
History	No
CS	Yes
Math	No
Math	No
CS	Yes
History	No
Math	Yes

Definition of Information Gain:

$IG(Y|X)$ = **I must transmit** Y**. How many bits on average would it save me if both ends of the line knew** X**?**

$IG(Y|X) = H(Y) - H(Y \mid X)$

Example:

- **H(Y) = 1**
- **H(Y|X) = 0.5**
- **Thus IG(Y|X) = 1 – 0.5 = 0.5**

Note to other teachers and users of these slides. Andrew would be delighted if you found this source material useful in giving your own lectures. Feel free to use these slides verbatim, or to modify them to fit your own needs. PowerPoint originals are available. If you make use of a significant portion of these slides in your own lecture, please include this message, or the following link to the source repository of Andrew's tutorials: http://www.cs.cmu.edu/~awm/tutorials . Comments and corrections gratefully received.

Back to Decision Trees

Andrew W. Moore
Associate Professor
School of Computer Science
Carnegie Mellon University
www.cs.cmu.edu/~awm
awm@cs.cmu.edu
412-268-7599

July 30, 2001

Learning Decision Trees

- A Decision Tree is a tree-structured plan of a set of attributes to test in order to predict the output.
- To decide which attribute should be tested first, simply find the one with the highest information gain.
- Then recurse…

A small dataset: Miles Per Gallon

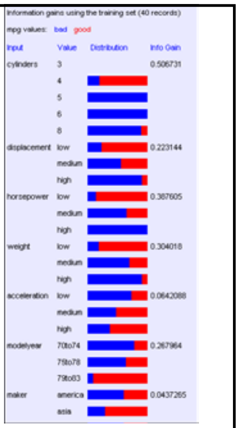
40 Records

mpg	cylinders	displacement	horsepower	weight	acceleration	modelyear	maker
good	4	low	low	low	high	75to78	asia
bad	6	medium	medium	medium	medium	70to74	america
bad	4	medium	medium	medium	low	75to78	europe
bad	8	high	high	high	low	70to74	america
bad	6	medium	medium	medium	medium	70to74	america
bad	4	low	medium	low	medium	70to74	asia
bad	4	low	medium	low	low	70to74	asia
bad	8	high	high	high	low	75to78	america
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
bad	8	high	high	high	low	70to74	america
good	8	high	medium	high	high	79to83	america
bad	8	high	high	high	low	75to78	america
good	4	low	low	low	low	79to83	america
bad	6	medium	medium	medium	high	75to78	america
good	4	medium	low	low	low	79to83	america
good	4	low	low	medium	high	79to83	america
bad	8	high	high	high	low	70to74	america
good	4	low	medium	low	medium	75to78	europe
bad	5	medium	medium	medium	medium	75to78	europe

From the UCI repository (thanks to Ross Quinlan)

Suppose we want to predict MPG.

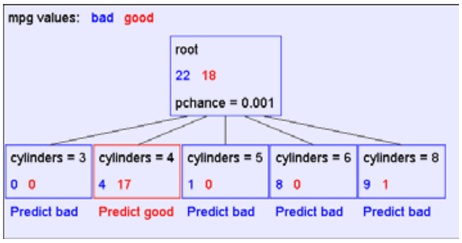
Look at all the information gains…


Information gains using the training set (40 records)

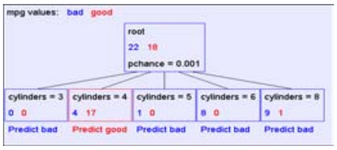
mpg values: bad good

Input	Value	Distribution	Info Gain
cylinders	3		0.506731
	4		
	5		
	6		
	8		
displacement	low		0.223144
	medium		
	high		
horsepower	low		0.387605
	medium		
	high		
weight	low		0.304018
	medium		
	high		
acceleration	low		0.0642088
	medium		
	high		
modelyear	70to74		0.267964
	75to78		
	79to83		
maker	america		0.0437265
	asia		

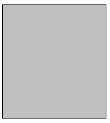
A Decision Stump



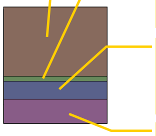
Recursion Step



Take the Original Dataset..



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



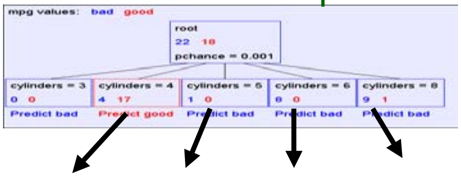
Records in which cylinders = 4

Records in which cylinders = 5

Records in which cylinders = 6

Records in which cylinders = 8

Recursion Step

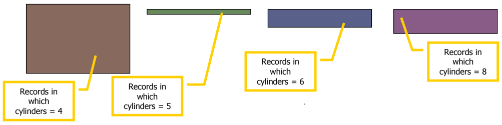


Build tree from These records..

Build tree from These records..

Build tree from These records..

Build tree from These records..



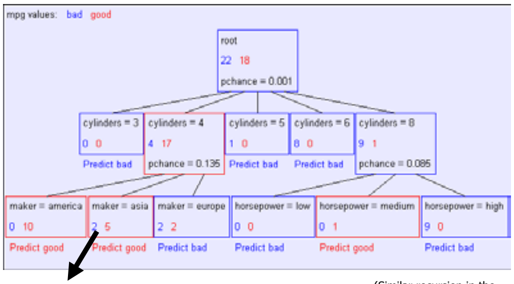
Records in which cylinders = 4

Records in which cylinders = 5

Records in which cylinders = 6

Records in which cylinders = 8

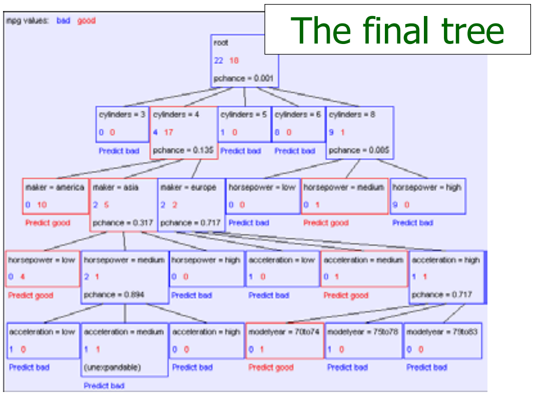
Second level of tree



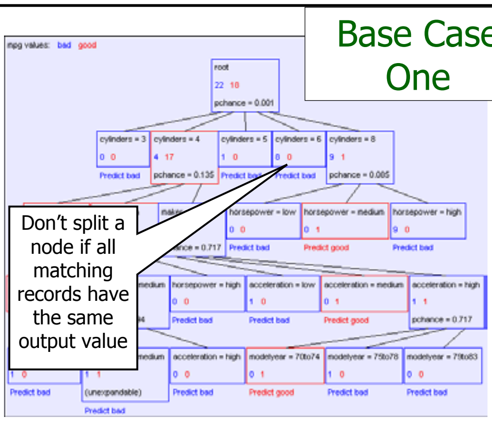
Recursively build a tree from the seven records in which there are four cylinders and the maker was based in Asia

(Similar recursion in the other cases)

The final tree

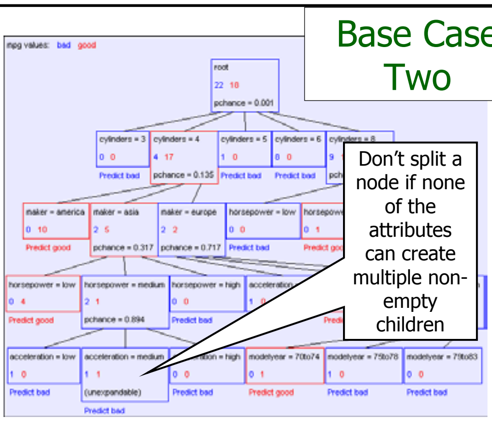


Base Case One



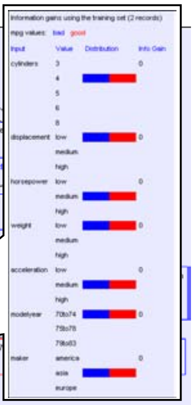
Don't split a node if all matching records have the same output value

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



Information gains using the training set (2 records)

mpg values: bad good

Input	Value	Distribution	Info Gain
cylinders	3		0
	4	🟦🟥	
	5		
	6		
	8		
displacement	low	🟦🟥	0
	medium		
	high		
horsepower	low		0
	medium	🟦🟥	
	high		
weight	low	🟦🟥	0
	medium		
	high		
acceleration	low		0
	medium	🟦🟥	
	high		
modelyear	70to74	🟦🟥	0
	75to78		
	79to83		
maker	america		0
	asia	🟦🟥	
	europe		

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

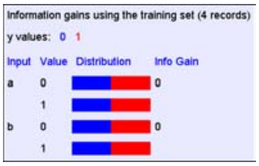
- *Is this a good idea?*

The problem with Base Case 3

a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

y = a XOR b

The information gains:

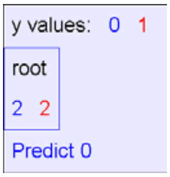


Information gains using the training set (4 records)

y values: 0 1

Input	Value	Distribution	Info Gain
a	0	🟦🟥	0
	1	🟦🟥	
b	0	🟦🟥	0
	1	🟦🟥	

The resulting decision tree:

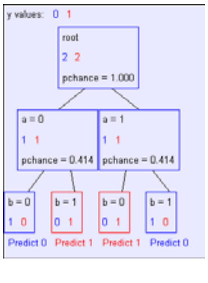


If we omit Base Case 3:

a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

y = a XOR b

The resulting decision tree:



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.

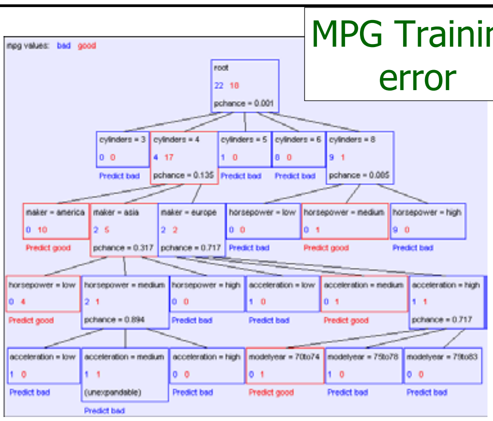
Training Set Error

- For each record, follow the decision tree to see what it would predict

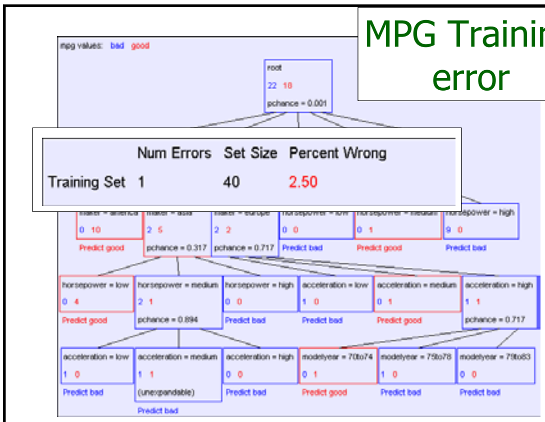
  For what number of records does the decision tree's prediction disagree with the true value in the database?

- This quantity is called the *training set error*. The smaller the better.

MPG Training error

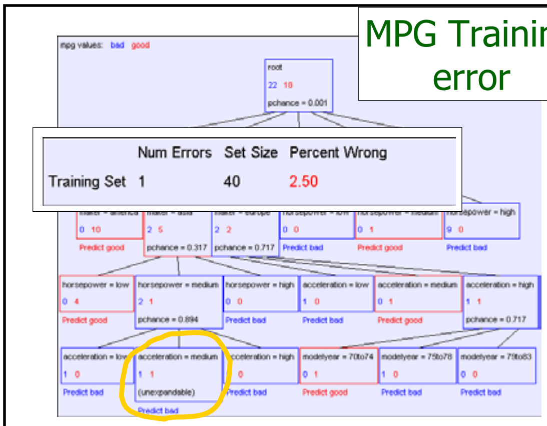


MPG Training error



	Num Errors	Set Size	Percent Wrong
Training Set	1	40	2.50

MPG Training error



	Num Errors	Set Size	Percent Wrong
Training Set	1	40	2.50

Stop and reflect: Why are we doing this learning anyway?

- It is not usually in order to predict the training data's output on data we have already seen.

Stop and reflect: Why are we doing this learning anyway?

- It is not usually in order to predict the training data’s output on data we have already seen.
- It is more commonly in order to predict the output value for future data we have not yet seen.

Stop and reflect: Why are we doing this learning anyway?

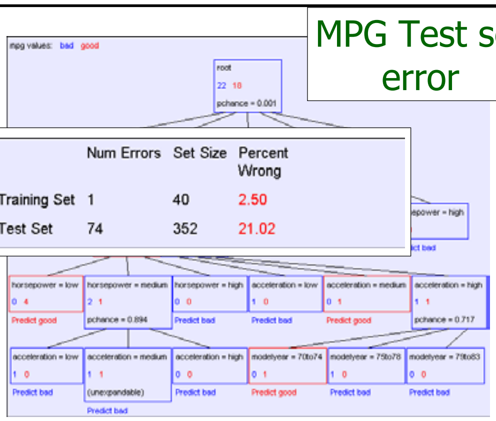
- It is not usually in order to predict the training data’s output on data we have already seen.
- It is more commonly in order to predict the output value for future data we have not yet seen.

Warning: A common data mining misperception is that the above two bullets are the only possible reasons for learning. There are at least a dozen others.

Test Set Error

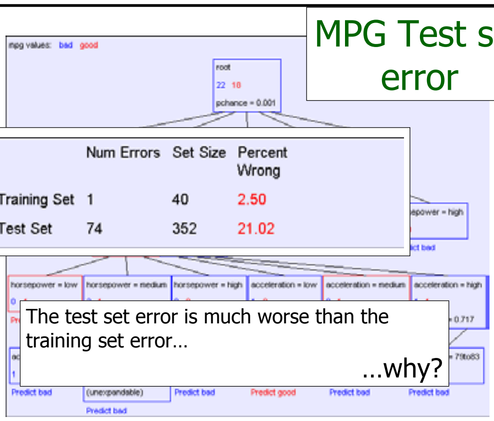
- Suppose we are forward thinking.
- We hide some data away when we learn the decision tree.
- But once learned, we see how well the tree predicts that data.
- This is a good simulation of what happens when we try to predict future data.
- And it is called Test Set Error.

MPG Test set error



	Num Errors	Set Size	Percent Wrong
Training Set	1	40	2.50
Test Set	74	352	21.02

MPG Test set error



	Num Errors	Set Size	Percent Wrong
Training Set	1	40	2.50
Test Set	74	352	21.02

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

…why?

An artificial example

- We’ll create a training dataset

Five inputs, all bits, are generated in all 32 possible combinations

Output y = copy of e, Except a random 25% of the records have y set to the opposite of e

32 records

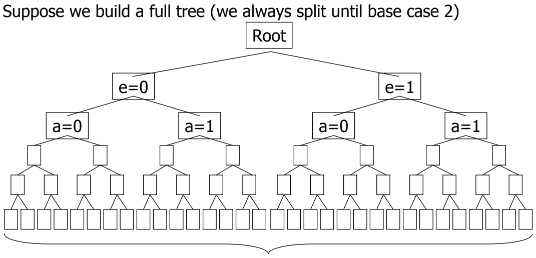
a	b	c	d	e	y
0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	1	0	0
0	0	0	1	1	1
0	0	1	0	0	1
:	:	:	:	:	:
1	1	1	1	1	1

In our artificial example

- Suppose someone generates a test set according to the same method.
- The test set is identical, except that some of the y's will be different.
- Some y's that were corrupted in the training set will be uncorrupted in the testing set.
- Some y's that were uncorrupted in the training set will be corrupted in the test set.

Building a tree with the artificial training set

- Suppose we build a full tree (we always split until base case 2)



25% of these leaf node labels will be corrupted

Training set error for our artificial tree

All the leaf nodes contain exactly one record and so…

- We would have a training set error of zero

Testing the tree with the test set

	1/4 of the tree nodes are corrupted	3/4 are fine
1/4 of the test set records are corrupted	1/16 of the test set will be correctly predicted for the wrong reasons	3/16 of the test set will be wrongly predicted because the test record is corrupted
3/4 are fine	3/16 of the test predictions will be wrong because the tree node is corrupted	9/16 of the test predictions will be fine

In total, we expect to be wrong on 3/8 of the test set predictions

What's this example shown us?

- This explains the discrepancy between training and test set error
- But more importantly… …it indicates there's something we should do about it if we want to predict well on future data.

Suppose we had less data

- Let's not look at the irrelevant bits

These bits are hidden

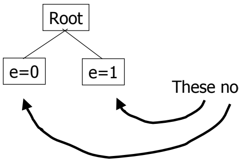
Output y = copy of e, except a random 25% of the records have y set to the opposite of e

32 records

a	b	c	d	e	y
0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	1	0	0
0	0	0	1	1	1
0	0	1	0	0	1
:	:	:	:	:	:
1	1	1	1	1	1

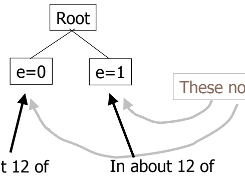
What decision tree would we learn now?

Without access to the irrelevant bits…



These nodes will be unexpandable

Without access to the irrelevant bits…



These nodes will be unexpandable

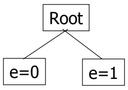
In about 12 of the 16 records in this node the output will be 0

So this will almost certainly predict 0

In about 12 of the 16 records in this node the output will be 1

So this will almost certainly predict 1

Without access to the irrelevant bits…



	almost certainly none of the tree nodes are corrupted	almost certainly all are fine
1/4 of the test set records are corrupted	n/a	1/4 of the test set will be wrongly predicted because the test record is corrupted
3/4 are fine	n/a	3/4 of the test predictions will be fine

In total, we expect to be wrong on only 1/4 of the test set predictions

Overfitting

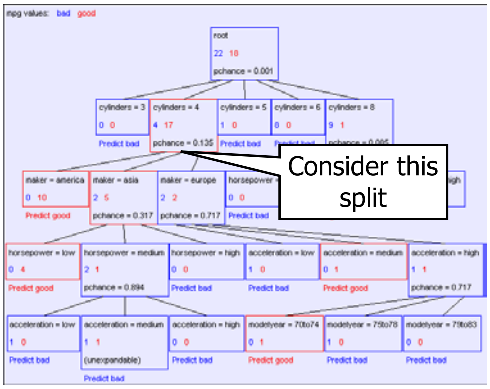
- Definition: If your machine learning algorithm fits noise (i.e. pays attention to parts of the data that are irrelevant) it is overfitting.
- Fact (theoretical and empirical): If your machine learning algorithm is overfitting then it may perform less well on test set data.

Avoiding overfitting

- Usually we do not know in advance which are the irrelevant variables
- …and it may depend on the context

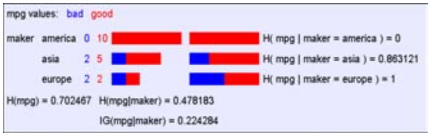
  For example, if y = a AND b then b is an irrelevant variable only in the portion of the tree in which a=0

But we can use simple statistics to warn us that we might be overfitting.



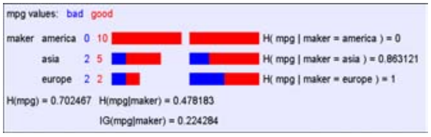
Consider this split

A chi-squared 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-squared 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-squared test, the answer is 13.5%.

What is a Chi-Square test?

- Google "chi square" for excellent explanations
- Takes into account "surprise" that a feature generates:

□ Σ((unsplit-number – split-number)^2/unsplit-number)

- Gives probability that rate you saw was generated by "luck of the draw"
- Does "likes-Matrix" predict "CS grad"?

	CS	Non CS
Likes Matrix	15972	145643
Hates Matrix	3	37

	CS	Non CS
Likes Matrix	21543	145643
Hates Matrix	3	173

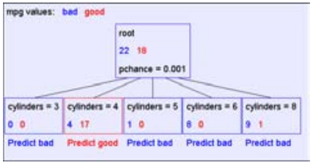
Using Chi-squared to avoid overfitting

- 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_{chance} > *MaxPchance.*
  - Continue working your way up until there are no more prunable nodes.

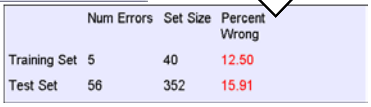
MaxPchance is a magic parameter you must specify to the decision tree, indicating your willingness to risk fitting noise.

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



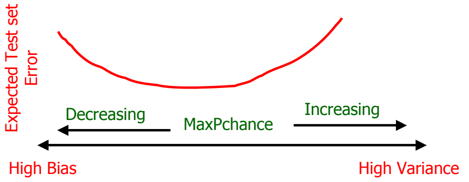
	Num Errors	Set Size	Percent Wrong
Training Set	5	40	12.50
Test Set	56	352	15.91

MaxPchance

- Good news: The decision tree can automatically adjust its pruning decisions according to the amount of apparent noise and data.
- Bad news: The user must come up with a good value of MaxPchance. (Note, Andrew usually uses 0.05, which is his favorite value for any magic parameter).
- Good news: But with extra work, the best MaxPchance value can be estimated automatically by a technique called cross-validation.

MaxPchance

- Technical note (dealt with in other lectures): MaxPchance is a regularization parameter.



The simplest tree

- Note that this pruning is heuristically trying to find

The simplest tree structure for which all within-leaf-node disagreements can be explained by chance

- This is not the same as saying "the simplest classification scheme for which…"
- Decision trees are biased to prefer classifiers that can be expressed as trees.

Expressiveness of Decision Trees

- Assume all inputs are Boolean and all outputs are Boolean.
- What is the class of Boolean functions that are possible to represent by decision trees?
- Answer: All Boolean functions.

Simple proof:

1. Take any Boolean function
2. Convert it into a truth table
3. Construct a decision tree in which each row of the truth table corresponds to one path through the decision tree.

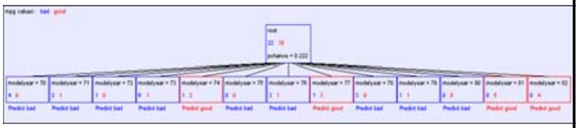
Real-Valued inputs

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

mpg	cylinders	displacemen	horsepower	weight	acceleration	modelyear	maker
good	4	97	75	2265	18.2	77	asia
bad	6	199	90	2648	15	70	america
bad	4	121	110	2600	12.8	77	europe
bad	8	350	175	4100	13	73	america
bad	6	198	95	3102	16.5	74	america
bad	4	108	94	2379	16.5	73	asia
bad	4	113	95	2228	14	71	asia
bad	8	302	139	3570	12.8	78	america
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
good	4	120	79	2625	18.6	82	america
bad	8	455	225	4425	10	70	america
good	4	107	86	2464	15.5	76	europe
bad	5	131	103	2830	15.9	78	europe

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 over fit

Note pchance is 0.222 in the above…if MaxPchance was 0.05 that would end up pruning away to a single root node.

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) P(X < t) + H(Y|X >= t) P(X >= t)

  - *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

Computational Issues

- You can compute IG*(Y|X) in time

$$R \log R + 2 R n_y$$

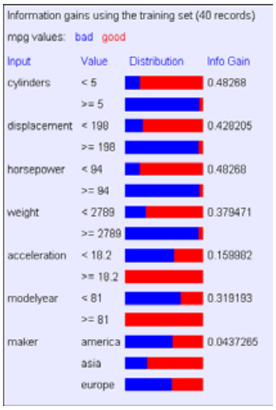
- Where

R is the number of records in the node under consideration

n_y is the arity (number of distinct values of) Y

How?

Sort records according to increasing values of X. Then create a 2xn_y contingency table corresponding to computation of IG(Y|X:x_{min}). Then iterate through the records, testing for each threshold between adjacent values of X, incrementally updating the contingency table as you go. For a minor additional speedup, only test between values of Y that differ.

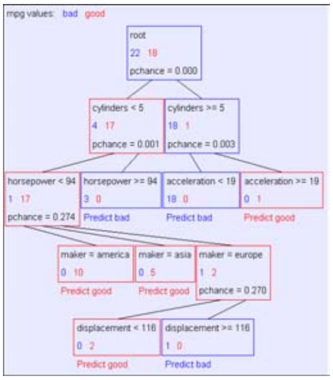


Information gains using the training set (40 records)

mpg values: bad good

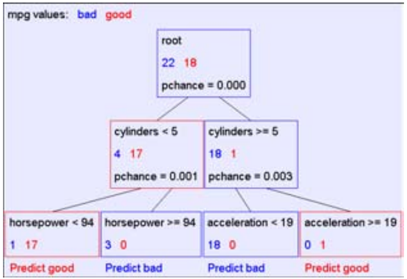
Input	Value	Distribution	Info Gain
cylinders	< 5	🟦🟥	0.48268
	>= 5	🟦🟥	
displacement	< 198	🟦🟥	0.428205
	>= 198	🟦🟥	
horsepower	< 94	🟦🟥	0.48268
	>= 94	🟦🟥	
weight	< 2789	🟦🟥	0.379471
	>= 2789	🟦🟥	
acceleration	< 18.2	🟦🟥	0.159982
	>= 18.2	🟥	
modelyear	< 81	🟦🟥	0.319193
	>= 81	🟥	
maker	america	🟦🟥	0.0437265
	asia	🟦🟥	
	europe	🟦🟥	

Example with MPG



Unpruned tree using reals

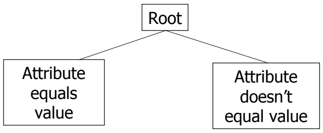
Pruned tree using reals



	Num Errors	Set Size	Percent Wrong
Training Set	1	40	2.50
Test Set	53	352	15.06

Binary categorical splits

- One of Andrew's favorite tricks
- Allow splits of the following form

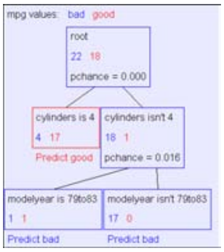


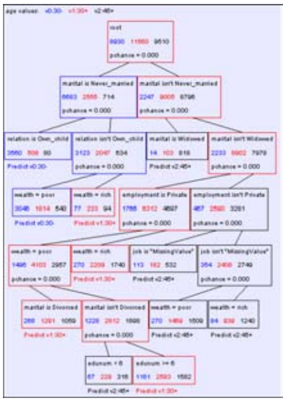
Root

Attribute equals value

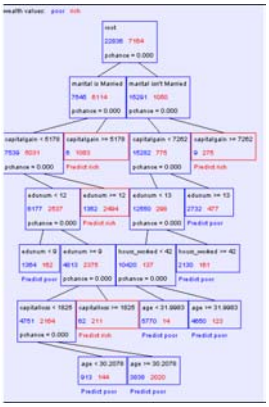
Attribute doesn't equal value

Example:



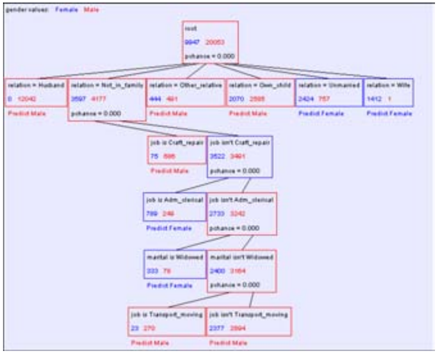


Predicting age from census



Predicting wealth from census

Predicting gender from census



Conclusions

- Decision trees are the single most popular data mining tool
  - Easy to understand
  - Easy to implement
  - Easy to use
  - Computationally cheap
- It's possible to get in trouble with overfitting
- They do classification: predict a categorical output from categorical and/or real inputs

What you should know

- What's information gain, and why we use it
- The recursive algorithm for building an unpruned decision tree
- What are training and test set errors
- Why test set errors can be bigger than training set
- Why pruning can reduce test set error
- How to exploit real-valued inputs

What we haven't discussed

- It's easy to have real-valued outputs too---these are called Regression Trees*
- Bayesian Decision Trees can take a different approach to preventing overfitting
- Computational complexity (straightforward and cheap) *
- Alternatives to Information Gain for splitting nodes
- How to choose MaxPchance automatically *
- The details of Chi-Squared testing *
- Boosting---a simple way to improve accuracy *

* = discussed in other Andrew lectures

For more information

- Two nice books
  - L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. Classification and Regression Trees. Wadsworth, Belmont, CA, 1984.
  - C4.5 : Programs for Machine Learning (Morgan Kaufmann Series in Machine Learning) by J. Ross Quinlan
- Dozens of nice papers, including
  - Learning Classification Trees, Wray Buntine, Statistics and Computation (1992), Vol 2, pages 63-73
  - Kearns and Mansour, On the Boosting Ability of Top-Down Decision Tree Learning Algorithms, STOC: ACM Symposium on Theory of Computing, 1996"
- Dozens of software implementations available on the web for free and commercially for prices ranging between $50 - $300,000

Discussion

- Instead of using information gain, why not choose the splitting attribute to be the one with the highest prediction accuracy?
- Instead of greedily, heuristically, building the tree, why not do a combinatorial search for the optimal tree?
- If you build a decision tree to predict wealth, and marital status, age and gender are chosen as attributes near the top of the tree, is it reasonable to conclude that those three inputs are the major causes of wealth?
- ..would it be reasonable to assume that attributes not mentioned in the tree are not causes of wealth?
- ..would it be reasonable to assume that attributes not mentioned in the tree are not correlated with wealth?
- What about multi-attribute splits?