Decision Trees

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

Decision Trees

Decision trees are powerful and popular tools for classification and prediction. The attractiveness of decision trees is due to the fact that, in contrast to neural networks, **decision trees represent rules**. Rules can readily be expressed so that humans can understand them or even directly used in a database access language like SQL so that records falling into a particular category may be retrieved.

In some applications, the accuracy of a classification or prediction is the only thing that matters. In such situations we do not necessarily care how or why the model works.

In other situations, the ability to explain the reason for a decision, is crucial. In marketing one has describe the customer segments to marketing professionals, so that they can utilize this knowledge in launching a successful marketing campaign. This domain experts must recognize and approve this discovered knowledge, and for this we need good descriptions.

There are a variety of algorithms for building decision trees that share the desirable quality of interpretability. A well known and frequently used over the years is C4.5 (or improved, but commercial version See5/C5.0).

What is a decision tree ?

Decision tree is a classifier in the form of a tree structure, where each node is either:

- **a leaf node** - indicates the value of the target attribute (class) of examples, or

- **a decision node** - specifies some test to be carried out on a single attribute-value, with one branch and sub-tree for each possible outcome of the test.

A decision tree can be used to classify an example by starting at the root of the tree and moving through it until a leaf node, which provides the classification of the instance.

Decision tree induction is a typical inductive approach to learn knowledge on classification. The key requirements to do mining with decision trees are:

Attribute-value description

- object or case must be expressible in terms of a fixed collection of properties or attributes. This means that we need to discretize continuous attributes, or this must have been provided in the algorithm.

Predefined classes (target attribute values)

- The categories to which examples are to be assigned must have been established beforehand (supervised data).

Discrete classes

- A case does or does not belong to a particular class, and there must be more cases than classes.

Sufficient data

- Usually hundreds or even thousands of training cases.

Decision tree types:

Decision tree has three other names:

- Classification tree analysis is a term used when the predicted outcome is the class to which the data belongs.

- Regression tree analysis is a term used when the predicted outcome can be considered a real number (e.g. the price of a house, or a patient's length of stay in a hospital).

- CART analysis is a term used to refer to both of the above procedures. The name CART is an acronym from the words Classification And Regression Trees, and was first introduced by Breiman et al.

Common formulas for tree construction:

Gini impurity

Used by the **CART** algorithm (Classification and Regression Trees). It is based on squared probabilities of membership for each target category in the node. It reaches its minimum (zero) when all cases in the node fall into a single target category.

Suppose y takes on values in {1, 2, ..., m}, and let f(i, j) = frequency of value j in node i. That is, f(i, j) is the proportion of records assigned to node i for which y = j.

$$I_G(i) = 1 - \sum_{j=1}^{m} f(i,j)^2 = \sum_{j \neq k} f(i,j) f(i,k)$$

Information gain

Used by the ID3, C4.5 and C5.0 tree generation algorithms. Information gain is based on the concept of entropy used in information theory.

$$I_E(i) = -\sum_{j=1}^{m} f(i,j) \log_2 f(i,j)$$

When to stop growing the tree?

Why not build a tree to maximum depth, so that all leaf nodes are either pure, or contain conflicting records?

Some algorithms, in fact, begin by building trees to their maximum depth. While such a tree can precisely predict all the instances in the training set, the problem with such a tree is that, more than likely, it has **overfit the data**.

Such a tree is **too specific** and will not find whatever general principles are at work.

Therefore **stopping rules** are used. They are usually based on several factors including maximum tree depth, minimum number of elements in a node considered for splitting, or its near equivalent, the minimum number of elements that must be in a new node.

Pruning Trees

After a data mining product grows a tree, an analyst must explore the model. Exploring the tree model, even one that is grown with stopping rules, may reveal nodes or subtrees that are **undesirable** because of **overfitting**, or may contain rules that the domain expert feels are inappropriate.

Pruning is a common technique used to **make a tree more general**. Pruning removes splits and the subtrees created by them. In some implementations, pruning is controlled by user configurable parameters that cause splits to be pruned because, for example, the computed difference between the resulting nodes falls below a threshold and is insignificant.

With such algorithms, users will want to experiment to see which pruning rule parameters result in a tree that predicts best on a test dataset. Algorithms that build trees to maximum depth will automatically invoke pruning. In some products users also have the ability to prune the tree interactively.

Here is a dataset

age	employme	education	edun	marital	…	job	relation	race	gender	hour	country	wealth
					…							
39	State_gov	Bachelors	13	Never_mar	…	Adm_cleri	Not_in_fan	White	Male	40	United_Sta	poor
51	Self_emp_	Bachelors	13	Married	…	Exec_man	Husband	White	Male	13	United_Sta	poor
39	Private	HS_grad	9	Divorced	…	Handlers_	Not_in_fan	White	Male	40	United_Sta	poor
54	Private	11th	7	Married	…	Handlers_	Husband	Black	Male	40	United_Sta	poor
28	Private	Bachelors	13	Married	…	Prof_speci	Wife	Black	Female	40	Cuba	poor
38	Private	Masters	14	Married	…	Exec_man	Wife	White	Female	40	United_Sta	poor
50	Private	9th	5	Married_sp	…	Other_serv	Not_in_fan	Black	Female	16	Jamaica	poor
52	Self_emp_	HS_grad	9	Married	…	Exec_man	Husband	White	Male	45	United_Sta	rich
31	Private	Masters	14	Never_mar	…	Prof_speci	Not_in_fan	White	Female	50	United_Sta	rich
42	Private	Bachelors	13	Married	…	Exec_man	Husband	White	Male	40	United_Sta	rich
37	Private	Some_coll	10	Married	…	Exec_man	Husband	Black	Male	80	United_Sta	rich
30	State_gov	Bachelors	13	Married	…	Prof_speci	Husband	Asian	Male	40	India	rich
24	Private	Bachelors	13	Never_mar	…	Adm_cleri	Own_child	White	Female	30	United_Sta	poor
33	Private	Assoc_ac	12	Never_mar	…	Sales	Not_in_fan	Black	Male	50	United_Sta	poor
41	Private	Assoc_voc	11	Married	…	Craft_repai	Husband	Asian	Male	40	*MissingV	rich
34	Private	7th_8th	4	Married	…	Transport_	Husband	Amer_India	Male	45	Mexico	poor
26	Self_emp_	HS_grad	9	Never_mar	…	Farming_fi	Own_child	White	Male	35	United_Sta	poor
33	Private	HS_grad	9	Never_mar	…	Machine_c	Unmarried	White	Male	40	United_Sta	poor
38	Private	11th	7	Married	…	Sales	Husband	White	Male	50	United_Sta	poor
44	Self_emp_	Masters	14	Divorced	…	Exec_man	Unmarried	White	Female	45	United_Sta	rich
41	Private	Doctorate	16	Married	…	Prof_speci	Husband	White	Male	60	United_Sta	rich
:	:	:	:	:	:	:	:	:	:	:	:	:

48,000 records, 16 attributes [Kohavi 1995]

About this dataset

- It is a tiny subset of the 1990 US Census.
- It is publicly available online from the UCI Machine Learning Datasets repository

Used Attributes

age	edunum	race	hours_worked
employment	marital	gender	country
taxweighting	job	capitalgain	wealth
education	relation	capitalloss	agegroup

This color = Real-valued This color = Symbol-valued

Successfully loaded a new dataset from the file \tadult.fds. It has 16 attributes and 48842 records.

What can you do with a dataset?

- Well, you can look at histograms...

Value	Frequency
Female	16192
Male	32650

Gender

Value	Frequency
Divorced	6633
Married_AF_spouse	37
Married	22379
Married_spouse_absent	628
Never_married	16117
Separated	1530
Widowed	1518

Marital Status

Contingency Tables

- A better name for a histogram:

  A One-dimensional Contingency Table

- Recipe for making a k-dimensional contingency table:

  1. Pick *k* attributes from your dataset. Call them $a_1, a_2, \ldots a_k$.
  2. For every possible combination of values, $a_1 = x_1, a_2 = x_2, \ldots a_k = x_k$, record how frequently that combination occurs

Fun fact: A database person would call this a "k-dimensional datacube"

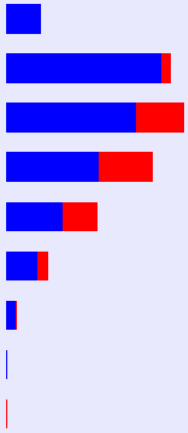
A 2-d Contingency Table

wealth values:		poor	rich
agegroup	10s	2507	3
	20s	11262	743
	30s	9468	3461
	40s	6738	3986
	50s	4110	2509
	60s	2245	809
	70s	668	147
	80s	115	16
	90s	42	13

- For each pair of values for attributes (agegroup,wealth) we can see how many records match.

A 2-d Contingency Table

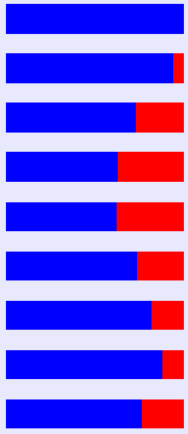
wealth values:		poor	rich
agegroup	10s	2507	3
	20s	11262	743
	30s	9468	3461
	40s	6738	3986
	50s	4110	2509
	60s	2245	809
	70s	668	147
	80s	115	16
	90s	42	13



- Easier to appreciate graphically

A 2-d Contingency Table

wealth values:		poor	rich
agegroup	10s	2507	3
	20s	11262	743
	30s	9468	3461
	40s	6738	3986
	50s	4110	2509
	60s	2245	809
	70s	668	147
	80s	115	16
	90s	42	13



- Easier to see “interesting” things if we stretch out the histogram bars

A bigger 2-d contingency table

job values:	Adm_clerical	Craft_repair		Farming_fishing			Machine_op_inspct			Priv_house_serv	Protective_serv		Tech_support	
MissingValue	Armed_Forces	Exec_managerial		Handlers_cleaners			Other_service			Prof_specialty		Sales		Transport_moving
marital	Divorced	270	1192	0	679	890	90	197	434	762	46	795	121	664	239	254
	Married_AF_spouse	5	6	0	4	3	1	1	1	5	0	4	1	5	0	1
	Married	928	1495	7	3818	3600	869	724	1469	1088	27	3182	583	2491	609	1489
	Married_spouse_absent	45	84	0	77	52	35	32	37	92	9	64	7	55	9	30
	Never_married	1242	2360	8	1301	1260	434	1029	872	2442	99	1849	237	1992	506	486
	Separated	97	224	0	160	126	23	63	123	275	21	145	23	146	48	56
	Widowed	222	250	0	73	155	38	26	86	259	40	133	11	151	35	39

Searching for High Info Gains

- Given something (e.g. wealth) you are trying to predict, it is easy to ask the computer to find which attribute has highest information gain for it.

wealth values:	poor	rich	
relation Husband	10870	8846	H(wealth | relation = Husband) = 0.992385
Not_in_family	11307	1276	H(wealth | relation = Not_in_family) = 0.473439
Other_relative	1454	52	H(wealth | relation = Other_relative) = 0.216617
Own_child	7470	111	H(wealth | relation = Own_child) = 0.110192
Unmarried	4816	309	H(wealth | relation = Unmarried) = 0.328606
Wife	1238	1093	H(wealth | relation = Wife) = 0.997207

H(wealth) = 0.793844 H(wealth|relation) = 0.628421

IG(wealth|relation) = 0.165423

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

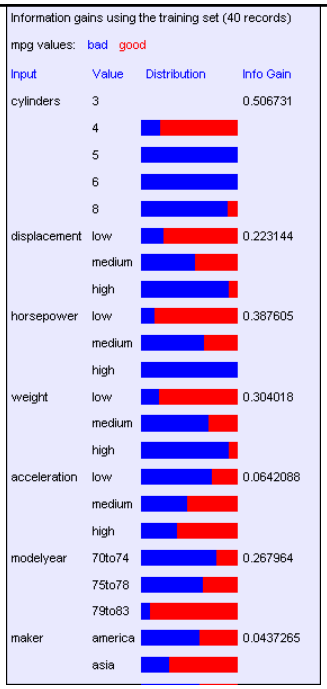
40 Records

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

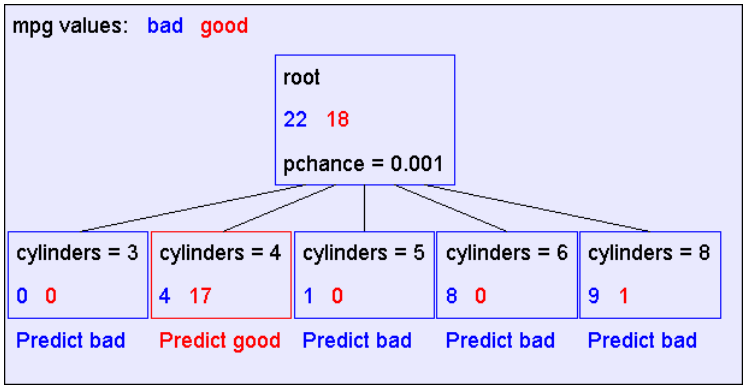
From the UCI repository (thanks to Ross Quinlan)

Suppose we want to predict MPG.

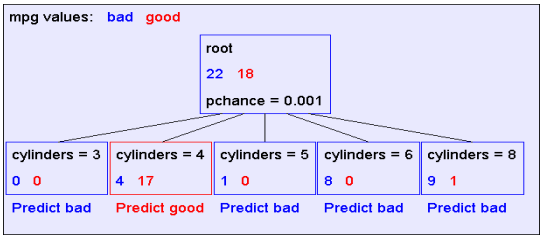
Look at all the information gains...



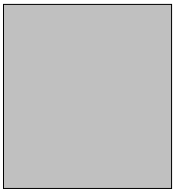
A Decision Stump



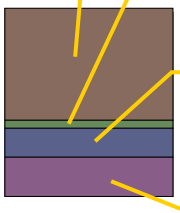
Recursion Step



Take the Original Dataset..



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



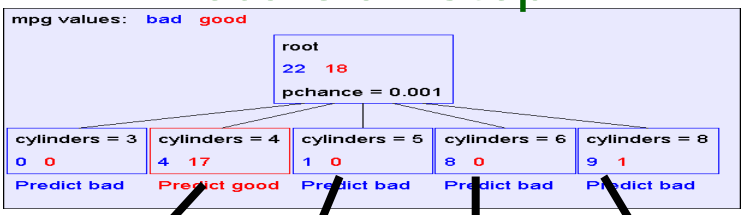
Records in which cylinders = 4

Records in which cylinders = 5

Records in which cylinders = 6

Records in which cylinders = 8

Recursion Step



Build tree from These records..

Records in which cylinders = 4

Build tree from These records..

Records in which cylinders = 5

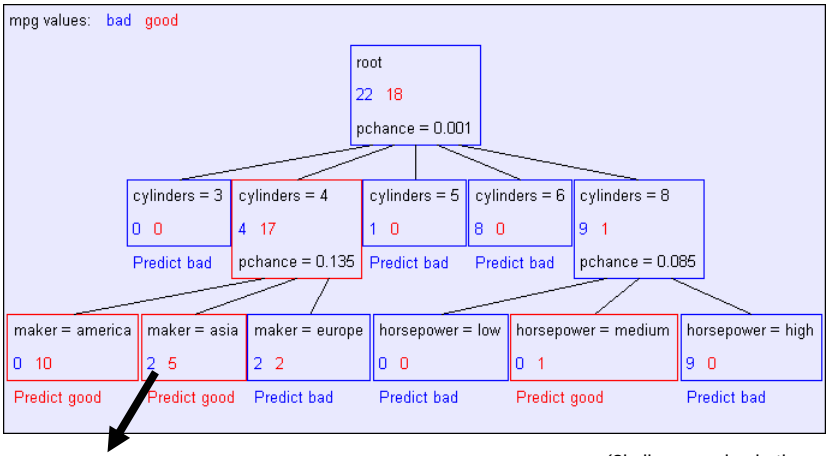
Build tree from These records..

Records in which cylinders = 6

Build tree from These records..

Records in which cylinders = 8

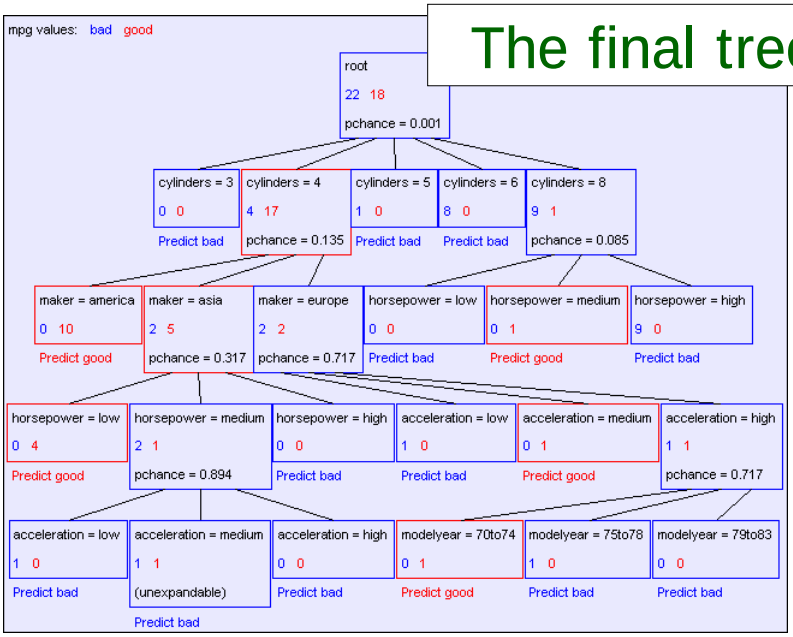
Second level of tree



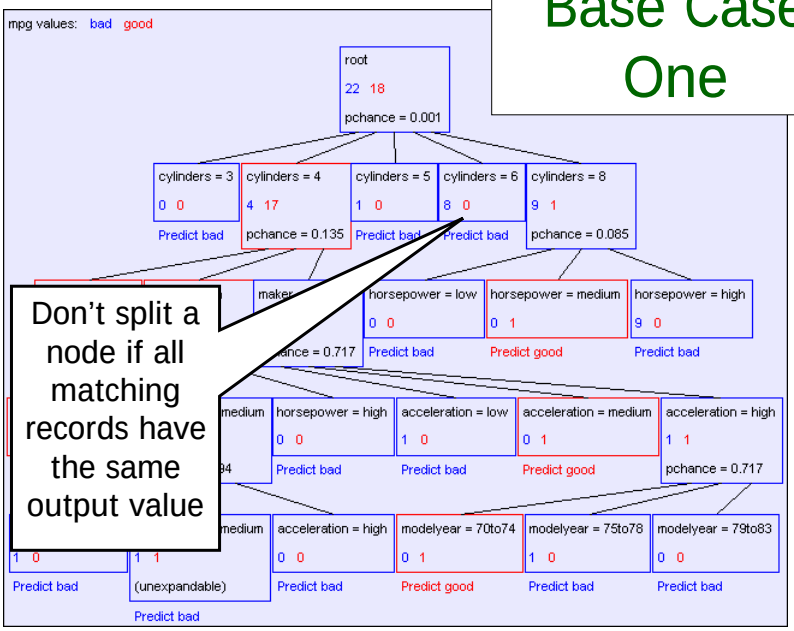
Recursively build a tree from the seven records in which there are four cylinders and the maker was based in Asia

(Similar recursion in the other cases)

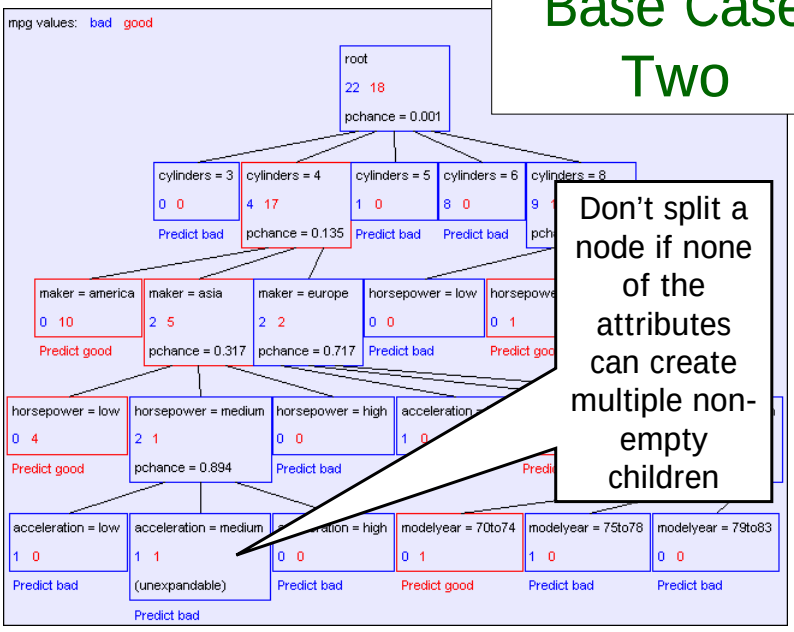
The final tree



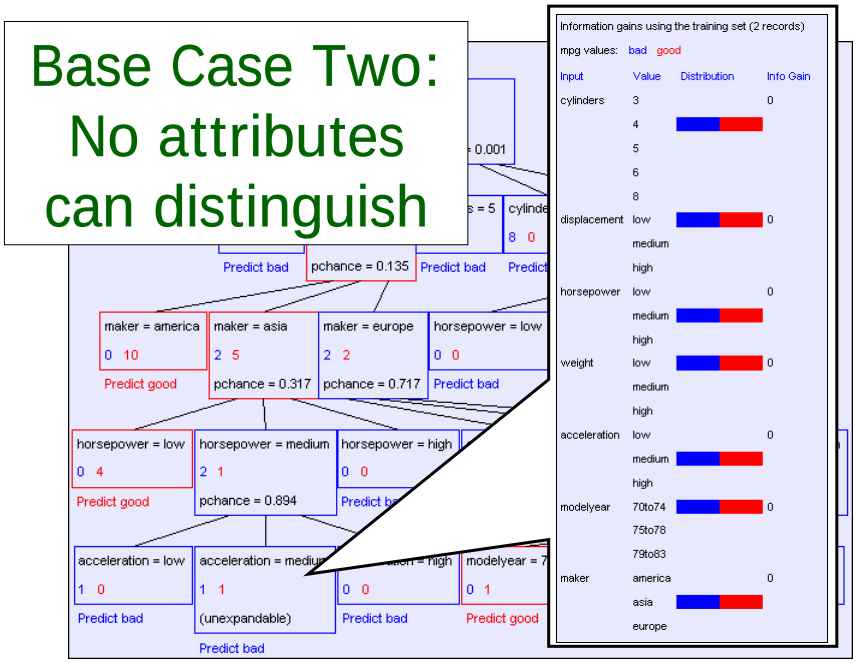
Base Case One



Base Case Two



Base Case Two: No attributes can distinguish

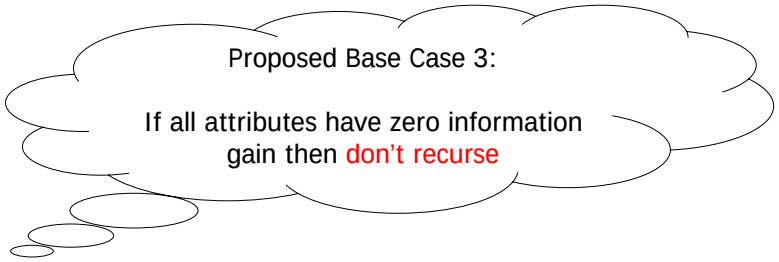


Base Cases

- 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



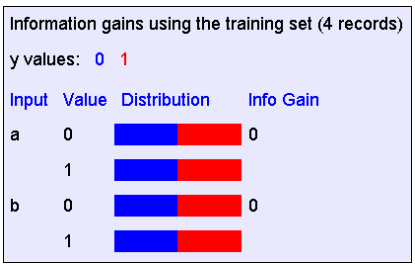
- *Is this a good idea?*

The problem with Base Case 3

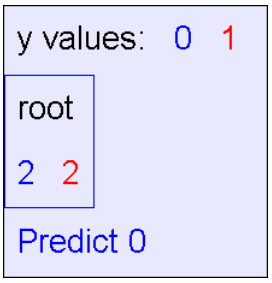
a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

y = a XOR b

The information gains:



The resulting decision tree:

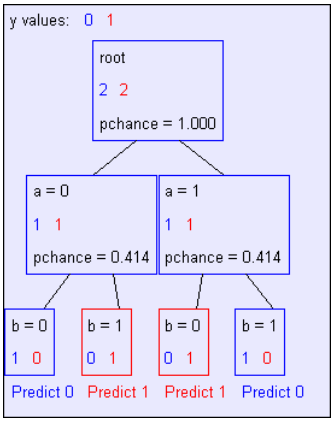


If we omit Base Case 3:

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

y = a XOR b

The resulting decision tree:



Simple example of tree construction using ID3 (Iterative Dichotomiser 3):

Suppose we want ID3 to decide whether the weather is amenable to playing baseball. Over the course of 2 weeks, data is collected to help ID3 build a decision tree (see table).

The target classification is "should we play baseball?" which can be yes or no.

The weather **attributes** are *outlook*, *temperature*, *humidity*, and *wind speed*. They can have the following values:

outlook = { sunny, overcast, rain }

temperature = {hot, mild, cool }

humidity = { high, normal }

wind = {weak, strong }

Examples of set S are:

Day	Outlook	Temperature	Humidity	Wind	Play ball
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Attribute Selection

How does ID3 decide which attribute is the best? A statistical property, called information **gain**, is used. Gain measures how well a given attribute separates training examples into targeted classes. The one with the highest information (information being the most useful for classification) is selected. In order to define gain, we first borrow an idea from information theory called entropy. Entropy measures the amount of information in an attribute.

Given a collection S of c outcomes

$$Entropy(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

where p_i is the proportion of S belonging to class i. S is over c. Log2 is log base 2.

Note that S is not an attribute but the entire sample set.

If S is a collection of 14 examples (see table) with 9 YES and 5 NO examples then

$$\text{Entropy(S)} = - (9/14) \text{ Log}_2 (9/14) - (5/14) \text{ Log}_2 (5/14) = 0.940$$

Notice entropy is 0 if all members of S belong to the same class (the data is perfectly classified). The range of entropy is 0 ("perfectly classified") to 1 ("totally random").

Gain(S, A) is information gain of example set S on attribute A is defined as

$$Gain(S,A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

Where:

S is each value v of all possible values of attribute A

S_v = subset of S for which attribute A has value v

$|S_v|$ = number of elements in S_v

|S| = number of elements in S

Suppose S is a set of 14 examples (see table) in which one of the attributes is wind speed. The values of Wind can be Weak or Strong. The classification of these 14 examples are 9 YES and 5 NO. For attribute Wind, suppose there are 8 occurrences of Wind = Weak and 6 occurrences of Wind = Strong. For Wind = Weak, 6 of the examples are YES and 2 are NO. For Wind = Strong, 3 are YES and 3 are NO. Therefore

$\text{Gain(S,Wind)} = \text{Entropy(S)} - (8/14) * \text{Entropy}(S_{weak}) - (6/14) * \text{Entropy}(S_{strong})$

$= 0.940 - (8/14) * 0.811 - (6/14) * 1.00 = 0.048$

$\text{Entropy}(S_{weak}) = - (6/8) * \log_2(6/8) - (2/8) * \log_2(2/8) = 0.811$

$\text{Entropy}(S_{strong}) = - (3/6) * \log_2(3/6) - (3/6) * \log_2(3/6) = 1.00$

For each attribute, the gain is calculated and the **highest gain** is used in the decision node.

We need to find which attribute will be the root node in our decision tree. The gain is calculated for all four attributes:

Gain(S, Outlook) = 0.246

Gain(S, Temperature) = 0.029

Gain(S, Humidity) = 0.151

Gain(S, Wind) = 0.048 (see before)

Outlook attribute has the highest gain, therefore it is used as the decision attribute in the root node.

Since Outlook has three possible values, the root node has three branches (sunny, overcast, rain). The next question is "what attribute should be tested at the Sunny branch node?" Since we have used Outlook at the root, we only decide on the remaining three attributes: Humidity, Temperature, or Wind.

S_{sunny} = {D1, D2, D8, D9, D11} = 5 examples from table 1 with outlook = sunny

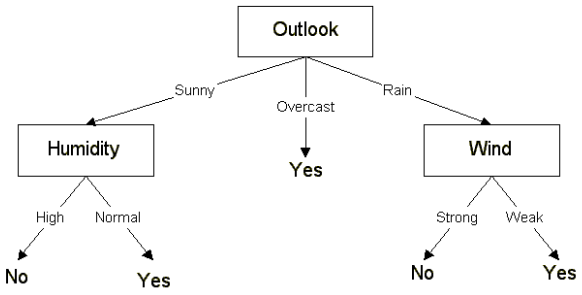
Gain(S_{sunny}, Humidity) = 0.970

Gain(S_{sunny}, Temperature) = 0.570

Gain(S_{sunny}, Wind) = 0.019

Humidity has the highest gain; therefore, it is used as the decision node. This process goes on until all data is classified perfectly or we run out of attributes.

The final decision tree



The decision tree can also be expressed in rule format:

IF outlook = sunny AND humidity = high THEN playball = no

IF outlook = rain AND humidity = high THEN playball = no

IF outlook = rain AND wind = strong THEN playball = yes

IF outlook = overcast THEN playball = yes

IF outlook = rain AND wind = weak THEN playball = yes

ID3 has been incorporated in a number of commercial rule-induction packages. Some specific applications include medical diagnosis, credit risk assessment of loan applications, equipment malfunctions by their cause, classification of soybean diseases, and web search classification.

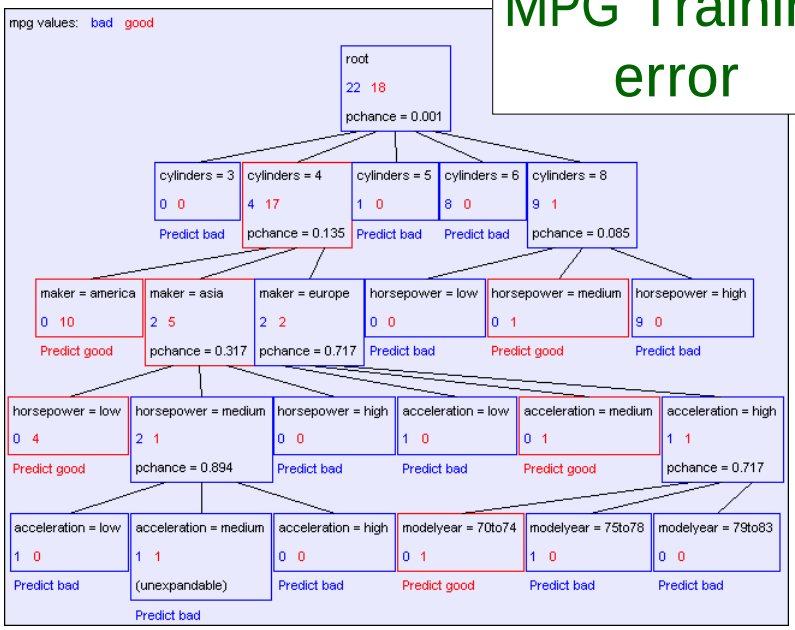
Training Set Error

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

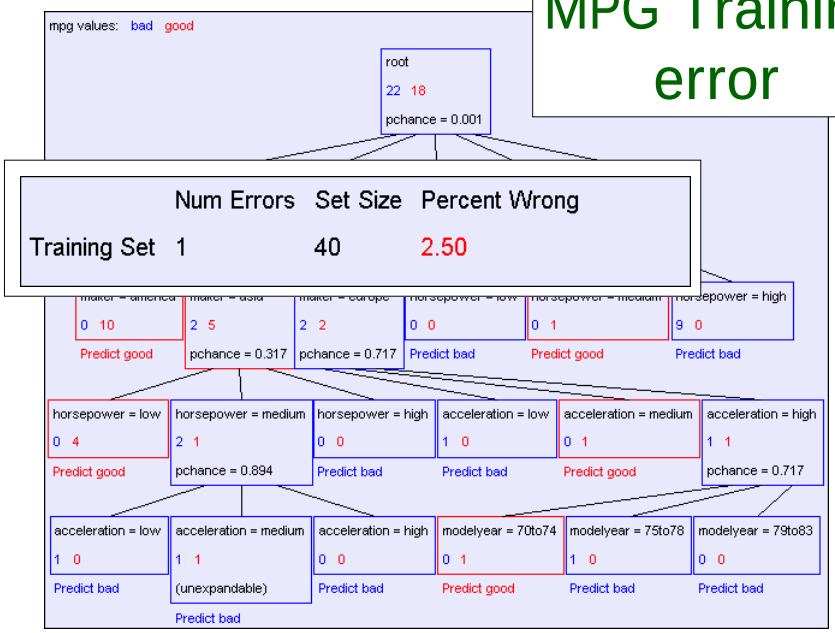
  For what number of records does the decision tree's prediction disagree with the true value in the database?

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

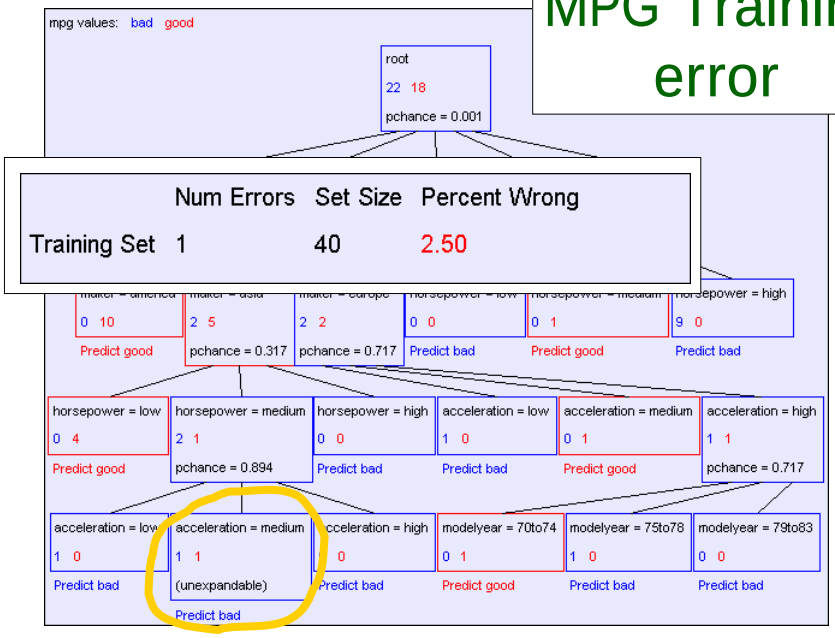
MPG Training error



MPG Training error



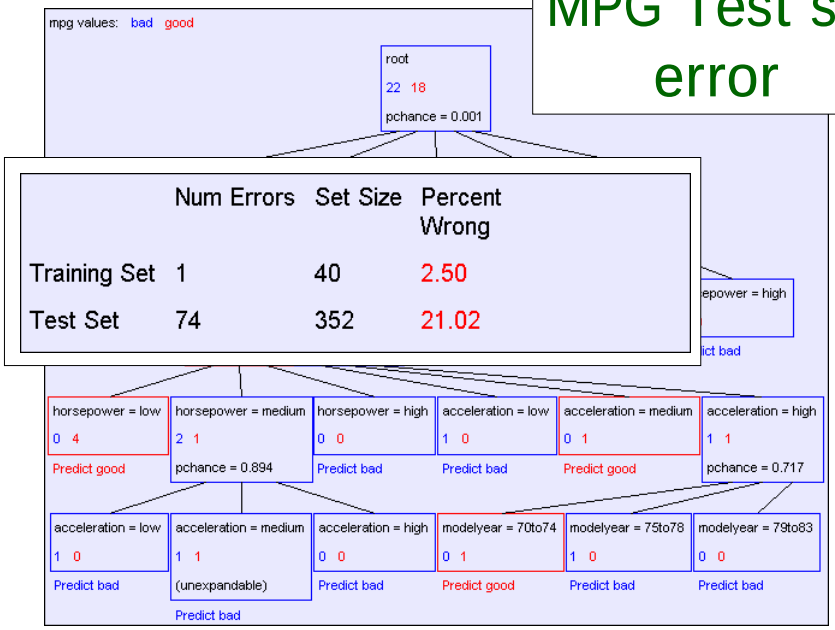
MPG Training error



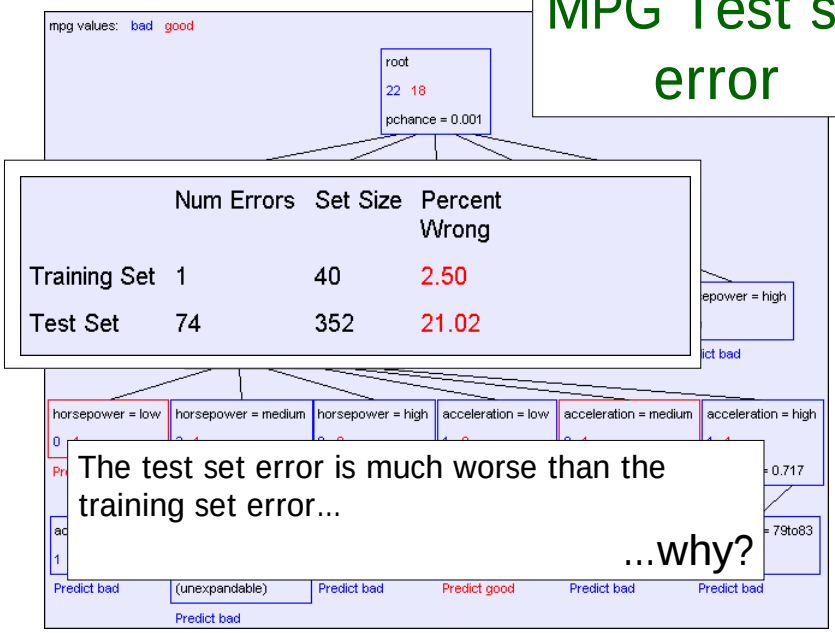
Test Set Error

- 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



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

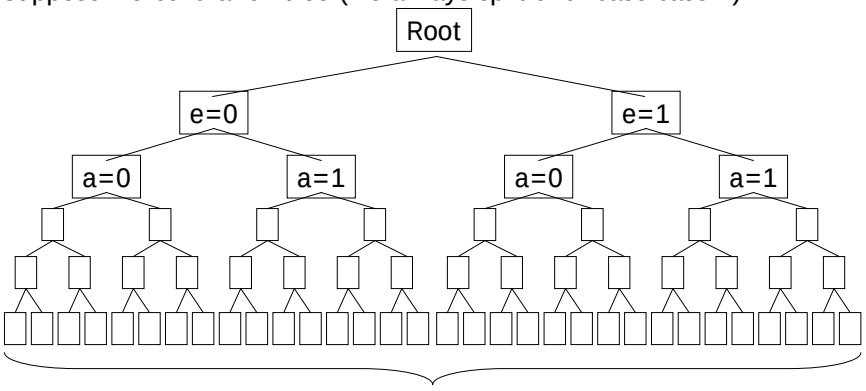
a	b	c	d	e	y
0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	1	0	0
0	0	0	1	1	1
0	0	1	0	0	1
:	:	:	:	:	:
1	1	1	1	1	1

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

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

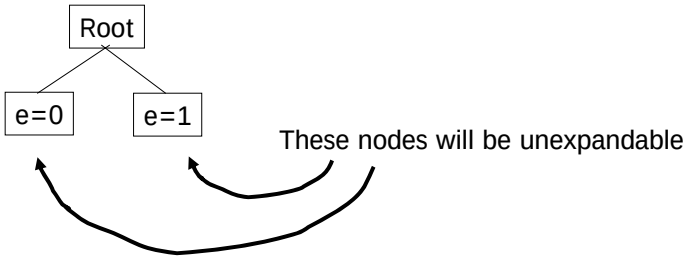
These bits are hidden

32 records

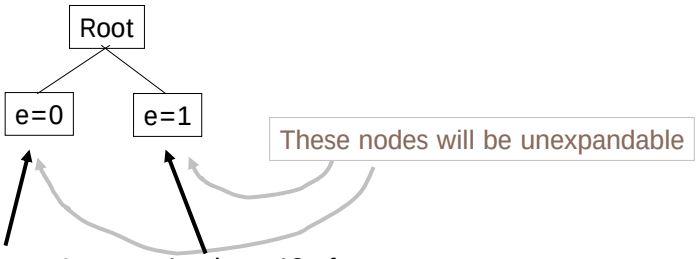
a	b	c	d	e	y
0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	1	0	0
0	0	0	1	1	1
0	0	1	0	0	1
:	:	:	:	:	:
1	1	1	1	1	1

What decision tree would we learn now?

Without access to the irrelevant bits...



Without access to the irrelevant bits...



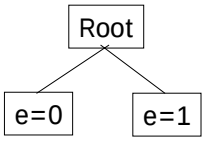
In about 12 of the 16 records in this node the output will be 0

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

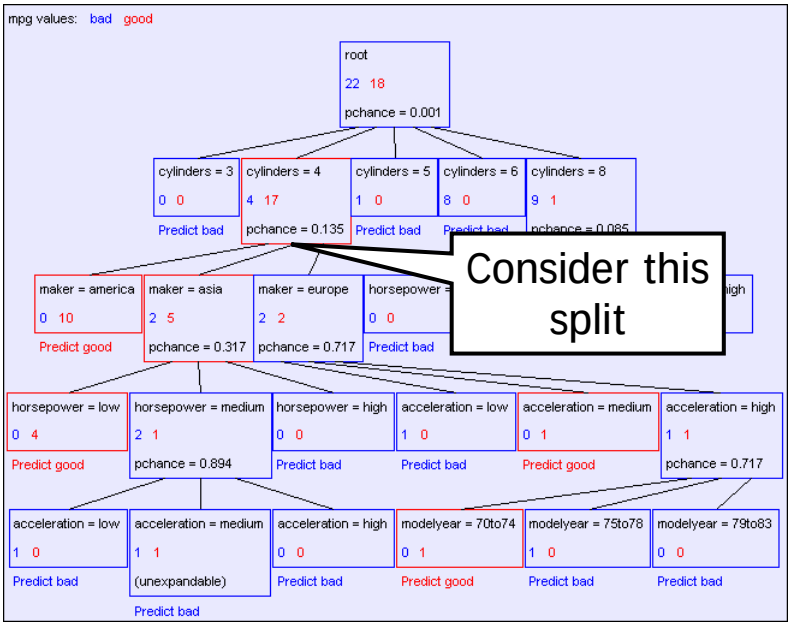
- Definition: If your machine learning algorithm fits noise (i.e. pays attention to parts of the data that are irrelevant) it is overfitting.
- Fact (theoretical and empirical): If your machine learning algorithm is overfitting then it may perform less well on test set data.

Avoiding overfitting

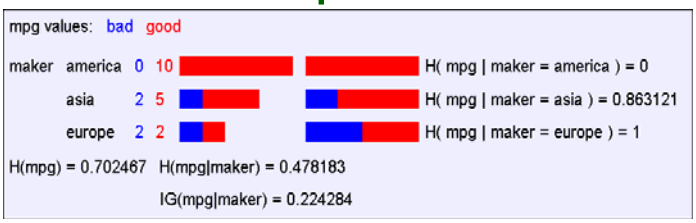
- Usually we do not know in advance which are the irrelevant variables
- ...and it may depend on the context

  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.

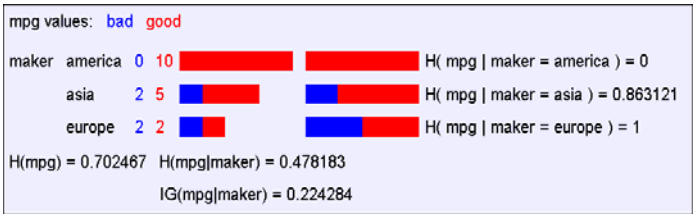


A chi-squared test



- 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%.

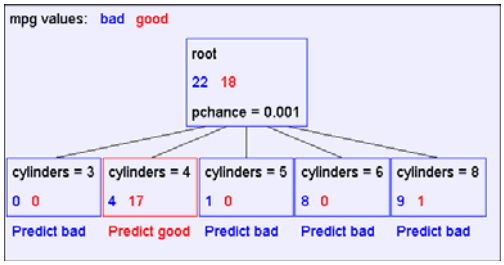
Using Chi-squared to avoid overfitting

- 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 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

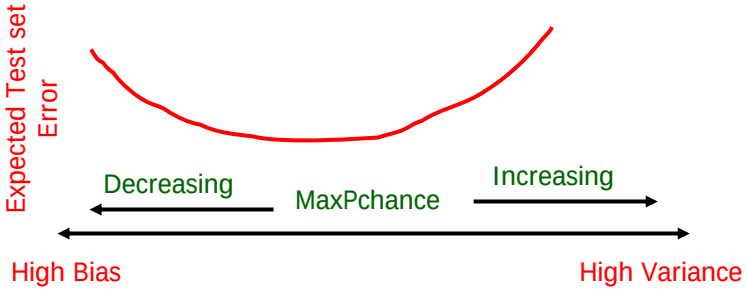
	Num Errors	Set Size	Percent Wrong
Training Set	5	40	12.50
Test Set	56	352	15.91

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.

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

Strengths and Weakness of Decision Tree Methods

The strengths of decision tree methods are:

- Decision trees are able to generate understandable rules.

- Decision trees perform classification without requiring much computation.

- Decision trees are able to handle both continuous and categorical variables.

- Decision trees provide a clear indication of which fields are most important for prediction or classification.

The weaknesses of decision tree methods

- Decision trees are less appropriate for estimation tasks where the goal is to predict the value of a continuous attribute.

- Decision trees are prone to errors in classification problems with many class and relatively small number of training examples.

- Decision tree can be computationally expensive to train. The process of growing a decision tree is computationally expensive. At each node, each candidate splitting field must be sorted before its best split can be found. In some algorithms, combinations of fields are used and a search must be made for optimal combining weights. Pruning algorithms can also be expensive since many candidate sub-trees must be formed and compared.

- Decision trees do not treat well non-rectangular regions. Most decision-tree algorithms only examine a single field at a time. This leads to rectangular classification boxes that may not correspond well with the actual distribution of records in the decision space.

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