Boosting continued

Machine Learning – CSE546
Carlos Guestrin
University of Washington

October 14, 2013

Voting (Ensemble Methods)

- Instead of learning a single (weak) classifier, learn **many weak classifiers** that are **good at different parts of the input space**

  $h_i : X \to Y \in \{-1, +1\}$
- **Output class:** (Weighted) vote of each classifier
  - Classifiers that are most "sure" will vote with more conviction
  - Classifiers will be most "sure" about a particular part of the space
  - On average, do better than single classifier!

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

t^{th} weak classifier

vote weight

e.g.

$$h_t(x) = \begin{cases} +1 & \text{if } x_i = 1 \\ -1 & \text{if } x_i = 0 \end{cases}$$

if email has word "CSE546" → no spam
else spam

- **But how do you ???**
  - force classifiers to learn about different parts of the input space?
  - weigh the votes of different classifiers?

AdaBoost

uniform weights

- Initialize weights to uniform dist: $D_1(j) = 1/N$
- For t = 1…T
  - Train weak learner h_t on distribution D_t over the data ← Focused on parts that have high weights
  - Choose weight α_t ← Magic of taking derivative & setting to 0
  - Update weights: for each j

    new weight, old weight

    $$D_{t+1}(j) = \frac{D_t(j)\exp(-\alpha_t y^j h_t(x^j))}{Z_t}$$

    supposed $\alpha_t > 0$
    if h_t correct on j
    $y^j h_t(x^j) > 0$
    ⇒ weight decrease
    if h_t incorrect on j
    weight increases

    - Where Z_t is normalizer: $$Z_t = \sum_{j=1}^{N} D_t(j)\exp(-\alpha_t y^j h_t(x^j))$$

      so weights add up to 1
- Output final classifier:

  $$H(x) = \text{Sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

Why choose α_t for hypothesis h_t this way?

[Schapire, 1989]

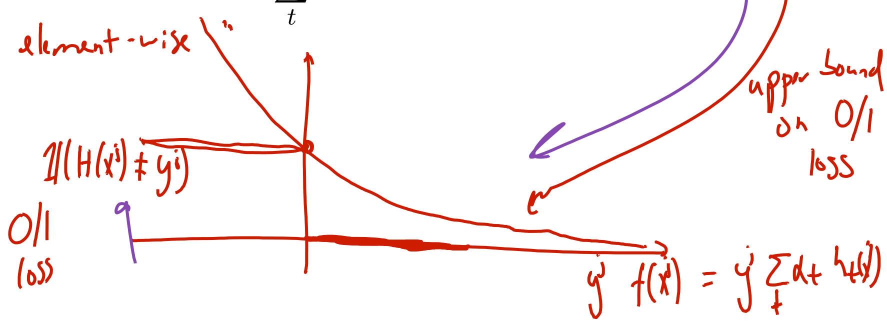
Training error of final classifier is bounded by:

unweighted train error of classifier

$$\frac{1}{N}\sum_{j=1}^{N} \mathbb{1}[H(x^j) \neq y^j] \leq \frac{1}{N}\sum_{j=1}^{N} \exp(-y^j f(x^j))$$

Where $f(x) = \sum_t \alpha_t h_t(x);\ H(x) = sign(f(x))$

element-wise



Why choose α_t for hypothesis h_t this way?

[Schapire, 1989]

We can minimize this bound by choosing α_t on each iteration to minimize Z_t

$$Z_t = \sum_{j=1}^{N} D_t(j)\exp(-\alpha_t y^j h_t(x^j))$$

take derivative & set to $\varnothing$

For boolean target function, this is accomplished by [Freund & Schapire '97]:

$$\boxed{\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)}$$

You'll prove this in your homework! ☺

Strong, weak classifiers

$\varepsilon_t < \frac{1}{2}$

$\Rightarrow h_t$ is better than random

→ easy

→ not easy

- If each classifier is (at least slightly) better than random
  - $\varepsilon_t < 0.5$

$\varepsilon_t < 0.5 - \gamma \;;\; \gamma > 0$

- AdaBoost will achieve zero *training error* (exponentially fast):

$$\underbrace{\frac{1}{N}\sum_{j=1}^{N} \mathbb{1}[H(x^j) \neq y^j]}_{\text{train error}} \leq \prod_{t=1}^{T} Z_t \leq \exp\left(-2\sum_{t=1}^{T}(1/2-\epsilon_t)^2\right)$$

$$= \prod_{t=1}^{T} \underbrace{e^{-2(\frac{1}{2}-\varepsilon_t)^2}}_{<1} \Rightarrow \varepsilon_t < \frac{1}{2}$$

$\Rightarrow \varepsilon_t > \frac{1}{2} \Leftarrow \alpha_t$ will flip sign automatically

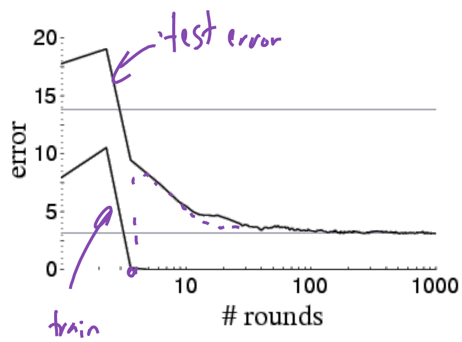
extra credit, no cheating!!

no coincident points

- Is it hard to achieve better than random ~~training~~ error? guessing

Boosting results – Digit recognition

[Schapire, 1989]

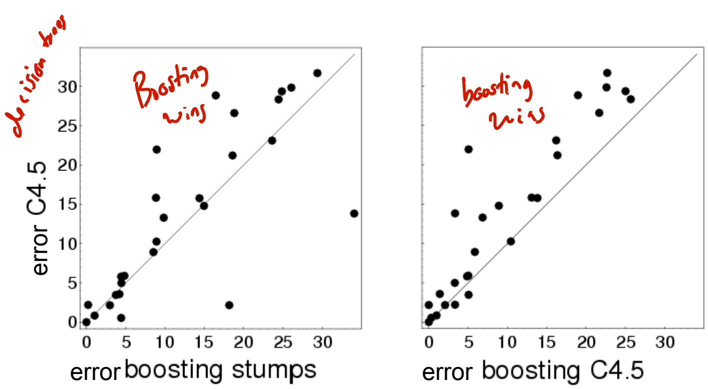


- Boosting often
  - Robust to overfitting
  - Test set error decreases even after training error is zero

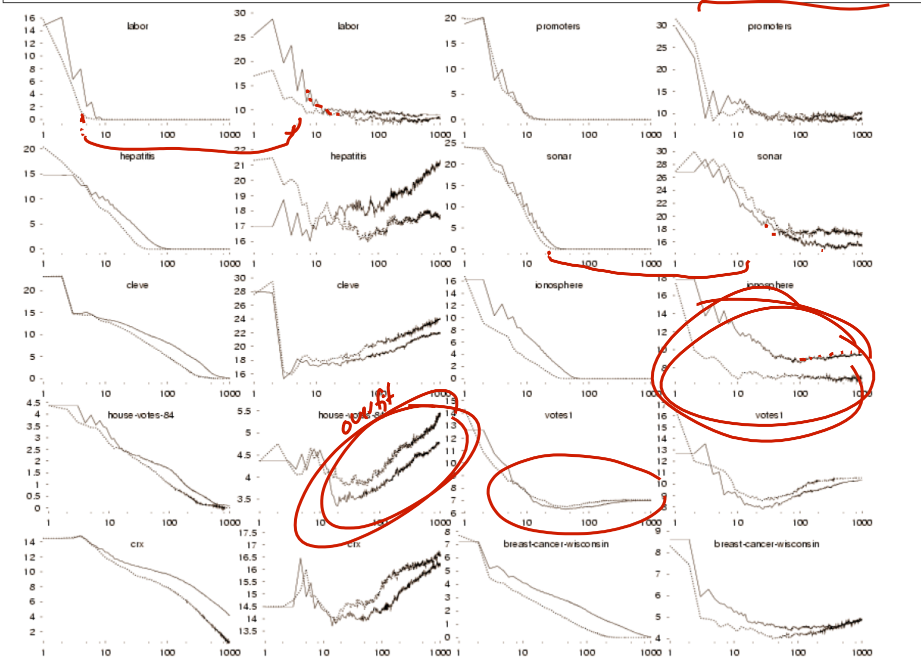
Boosting: Experimental Results

[Freund & Schapire, 1996]

Comparison of C4.5, Boosting C4.5, Boosting decision stumps (depth 1 trees), 27 benchmark datasets



AdaBoost and AdaBoost.MH on Train (left) and Test (right) data from Irvine repository. [Schapire and Singer, ML 1999]



Boosting and Logistic Regression

Logistic regression assumes:

$f(x) = w_0 + \sum_i w_i h_i(x)$

$$P(Y=1|X) = \frac{1}{1+\exp(f(x))}$$

And tries to maximize data likelihood:

$y_j \in \{-1,+1\}$

$$\ln P(\mathcal{D}|H) = \ln \prod_{j=1}^{N} \frac{1}{1+\exp(-y^j f(x^j))}$$

Equivalent to minimizing log loss

$$\sum_{j=1}^{N} \ln(1+\exp(-y^j f(x^j)))$$

Boosting and Logistic Regression

$\arg\min_w f(w) = \arg\min_w c\, f(w)$

$c > 0$

Logistic regression equivalent to minimizing log loss

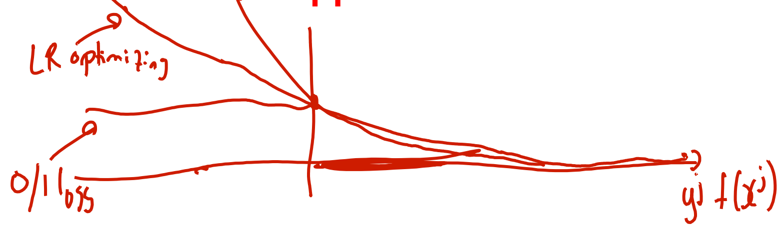
$$c \sum_{j=1}^{N} \ln(1 + \exp(-y^j f(x^j)))$$

Boosting minimizes similar loss function!!

$$\frac{1}{N} \sum_{j=1}^{N} \exp(-y^j f(x^j)) = \prod_{t=1}^{T} Z_t$$

boosting optimizing

Both smooth approximations of 0/1 loss!



Logistic regression and Boosting

Logistic regression:

- Minimize loss fn

$$\sum_{j=1}^{N} \ln(1 + \exp(-y^j f(x^j)))$$

- Define

fixed

$$f(x) = w_0 + \sum_i w_i x_i$$

where features x_i are predefined

- Weights w_i are learned in joint optimization

Boosting:

- Minimize loss fn

$$\sum_{j=1}^{N} \exp(-y^j f(x^j))$$

- Define

learned

$$f(x) = \sum_t \alpha_t h_t(x)$$

where $h_t(x)$ defined dynamically to fit data

(not a linear classifier)

greedy

- Weights α_t learned incrementally

What you need to know about Boosting

- Combine weak classifiers to obtain very strong classifier
  - Weak classifier – slightly better than random on training data
  - Resulting very strong classifier – can eventually provide zero training error
- AdaBoost algorithm
- Boosting v. Logistic Regression
  - Similar loss functions
  - Single optimization (LR) v. Incrementally improving classification (B)
- Most popular application of Boosting:
  - Boosted decision stumps!
  - Very simple to implement, very effective classifier

Projects

- An opportunity to exercise what you learned and to learn new things
- Individually or groups of two
- Must involve real data
  - Must be data that you have available to you by the time of the project proposals
- Must involve machine learning
- It's encouraged to be related to your research, but must be something new you did this quarter
  - Not a project you worked on during the summer, last year, etc.
- Sample projects on course website
- Wed., October 23 at 9:00am: **Project Proposals**
- Mon., November 11 at 9:00am: **Project Milestone**
- Wed., December 4, 3-5pm: **Poster Session**
- Mon., December 9 at 9:00am: **Project Report**

Decision Trees

Machine Learning – CSE546

Carlos Guestrin

University of Washington

October 16, 2013

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 $\mathbf{x}=\{x_1,\ldots,x_k\}$ is a positive example
    - $w_0 + \sum_i w_i x_i < 0$; if $\mathbf{x}=\{x_1,\ldots,x_k\}$ is a negative example

Not linearly separable data

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

$X_1 \oplus X_2$ $\quad$ $X_1 \wedge \neg X_2 \vee \neg X_1 \wedge X_2$

\+ $\quad$ −

no linear separator

− $\quad$ +

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 $h_{\mathbf{w}}(\mathbf{x})$ still linear in parameters $\mathbf{w}$
  - As easy to learn
  - Data is linearly separable in higher dimensional spaces
  - More discussion later this quarter

quadratic features

\+ $\quad$ −

− $\quad$ +

Addressing non-linearly separable data – Option 2, non-linear classifier

- Choose a classifier $h_{\mathbf{w}}(\mathbf{x})$ that is non-linear in parameters $\mathbf{w}$, 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

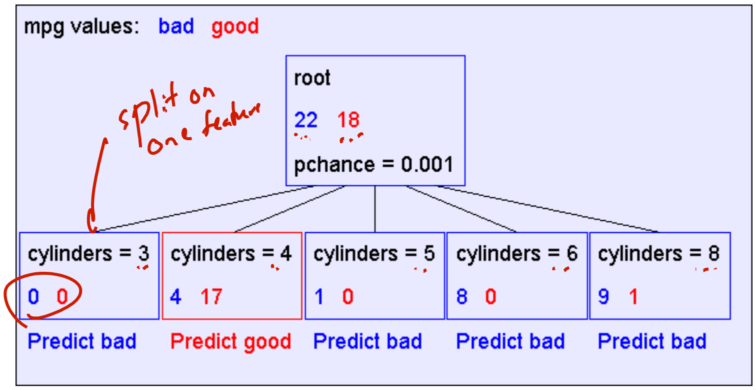
X → Y : MPG

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

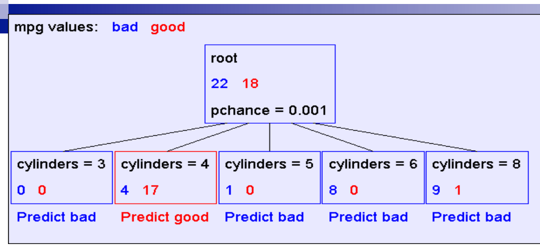
40 training examples

From the UCI repository (thanks to Ross Quinlan)

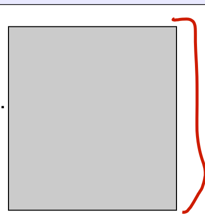
A Decision Stump



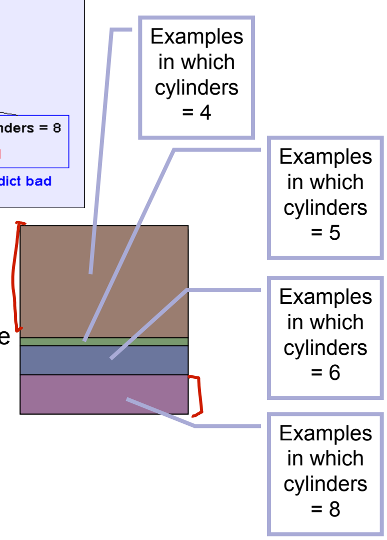
Recursion Step



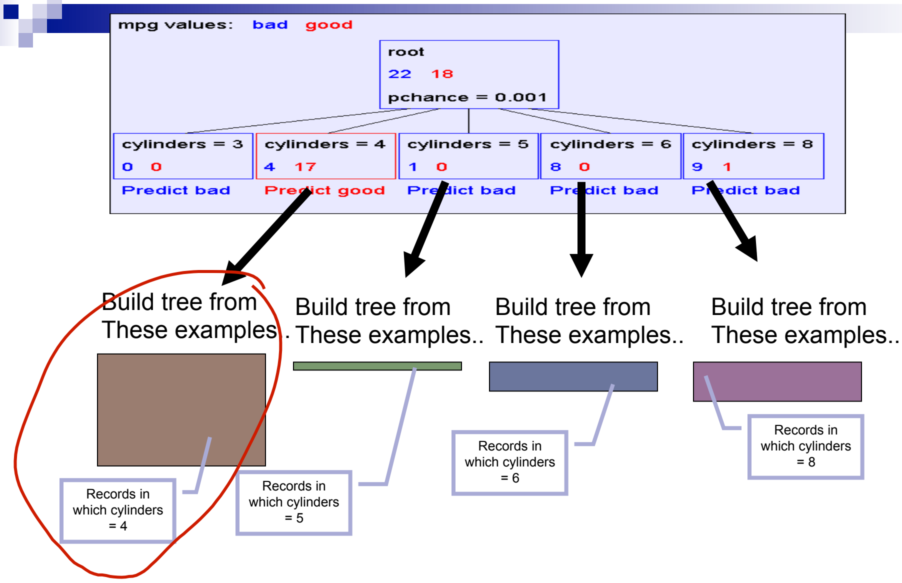
Take the Original Dataset..



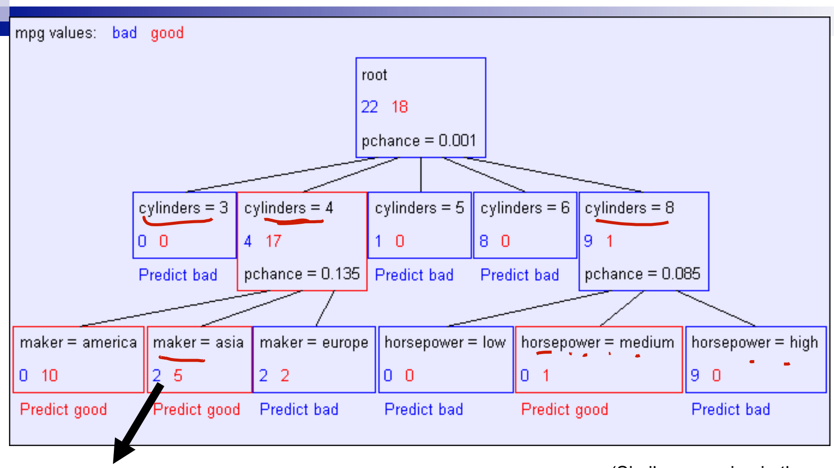
And partition it according to the value of the attribute we split on



Recursion Step



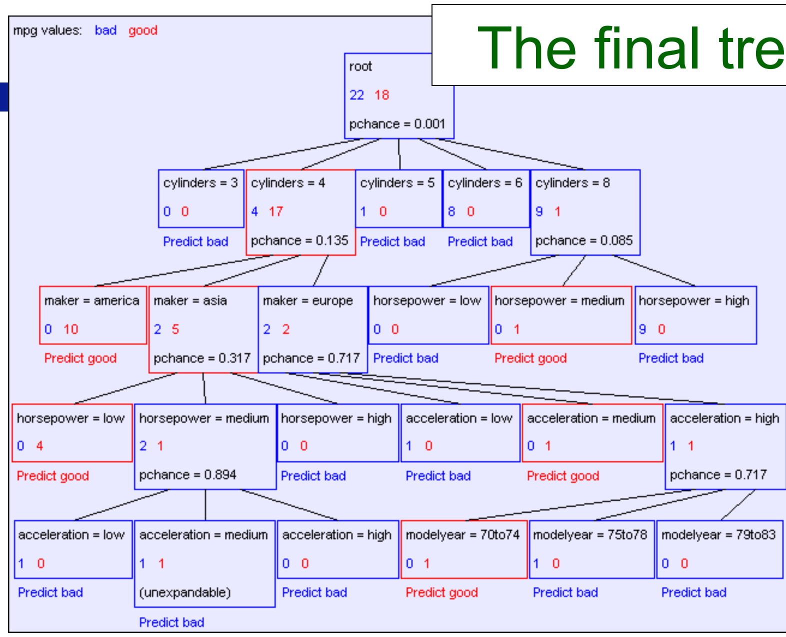
Second level of tree



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

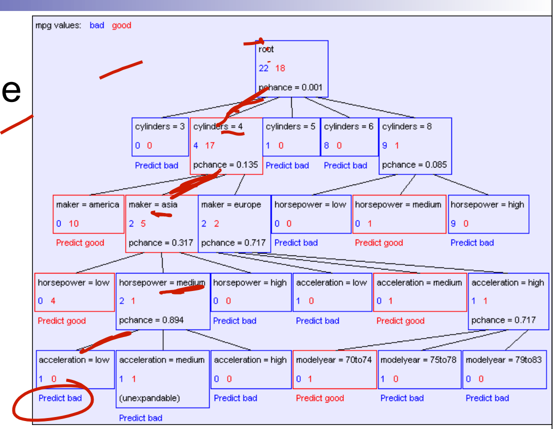
(Similar recursion in the other cases)

The final tree



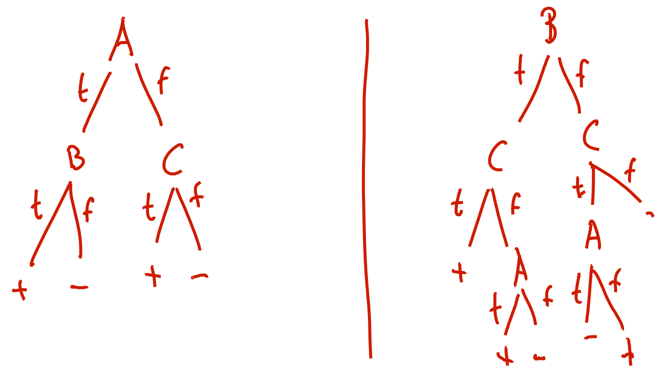
Classification of a new example

- 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., $\phi = A \wedge B \vee \neg A \wedge C$ ((A and B) or (not A and C))



much larger, but represent same fn.

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

X_1	X_2	Y
T	T	T
T	F	T
T	T	T
T	F	T
F	T	T
F	F	F
F	T	F
F	F	F

X_1: t / f

Y: t:4 f:0 | t:1 f:3

sure!

kind of sure

X_2: t / f

Y: t:3 f:1 | Y: t:2 f:2

don't know nothing

X_1 is a "better" split, because I become more sure

Measuring uncertainty

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

After the split

more sure, less uncertainty

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

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

← uniform bad

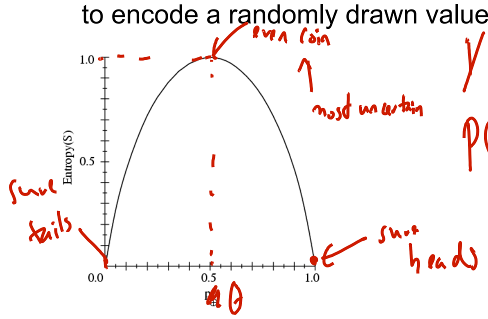
Entropy

Entropy $H(\cancel{X}Y)$ 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)



Y is binary

$P(Y) = \theta$

Andrew Moore's Entropy in a nutshell

Low Entropy

High Entropy

Andrew Moore's Entropy in a nutshell

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

Information gain

$P(Y=T) = 5/6$

X_1	X_2	Y
T	T	T
T	F	T
T	T	T
T	F	T
F	T	T
F	F	F

- Advantage of attribute – decrease in uncertainty
  - Entropy of Y before you split $H(Y) = -\sum_y P(y) \log P(y)$
    $= -\frac{5}{6}\log\frac{5}{6} - \frac{1}{6}\log\frac{1}{6} = .65$
  - Entropy after split
    - Weight by probability of following each branch, i.e., normalized number of records

$H(Y \mid X = x_j)$

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

how often I take on branch x_j

X_1: t $2/3$, f $1/3$

entropy 0 → Y: t:4, f:0

Y: t:1, f:1 ← entropy = 1

$H(Y|X_1) = \frac{2}{3} \cdot 0 + \frac{1}{3} \cdot 1 = \frac{1}{3}$

- Information gain is difference $IG(X) = H(Y) - H(Y \mid X)$

$IG(X_1) = 0.65 - \frac{1}{3} \approx 0.32$

Learning decision trees

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

when do I stop?
1. when info gain is small??
2. entropy $\emptyset$ on leaf, correct class
3. nothing to split on — used all features in this path
   — no features help??

Suppose we want to predict MPG

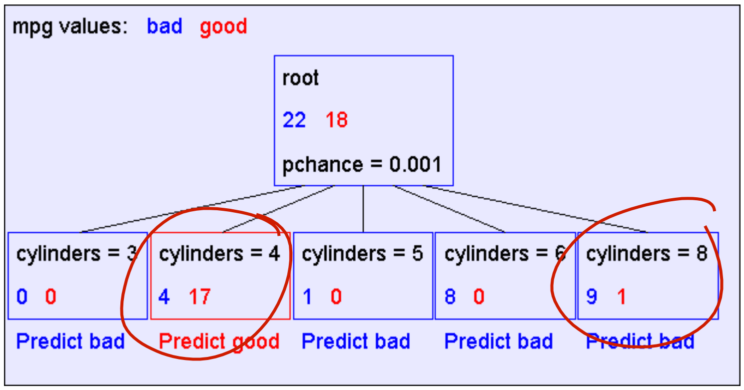
Look at all the information gains…

Information gains using the training set (40 records)

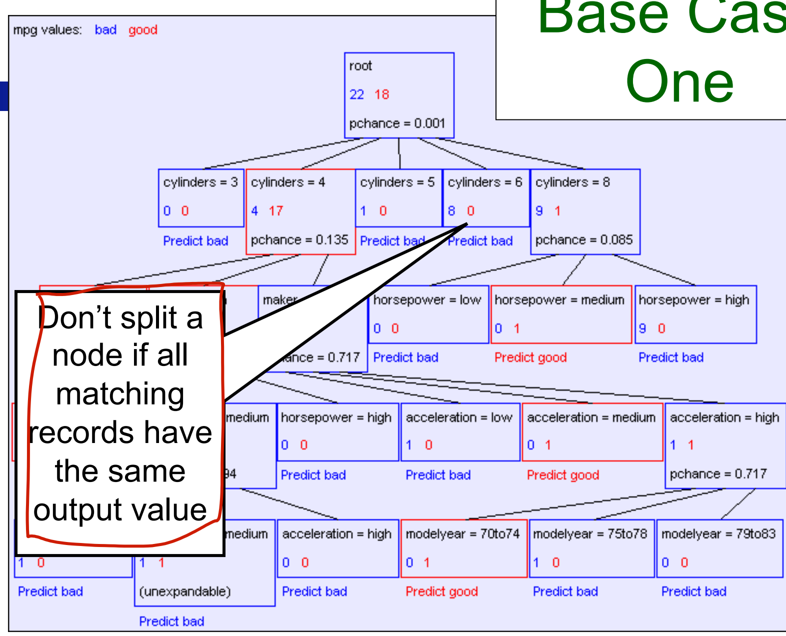
mpg values: bad good

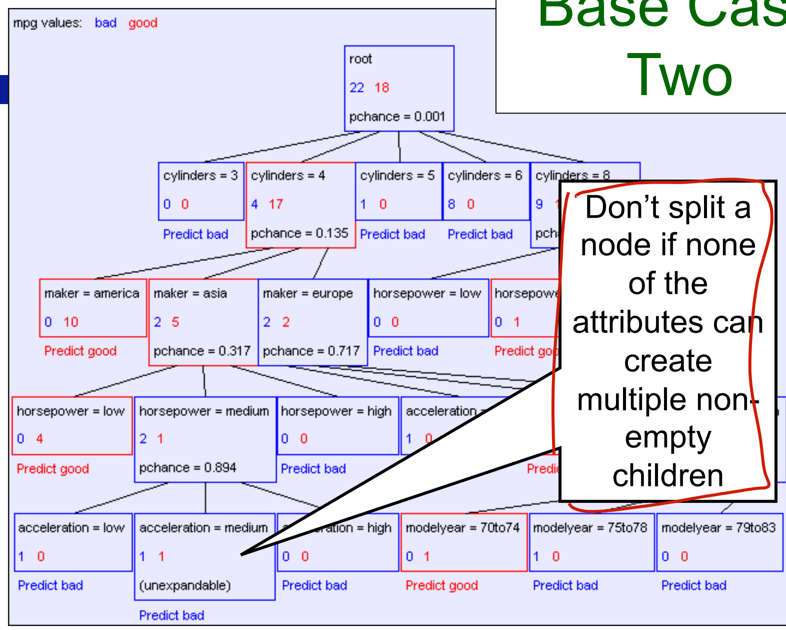
Input	Value	Distribution	Info Gain
cylinders	3		0.506731
	4		
	5		
	6		
	8		
displacement	low		0.223144
	medium		
	high		
horsepower	low		0.387605
	medium		
	high		
weight	low		0.304018
	medium		
	high		
acceleration	low		0.0642088
	medium		
	high		
modelyear	70to74		0.267964
	75to78		
	79to83		
maker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

A Decision Stump



Base Case One

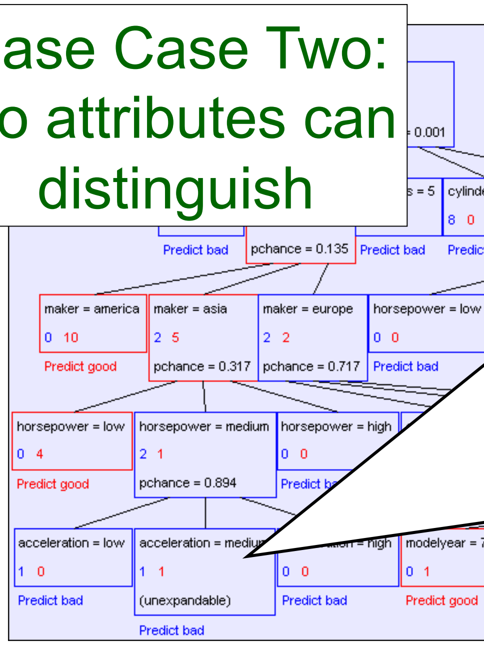




Base Case Two

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

$H(Y)=1$

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

Y = A XOR B

A

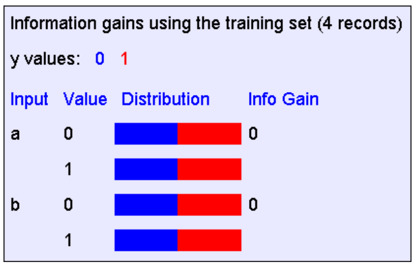
t f

$H(Y|A=t)=1$ Y = t: 1 f: 1   Y = t: 1 f: 1 $H(Y|A=f)=1$

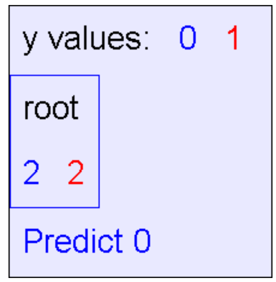
$IG(A)=0$

$IG(B)=0$

The information gains:



The resulting bad decision tree:



If we omit Base Case 3:

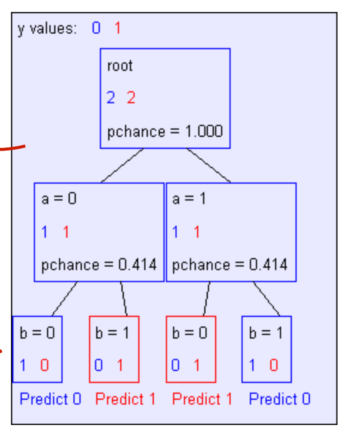
a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

y = a XOR b

~~The resulting decision tree:~~

low info gain is not a good stopping criteria

perfect classification



Basic Decision Tree Building Summarized

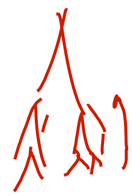
BuildTree(*DataSet*,*Output*)

- 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.

go for ever



prune to simplify