Decision Trees, MDL, Boosting

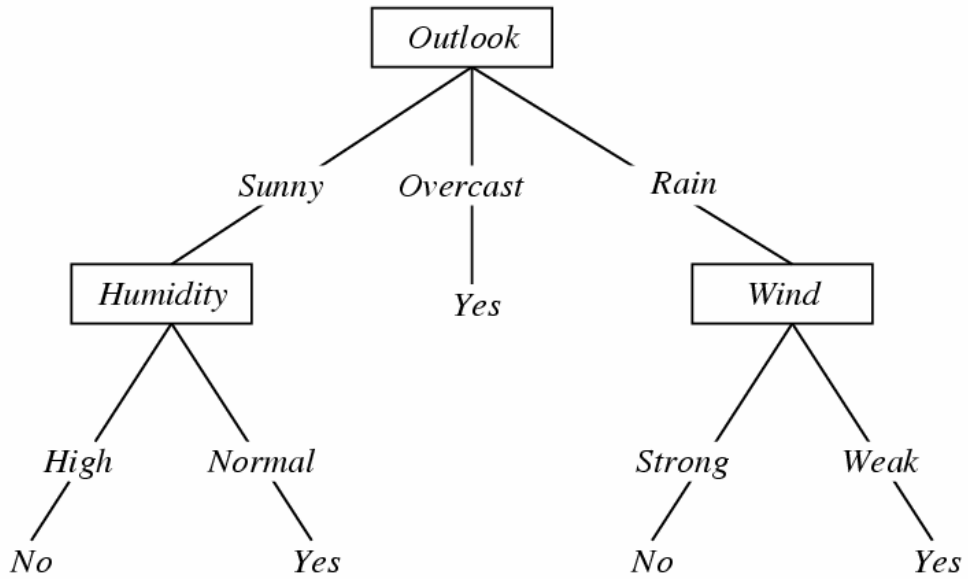
Recommended reading:

- Decision trees: Mitchell Chapter 3
- MDL: Mitchell Chapter 6.6; Bishop Chapter 10.10;
- Boosting: "The Boosting Approach to Machine Learning: An Overview," R. Schapire, MSRI Workshop on Nonlinear Estimation and Classification, 2002. (see class website)

Machine Learning 10-701

Tom M. Mitchell
Carnegie Mellon University

Decision Tree for *PlayTennis*



How would you represent

AB V CD(¬E)?

Each internal node: test one attribute X_i

Each branch from a node: selects one value for X_i

Each leaf node: predict Y (or $P(Y|X \in \text{leaf})$)

A Tree to Predict C-Section Risk

Learned from medical records of 1000 women

Negative examples are C-sections

```
[833+,167-] .83+ .17-
Fetal_Presentation = 1: [822+,116-] .88+ .12-
| Previous_Csection = 0: [767+,81-] .90+ .10-
| | Primiparous = 0: [399+,13-] .97+ .03-
| | Primiparous = 1: [368+,68-] .84+ .16-
| | | Fetal_Distress = 0: [334+,47-] .88+ .12-
| | | | Birth_Weight < 3349: [201+,10.6-] .95+ .
| | | | Birth_Weight >= 3349: [133+,36.4-] .78+
| | | Fetal_Distress = 1: [34+,21-] .62+ .38-
| Previous_Csection = 1: [55+,35-] .61+ .39-
Fetal_Presentation = 2: [3+,29-] .11+ .89-
Fetal_Presentation = 3: [8+,22-] .27+ .73-
```

Top-Down Induction of Decision Trees

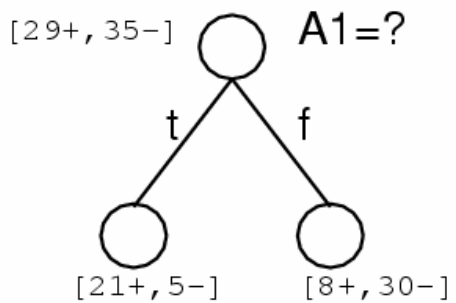
[ID3, C4.5, …]

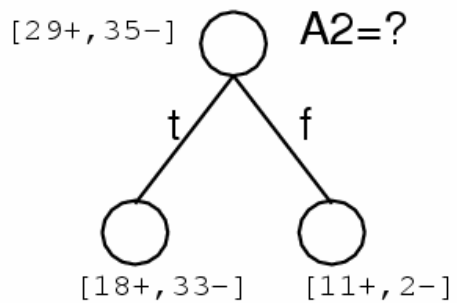
node = Root

Main loop:

1. $A \leftarrow$ the "best" decision attribute for next *node*
2. Assign A as decision attribute for *node*
3. For each value of A, create new descendant of *node*
4. Sort training examples to leaf nodes
5. If training examples perfectly classified, Then STOP, Else iterate over new leaf nodes

Which attribute is best?





Entropy

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

$$H(X) = -\sum_{i=1}^{n} P(X=i)\log_2 P(X=i)$$

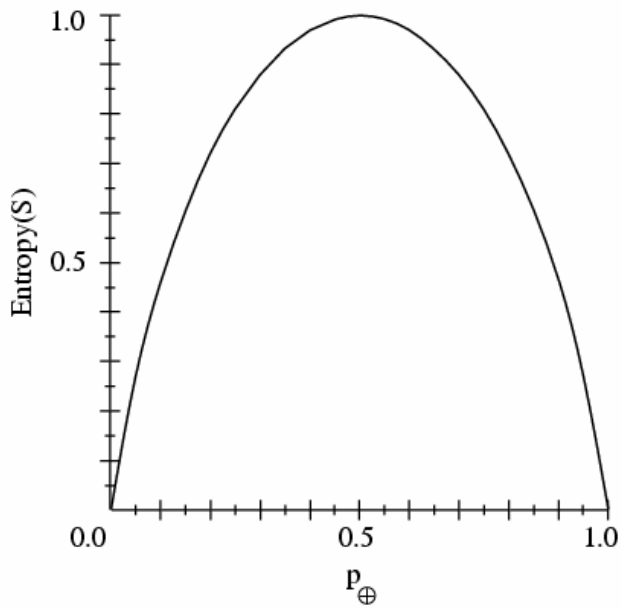
$H(X)$ is the expected number of bits needed to encode a randomly drawn value of X (under most efficient code)

Why? Information theory:

- Most efficient code assigns $-\log_2 P(X=i)$ bits to encode the message $X=i$
- So, expected number of bits is:

$$\sum_{i=1}^{n} P(X=i)(-\log_2 P(X=i))$$

Sample Entropy



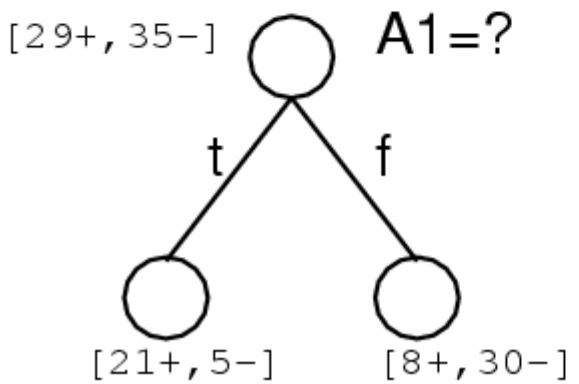
- S is a sample of training examples
- $p_\oplus$ is the proportion of positive examples in S
- $p_\ominus$ is the proportion of negative examples in S
- Entropy measures the impurity of S

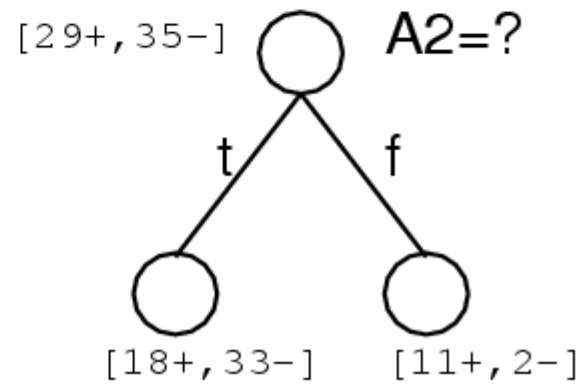
$$Entropy(S) \equiv -p_\oplus \log_2 p_\oplus - p_\ominus \log_2 p_\ominus$$

Information Gain

$Gain(S, A)$ = expected reduction in entropy due to sorting on A

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



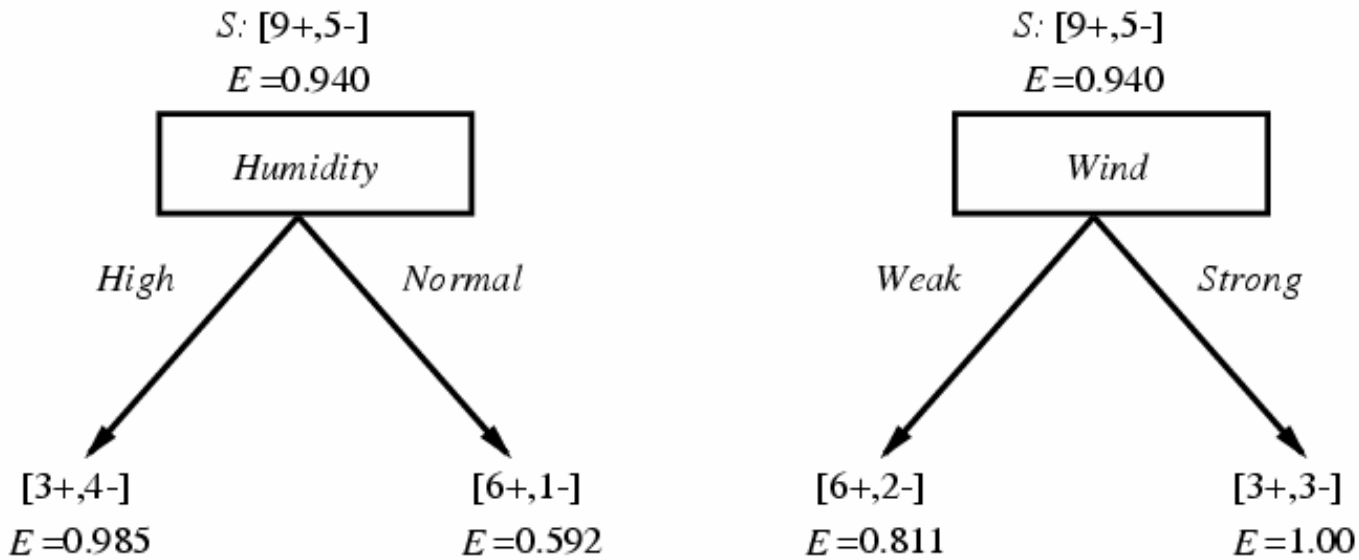


Training Examples

Day	Outlook	Temperature	Humidity	Wind	PlayTenn
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

Selecting the Next Attribute

Which attribute is the best classifier?

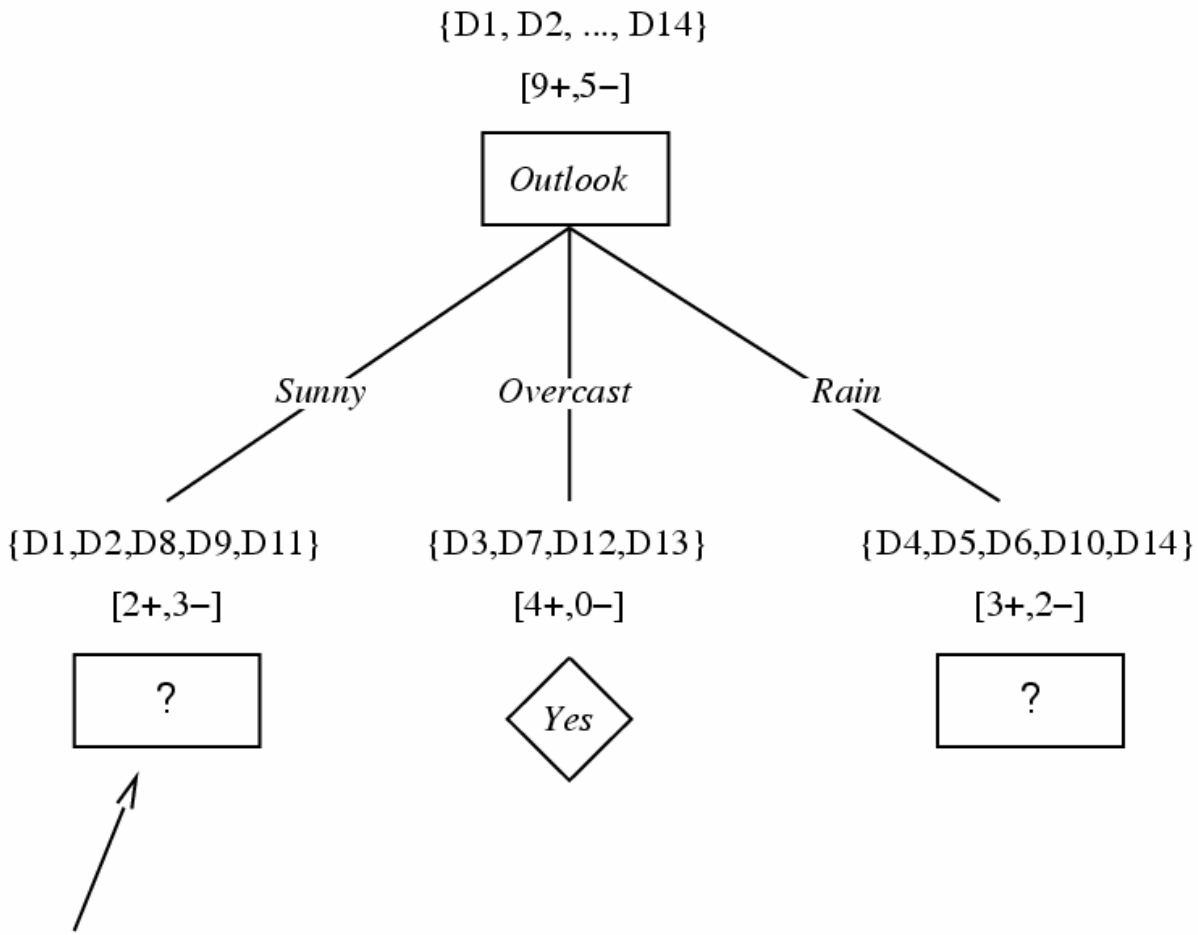


Gain (S, Humidity)

= .940 - (7/14).985 - (7/14).592
= .151

Gain (S, Wind)

= .940 - (8/14).811 - (6/14)1.0
= .048



Which attribute should be tested here?

S_{sunny} = {D1,D2,D8,D9,D11}

Gain (S_{sunny} , *Humidity*) = .970 − (3/5) 0.0 − (2/5) 0.0 = .970

Gain (S_{sunny} , *Temperature*) = .970 − (2/5) 0.0 − (2/5) 1.0 − (1/5) 0.0 = .570

Gain (S_{sunny} , *Wind*) = .970 − (2/5) 1.0 − (3/5) .918 = .019

Inductive Bias in ID3

Note H is the power set of instances X

$\rightarrow$Unbiased?

Not really...

- Preference for short trees, and for those with high information gain attributes near the root
- Bias is a *preference* for some hypotheses, rather than a *restriction* of hypothesis space H
- Occam's razor: prefer the shortest hypothesis that fits the data

Occam's Razor

Why prefer short hypotheses?

Argument in favor:

- Fewer short hyps. than long hyps.

→ a short hyp that fits data unlikely to be coincidence

→ a long hyp that fits data might be coincidence

Argument opposed:

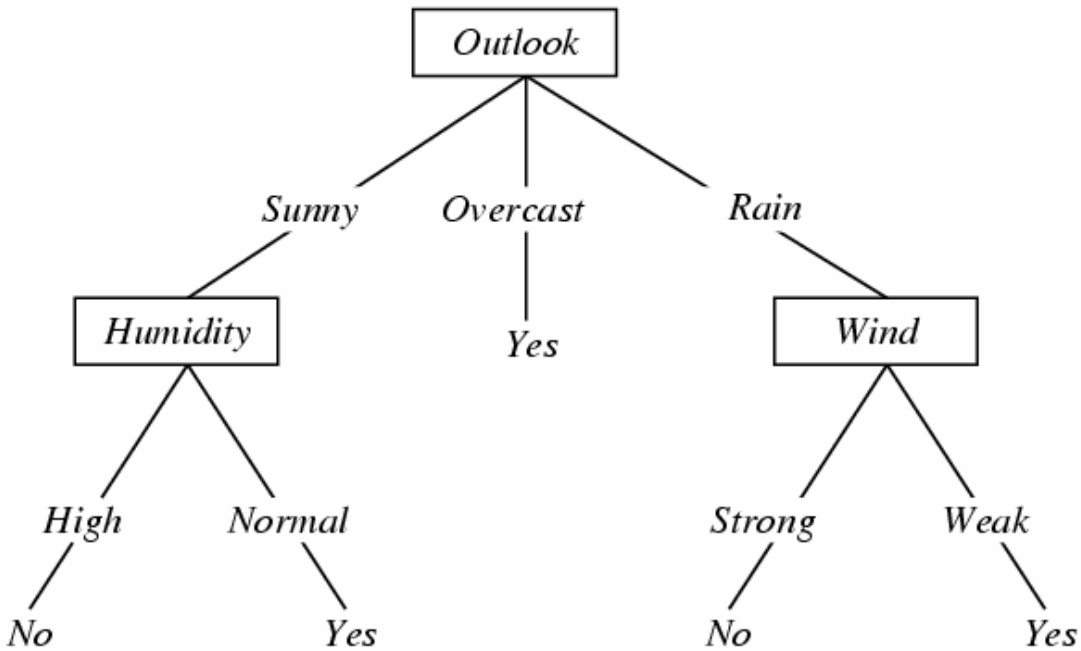
- There are many ways to define small sets of hyps
- e.g., all trees with a prime number of nodes that use attributes beginning with "Z"
- What's so special about small sets based on *size* of hypothesis??

Overfitting in Decision Trees

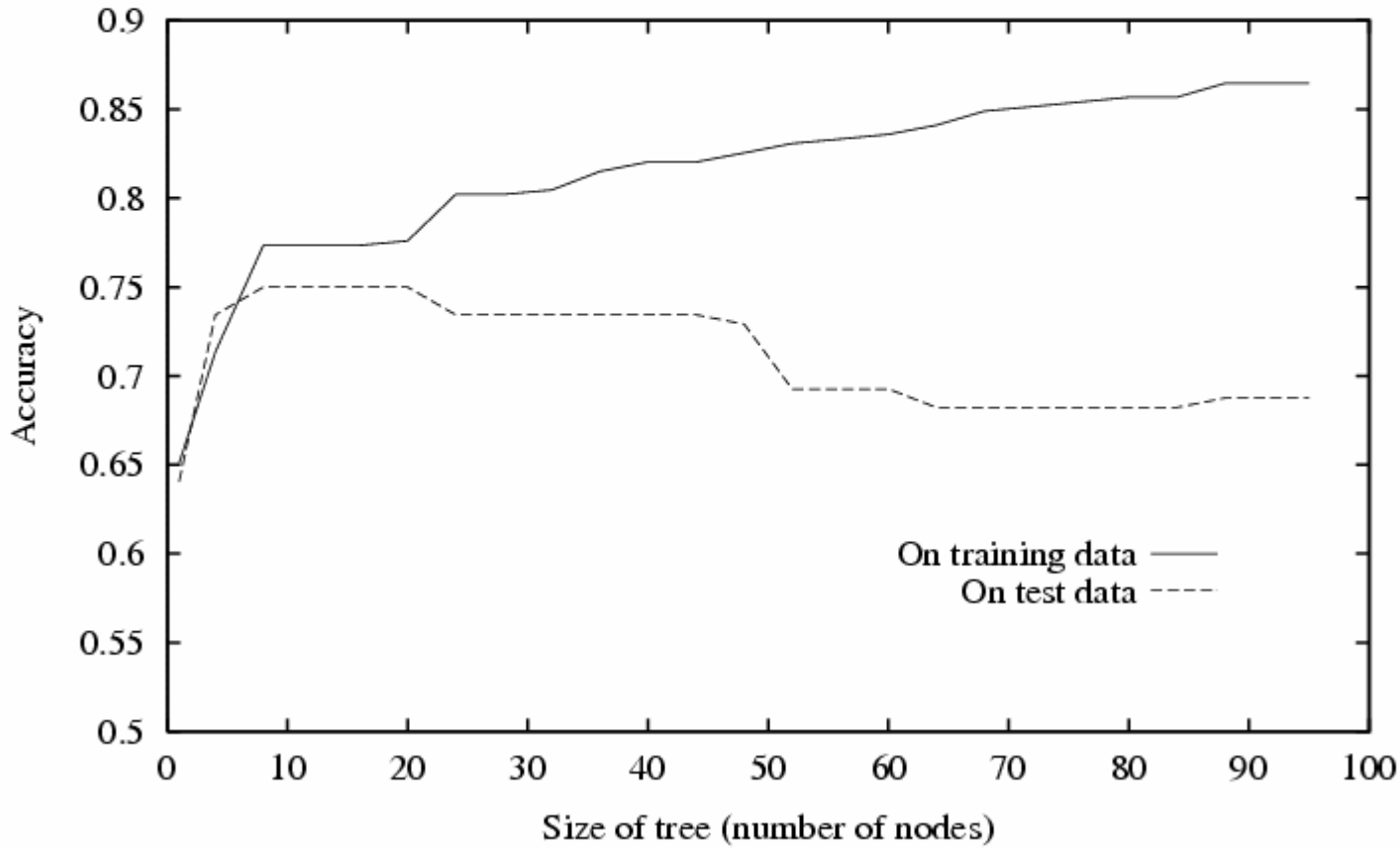
Consider adding noisy training example #15:

$Sunny,\ Hot,\ Normal,\ Strong,\ PlayTennis = No$

What effect on earlier tree?



Overfitting in Decision Tree Learning



Avoiding Overfitting

How can we avoid overfitting?

- stop growing when data split not statistically significant
- grow full tree, then post-prune

How to select "best" tree:

- Measure performance over training data
- Measure performance over separate validation data set
- MDL: minimize
  $size(tree) + size(misclassifications(tree))$

Minimum Description Length Principle

Occam's razor: prefer the shortest hypothesis

MDL: prefer the hypothesis h that minimizes

$$h_{MDL} = \underset{h \in H}{\operatorname{argmin}} L_{C_1}(h) + L_{C_2}(D|h)$$

where $L_C(x)$ is the description length of x under encoding C

Example: H = decision trees, D = training data labels

- $L_{C_1}(h)$ is # bits to describe tree h
- $L_{C_2}(D|h)$ is # bits to describe D given h
  - Note $L_{C_2}(D|h) = 0$ if examples classified perfectly by h. Need only describe exceptions
- Hence h_{MDL} trades off tree size for training errors

Assume X values known, labels Y encoded

Minimum Description Length Principle

$$
\begin{aligned}
h_{MAP} &= \arg\max_{h \in H} P(D|h)P(h) \\
&= \arg\max_{h \in H} \log_2 P(D|h) + \log_2 P(h) \\
&= \arg\min_{h \in H} -\log_2 P(D|h) - \log_2 P(h) \quad (1)
\end{aligned}
$$

Interesting fact from information theory:

> The optimal (shortest expected coding length) code for an event with probability p is $-\log_2 p$ bits.

So interpret (1):

- $-\log_2 P(h)$ is length of h under optimal code
- $-\log_2 P(D|h)$ is length of D given h under optimal code

$\rightarrow$ prefer the hypothesis that minimizes

$$length(h) + length(misclassifications)$$

Reduced-Error Pruning

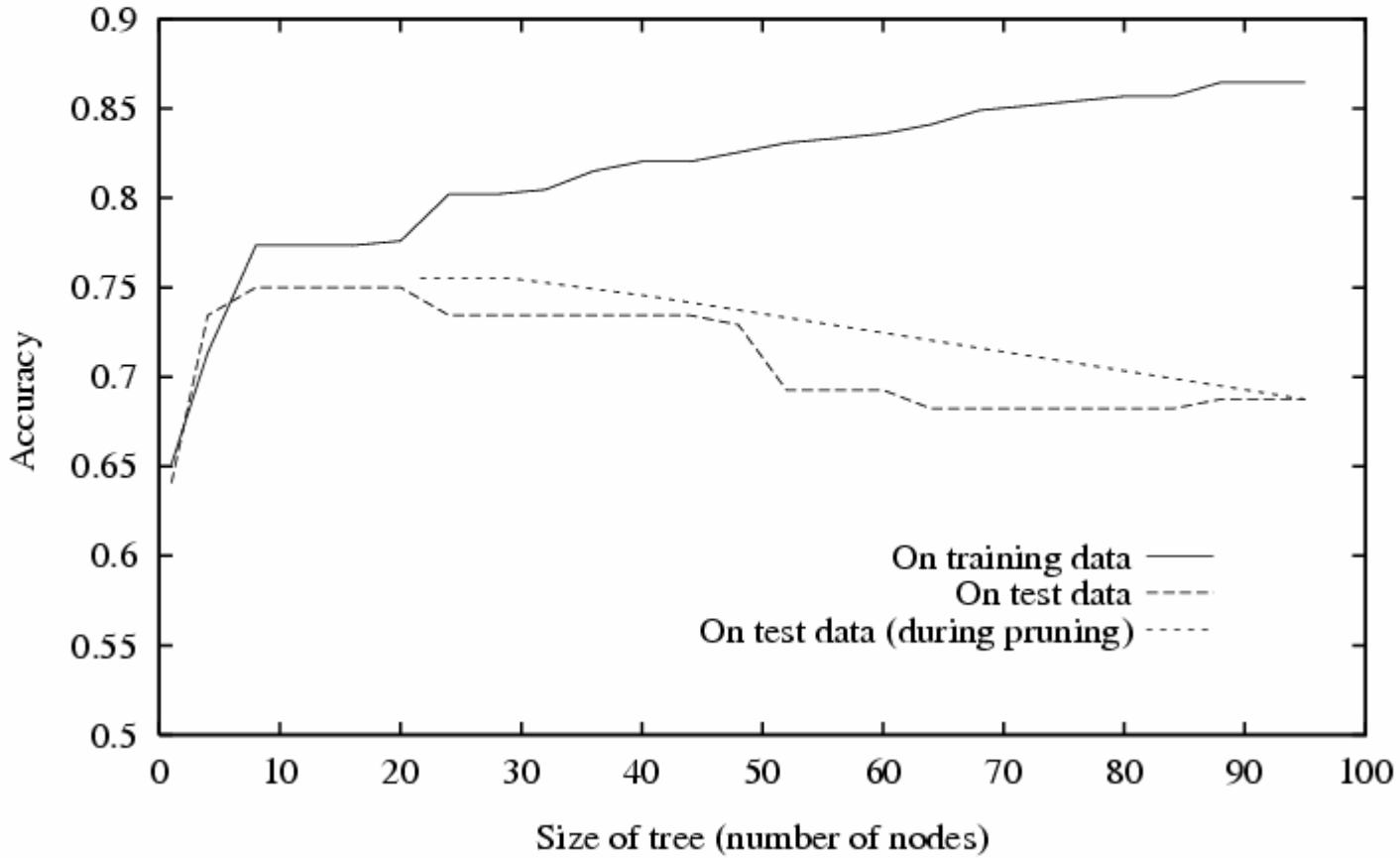
Split data into *training* and *validation* set

Do until further pruning is harmful:

1. Evaluate impact on *validation* set of pruning each possible node (plus those below it)
2. Greedily remove the one that most improves *validation* set accuracy

- produces smallest version of most accurate subtree
- What if data is limited?

Effect of Reduced-Error Pruning

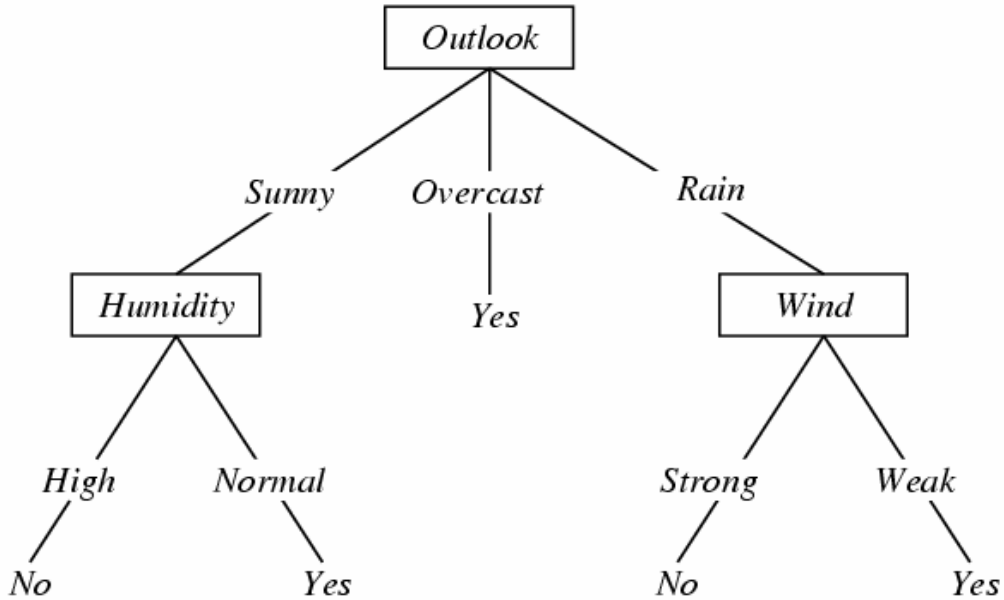


Rule Post-Pruning

1. Convert tree to equivalent set of rules
2. Prune each rule independently of others
3. Sort final rules into desired sequence for use

Perhaps most frequently used method (e.g., C4.5)

Converting A Tree to Rules



IF $(Outlook = Sunny) \wedge (Humidity = High)$
THEN $PlayTennis = No$

IF $(Outlook = Sunny) \wedge (Humidity = Normal)$
THEN $PlayTennis = Yes$

Continuous Valued Attributes

Create a discrete attribute to test continuous

- $Temperature = 82.5$
- $(Temperature > 72.3) = t, f$

Temperature:	40	48	60	72	80	90
PlayTennis:	No	No	Yes	Yes	Yes	No

Attributes with Many Values

Problem:

- If attribute has many values, $Gain$ will select it
- Imagine using $Date = Jun\_3\_1996$ as attribute

One approach: use $GainRatio$ instead

$$GainRatio(S, A) \equiv \frac{Gain(S, A)}{SplitInformation(S, A)}$$

$$SplitInformation(S, A) \equiv -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

where S_i is subset of S for which A has value v_i

Unknown Attribute Values

What if some examples missing values of A?
Use training example anyway, sort through tree

- If node n tests A, assign most common value of A among other examples sorted to node n
- assign most common value of A among other examples with same target value
- assign probability p_i to each possible value v_i of A
  - assign fraction p_i of example to each descendant in tree

Classify new examples in same fashion

Boosting [Schapire, 1989]

- Idea: given a weak learner, run it multiple times on (reweighted) training data, then let learned classifiers vote
- On each iteration, weight each training example by how incorrectly it was classified

- Practically useful
- Theoretically interesting

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X$, $y_i \in Y = \{-1, +1\}$
Initialize $D_1(i) = 1/m$.
For $t = 1, \ldots, T$:

- Train base learner using distribution D_t.
- Get base classifier $h_t : X \to \mathbb{R}$.
- Choose $\alpha_t \in \mathbb{R}$.
- Update:

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where Z_t is a normalization factor

$$Z_t = \sum_{i=1}^{m} D_t(i) \exp(-\alpha_t y_i h_t(x_i))$$

Output the final classifier:

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

Figure 1: The boosting algorithm AdaBoost.

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X, y_i \in Y = \{-1, +1\}$
Initialize $D_1(i) = 1/m$.
For $t = 1, \ldots, T$:

- Train base learner using distribution D_t.
- Get base classifier $h_t : X \to \mathbb{R}$.
- Choose $\alpha_t \in \mathbb{R}$.
- Update:

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where Z_t is a normalization factor

$$Z_t = \sum_{i=1}^{m} D_t(i) \exp(-\alpha_t y_i h_t(x_i))$$

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

Output the final classifier:

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

Figure 1: The boosting algorithm AdaBoost.

What α_t to choose for hypothesis h_t?

Training error of final classifier is bounded by:

$$\frac{1}{m}\sum_{i=1}^{m}\delta(H(x_i)\neq y_i) \leq \frac{1}{m}\sum_{i}\exp(-y_i f(x_i)) = \prod_t Z_t$$

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

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

$$Z_t = \sum_{i=1}^{m} D_t(i)\exp(-\alpha_t y_i h_t(x_i))$$

What α_t to choose for hypothesis h_t?

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

$$Z_t = \sum_{i=1}^{m} D_t(i) \exp(-\alpha_t y_i h_t(x_i))$$

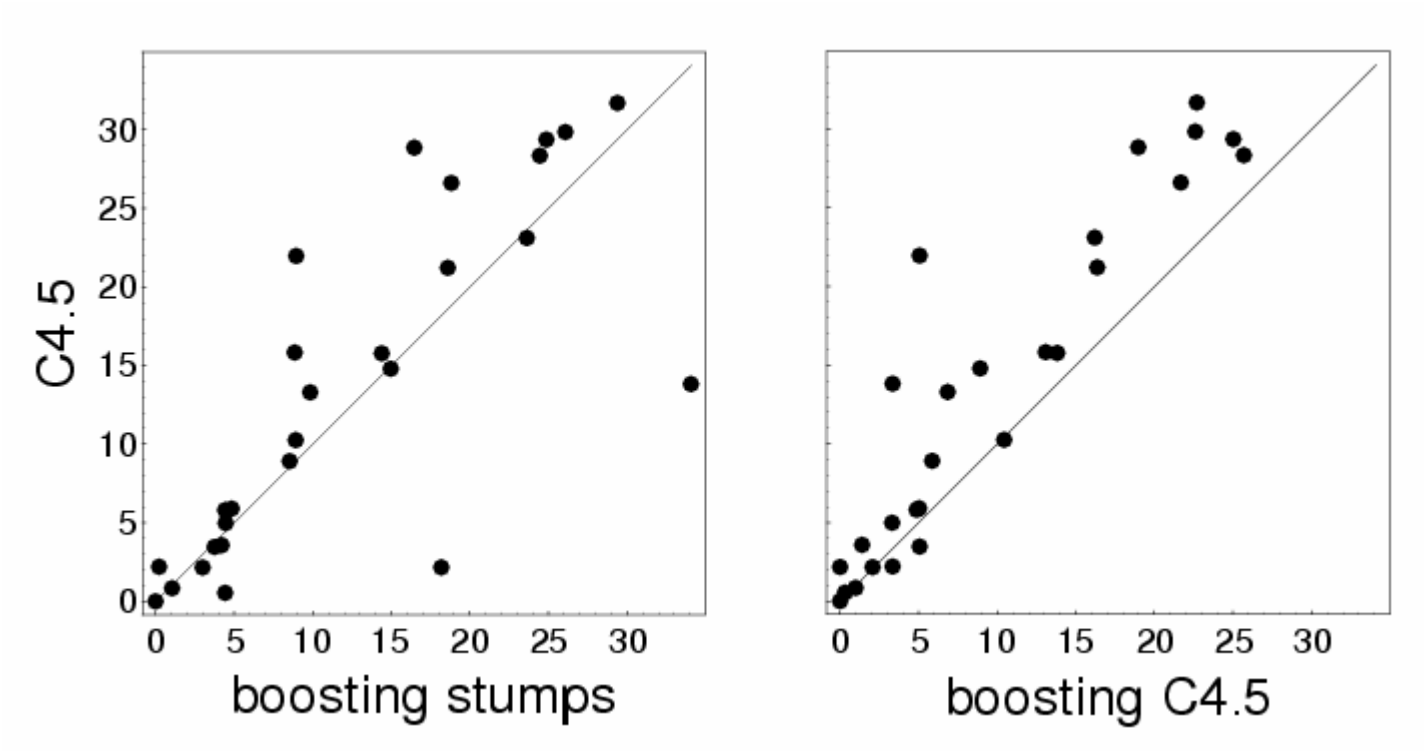
For boolean target function, this is accomplished by:

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

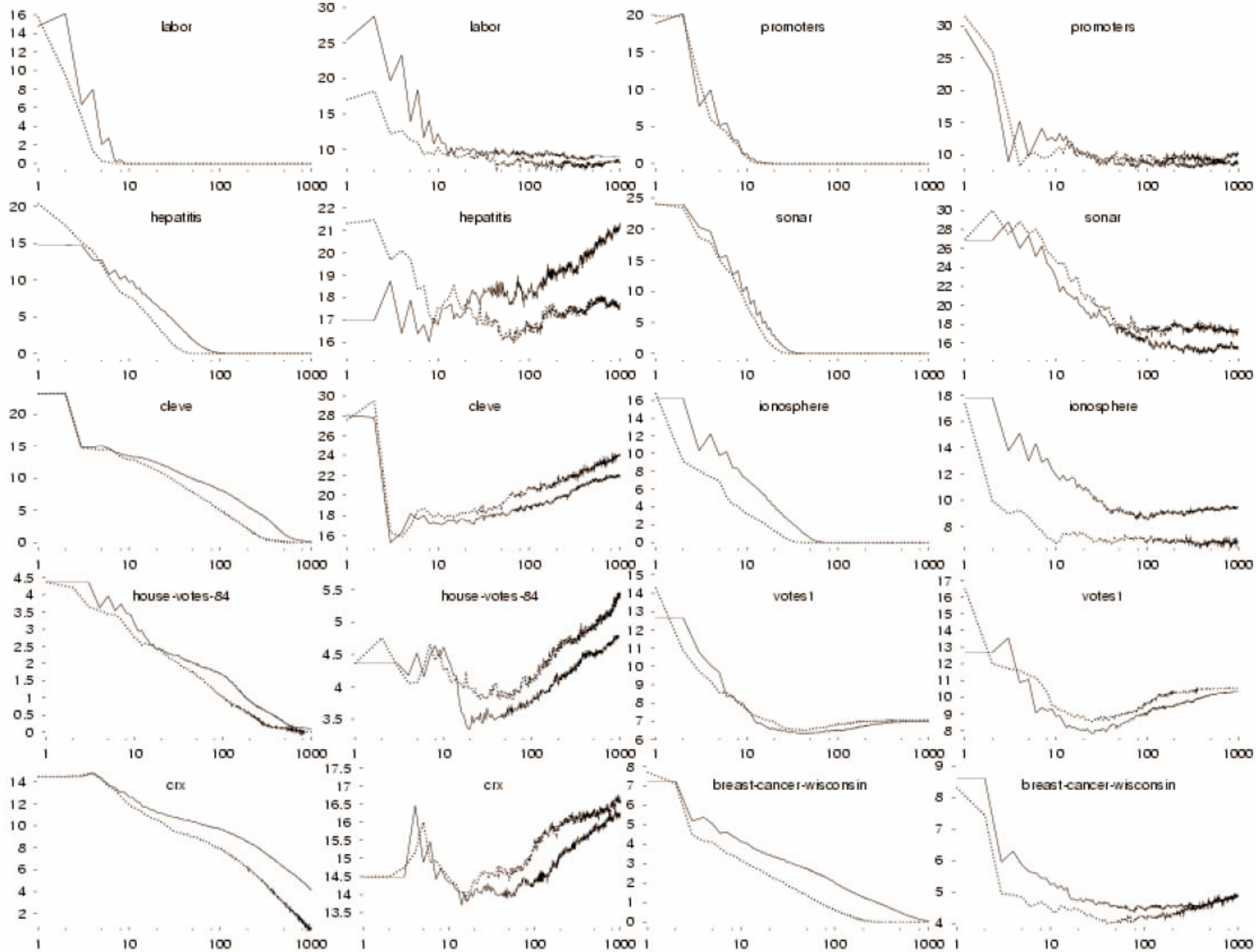
$$\epsilon_t = \frac{1}{\sum_{i=1}^{n} D_t(i)} \sum_{i=1}^{m} D_t(i) \delta(h_t(x_i) \neq y_i)$$

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:

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

And tries to maximize data likelihood:

$$P(data|H) = \prod_{i=1}^{m} \frac{1}{1 + \exp(-y_i f(x_i))}$$

Equivalent to minimizing log loss

$$\sum_{i=1}^{m} \ln(1 + \exp(-y_i f(x_i)))$$

Boosting and Logistic Regression

Logistic regression assumes:

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

And tries to maximize conditional data likelihood:

$$\prod_{i=1}^{m} \frac{1}{1 + \exp(-y_i f(x_i))}$$

Equivalent to minimizing log loss

$$\sum_{i=1}^{m} \ln(1 + \exp(-y_i f(x_i)))$$

Boosting minimizes similar loss function!!

$$\frac{1}{m} \sum_i \exp(-y_i f(x_i)) = \prod_t Z_t$$

Logistic regression and Boosting

- Minimize loss fn

$$\sum_{i=1}^{m} \ln(1 + \exp(-y_i f(x_i)))$$

- Define

$$f(x) = \sum_j w_j x_j$$

where x_j predefined

- Minimize loss fn

$$\sum_{i=1}^{m} \exp(-y_i f(x_i))$$

- Define

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

where $h(x_i)$ defined dynamically to fit data

- Weights α_j learned incrementally

What you should know:

- Decision trees
  - ID3, C4.5
  - Rule extraction from trees
  - Overfitting and tree/rule post-pruning
  - Extensions…
- Minimum description length approach
  - And it's Bayesian interpretation
- Boosting
  - Practical approach to improving accuracy
  - Exponential loss function;
  - relationship to logistic regression