Learning Decision Trees

- Framework
  - Classification Learning
  - Def'n: Decision Trees
- Algorithm for Learning Decision Tree
  - Entropy
  - Inductive Bias (Occam's Razor)
- Evaluation
  - CrossValidation
- Overfitting
  - PostPruning
- Topics:
  - k-ary attribute values
  - Real attribute values
  - Other splitting criteria
  - Attribute Cost
  - Missing Values
  - ...

Supervised Learning

- Many forms of machine learning ≈ *function approximation*:

Given:	Set of **training examples**: $\{(x_1, f(x_1)), \ldots, (x_m, f(x_m))\}$ Space of **hypotheses** H
Find:	Hypothesis $h \in H$ that is good approx'n to f (ie, s.t. $h(x) \approx f(x)$ for most x in space)

Note: $f : X \mapsto R$ not known
(f need not be in H)

Want h that works well throughout "instance space" X
. . . training examples only small subset

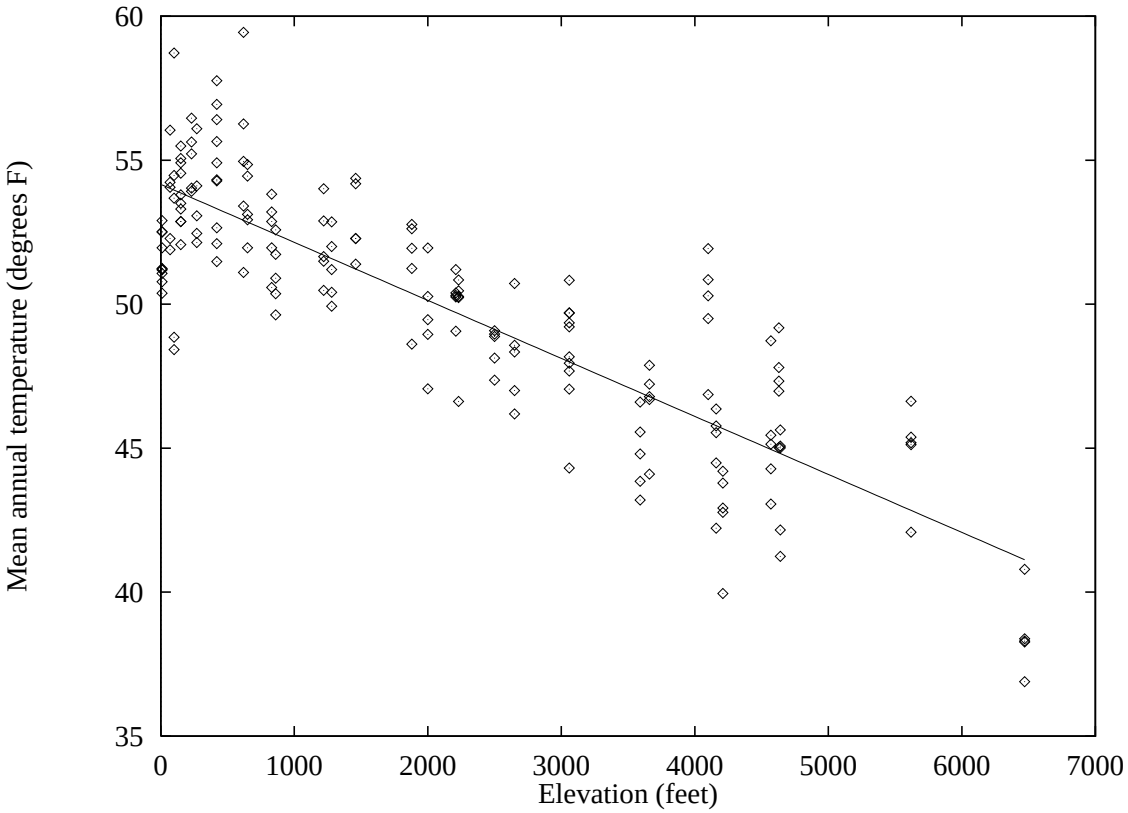
Typically x_i represented by vector of **feature values**:

$$x_i = \langle x_i^{(1)}, \ldots, x_n^{(n)} \rangle \in X$$

. . . perhaps $x_i \in \Re^n$, or discrete, or combination, or . . .

- Kinds of Functions
  - **Real-valued Functions:** $f(x) \in \Re$
  - **Probability Distributions:** $P_f(\, y \,|\, x \,) = P(\, f(x) \,|\, x \,)$
  - **Classifiers:** $f(x) \in \{c_1, \ldots, c_k\}$
    . . . perhaps $\{c_1, c_2\} = \{T, F\}$

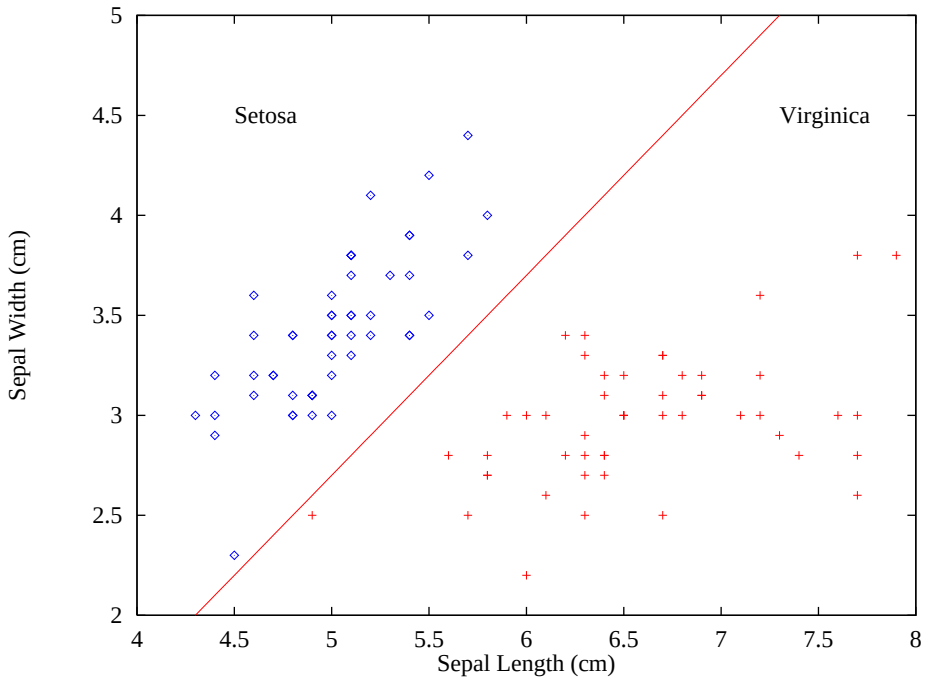
Real-Valued Functions: Regression



- **Unknown function:** maps elevation to mean annual temperature
- **Examples:** 175 weather stations with known elevation and temperature
- **Hypothesis space:** All functions of form

$$temp = w_0 + w_1 \cdot elev$$

Discrete-Valued Functions: Classification



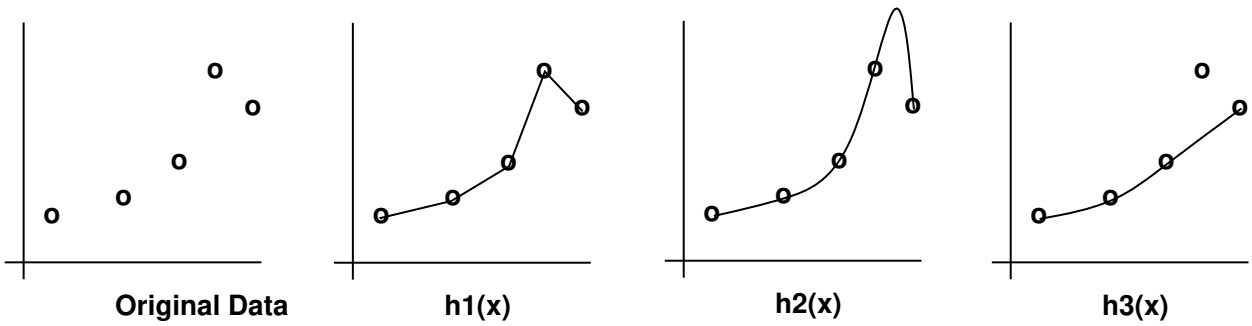
- **Unknown function:** maps from flower measurements to species of flower
- **Examples:** 100 flowers measured and classified by R.A. Fisher
- **Hypothesis Space:** All linear discriminators of form

$$h(x) = \begin{cases} Setosa & \text{if } w_0 + w_1 \cdot x.SepalWidth + w_2 \cdot x.SepalLength > 0 \\ Virginica & \text{otherwise} \end{cases}$$

Classification Tasks

- What is it?
  - Type of flower?
  - Star, Galaxy, or GasCloud?
  - Good vs Bad Stock?
  - Does patient has disease X, given symptoms Z?
  - ...
- Which action to take?
  - Dig here, or not?
  - Should I buy stockX?
  - Is treatment Y effective, for disease X?
  - Which move in chess?
  - Should Switch7 be turned on?
  - Will meeting be 1/2/hour? ...1hour?
  - Will X wait for restaurant?
  - ...

Goal: Generalization



- Which of $h1$, $h2$, $h3$ is best?
- Depends. . .
  - Is data noisy?
  - What is known about $f(\cdot)$?
    . . . piece-wise linear, smooth, . . .

Goal: Good performance over *distribution* including UNSEEN data

$\neq$ simply fitting observed data
(Otherwise: just memorize observations!)

$\Rightarrow$ Need "bias". . .

"Bias" of Learning Algorithms

- Learning algorithm embodies some "bias"
  to prefer one hypothesis over another

  . . . ideally: matched to assumptions/environment

- Two types of bias:
  - **restriction bias or language bias**
    Specifies what hypothesis space is searched
    (Eg, Gaussian, Mixture of Gaussians, DecisionTrees, Piece-wise linear functions, . . .)
  - **preference bias or search bias**
    specifies how hypothesis space is explored
    $\Rightarrow$ leads to different (first) answer

- Tradeoffs (similar to reasoning)
  More expressive "language"
  $\Rightarrow$ Harder to find good hypothesis

  (Size of H, VCDimension of H)

Decision Trees

Hypothesis space is. . .

- **Variable Size**: Can represent *any* boolean function
- **Deterministic**
- **Discrete and Continuous Parameters**

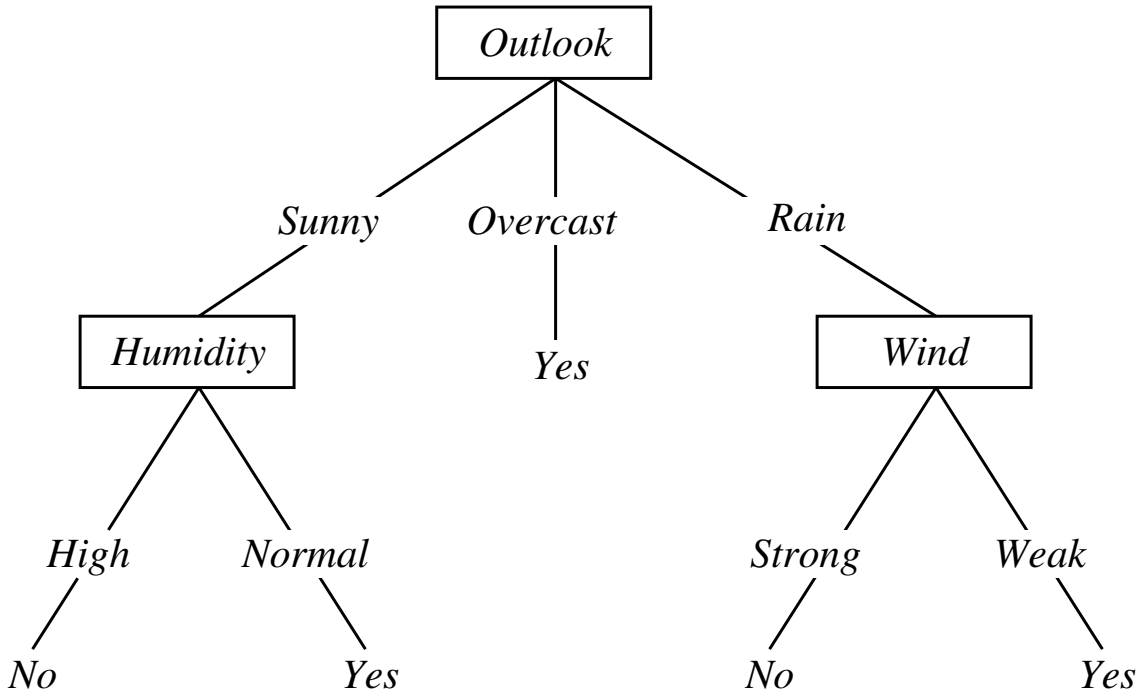
Learning algorithm is. . .

- **Constructive Search**: Build tree by adding nodes
- **Eager**
- **Batch** (although $\exists$ online algorithms)

Decision Tree Hypothesis Space

Internal nodes test value of features x_j
and branch according to results of test

Leaf nodes specify class $h(x)$



Instance: $\langle$ Outlook = Sunny, Temperature = Hot, Humidity = High, Wind = Strong $\rangle$

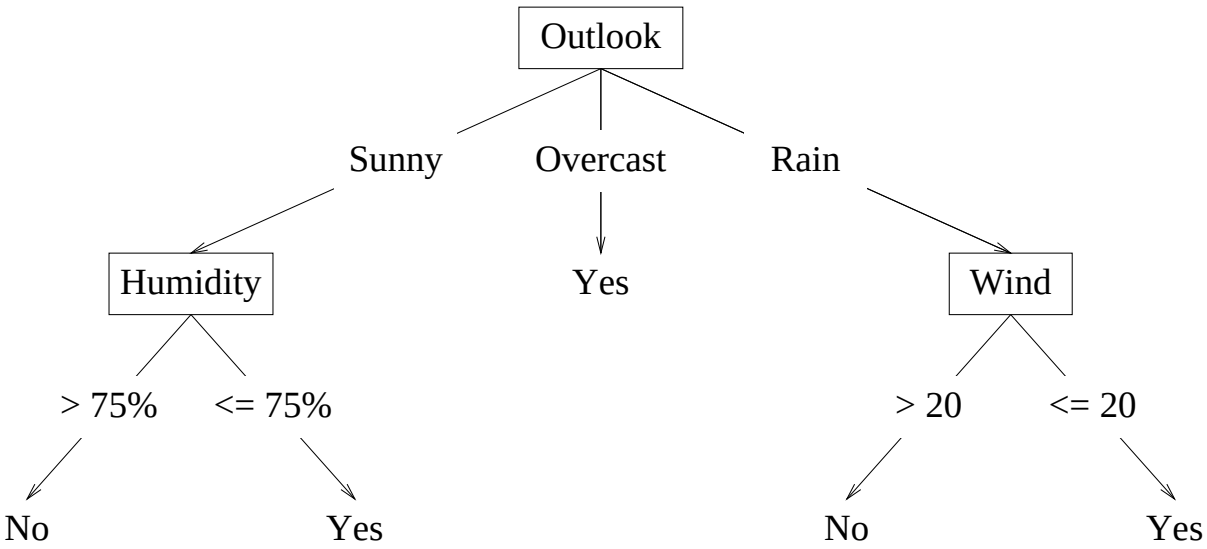
classified as "No"
(`Temperature` is irrelevant)

Easy to use in Classification
Divide-and-Conquer
. . . keep splitting based on some test

Continuous Features

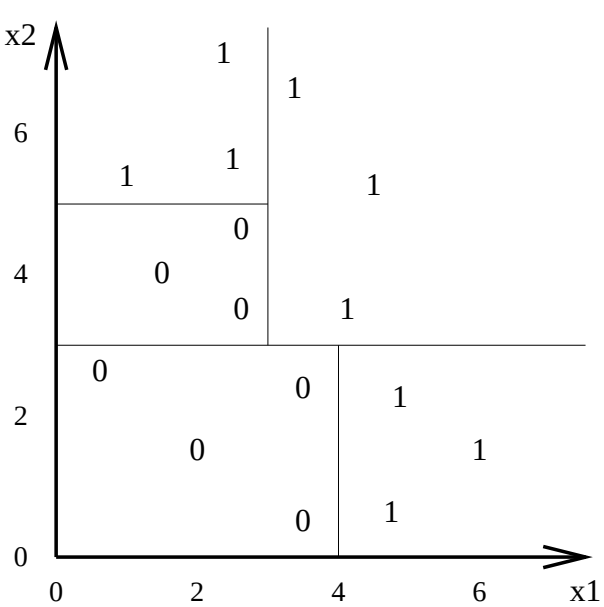
If feature is continuous:

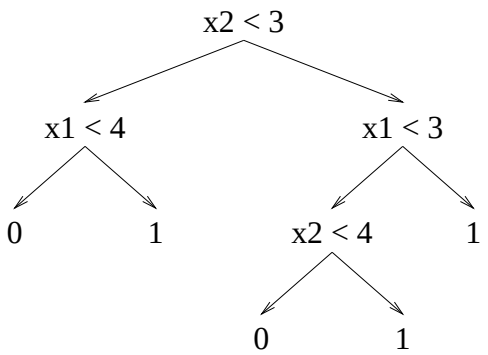
internal nodes may test value against threshold



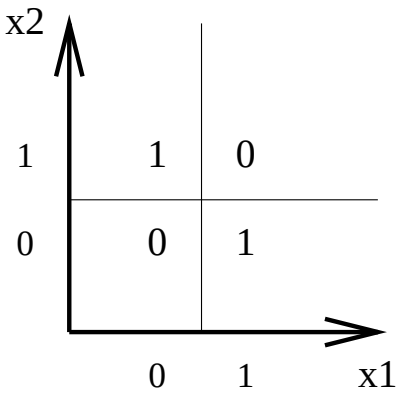
Decision Tree Decision Boundaries

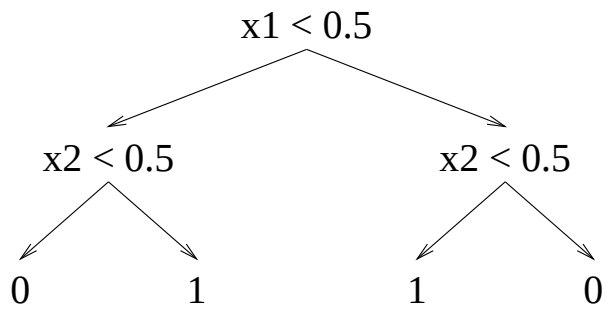
- Decision trees divide feature space into axis-parallel rectangles, labeling each rectangle with one class





Can Represent Any Boolean Function





- $\wedge$, $\vee$, XOR
  $(A \wedge B) \vee (C \wedge \neg D \wedge E)$
  M of N
- . . . but may require exponentially many nodes. . .

Variable-Size Hypothesis Space

Can "grow" hypothesis space by increasing number of nodes

- **depth 1** ("decision stump"): represent any boolean function of one feature
- **depth 2**: Any boolean function of two features; some boolean functions involving three features
  $(x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_3)$
- etc.

Using Decision Trees

Requirements: Instances represented by Attribute-Value pairs

"`Bar = Yes`", "`Size = Large`"

"`Type = French`", "`Temp = 82.6`", ...

(Boolean, Discrete, Nominal, Continuous)

Can handle:

+ Disjunctive descriptions
+ Errors in Training data (class, attributes)
+ Missing attribute values in data

Our focus:

Target function output is discrete

(DT also work for continuous outputs [regression])

Eg: + Equipment or medical diagnosis
+ Credit risk analysis
+ Modeling calendar scheduling preferences

(Simplified) Algorithm for Learning Decision Tree

```
GrowTree(S)
   if (y = 0) for all ⟨x, y⟩ ∈ S return new leaf(0)
   if (y = 1) for all ⟨x, y⟩ ∈ S return new leaf(1)
   /* else */

   x_j = ChooseBestAttribute(S)
   S_0 = all ⟨x, y⟩ ∈ S with x_j = 0
   S_1 = all ⟨x, y⟩ ∈ S with x_j = 1
   return new node(x_j, GrowTree(S_0), GrowTree(S_1) )
```

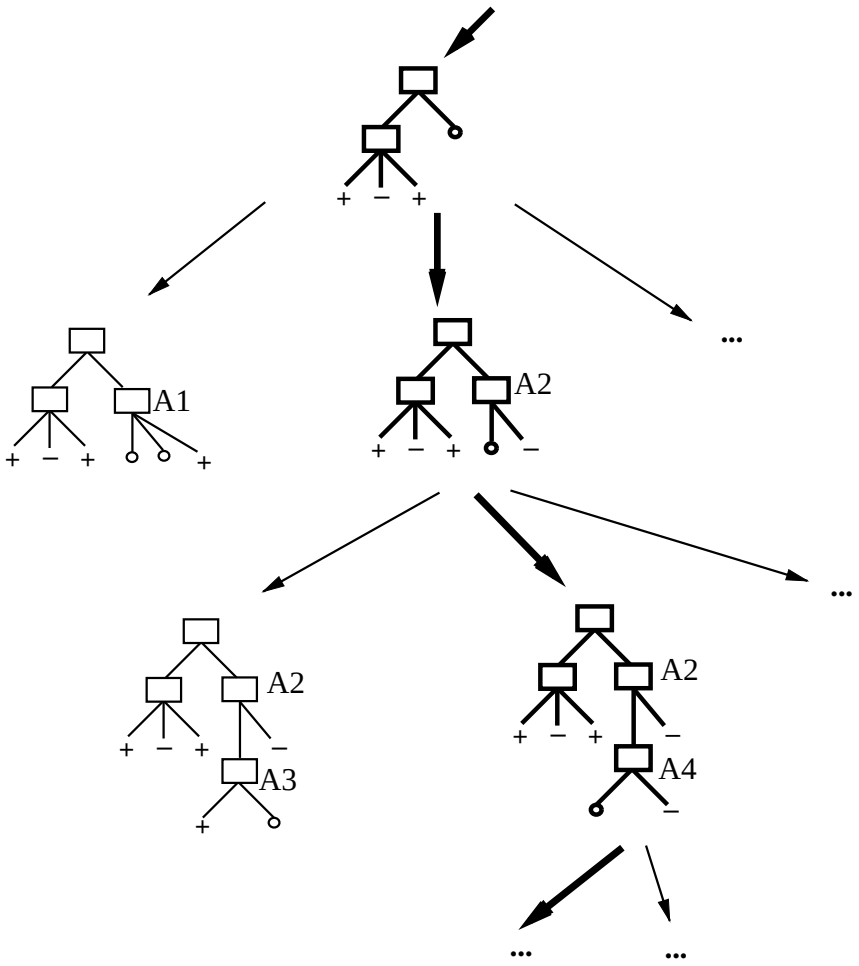
Many fields independently discovered this learning alg. . .

Algorithm for Learning Decision Trees

function DECISION-TREE-LEARNING(*examples, attributes, default*) **returns** a decision tree
  inputs: *examples*, set of examples
    attributes, set of attributes
    default, default value for the goal predicate

  if *examples* is empty **then return** *default*
  else if all *examples* have the same classification **then return** the classification
  else if *attributes* is empty **then return** MAJORITY-VALUE(*examples*)
  else
    best $\leftarrow$ CHOOSE-ATTRIBUTE(*attributes*, *examples*)
    tree $\leftarrow$ a new decision tree with root test *best*
    for each value v_i of *best* **do**
      $examples_i \leftarrow$ {elements of *examples* with *best* = v_i}
      subtree $\leftarrow$ DECISION-TREE-LEARNING($examples_i$, *attributes* $-$ *best*,
                                MAJORITY-VALUE(*examples*))
      add a branch to *tree* with label v_i and subtree *subtree*
    end
    return *tree*

Search for Good Decision Tree



- Local search – expanding one leaf at-a-time
  (no backtracking)
- Trivial to find tree that perfectly "fits" training data

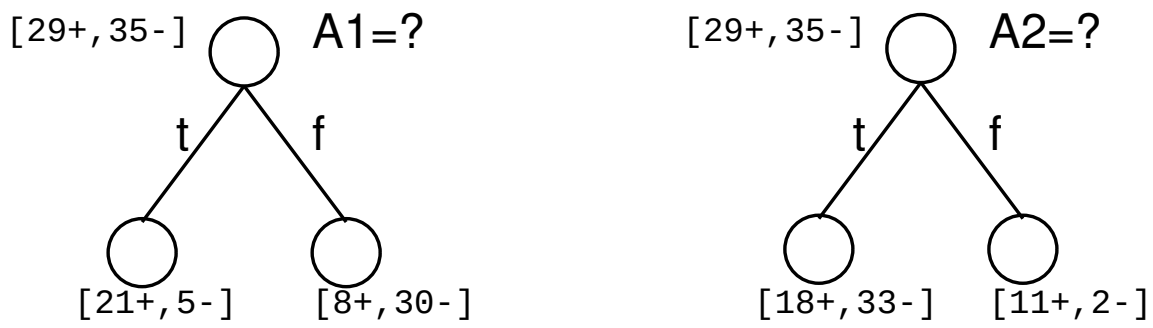
but. . . this is NOT necessarily best tree
NP-hard to find *smallest* tree that fits data

Issues in Design of Decision Tree Learner

- What attribute to split on?
- When to stop?
- Should tree by pruned?
- How to evaluate classifier? (decision tree)
  . . . learner?

Choosing Best Splitting Test

- How to choose best feature to split?



Goal: Seek **short tree** consistent with samples (Occam's razor)

- Select **most informative** feature at each step

  . . . one that best splits (classifies) examples

- Ideas:
  1. Simple 1-step look-ahead
  2. **information theory**

Choosing Best Splitting Test

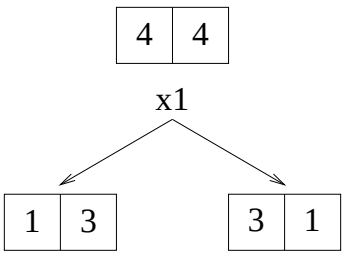
Idea1: Perform 1-step look-ahead search:
choose attribute giving lowest error
on training data.

```
ChooseBestAttribute(S)
 x_j = argmin_{x_j}{Score(x_j, S)}
 return( x_j )
```

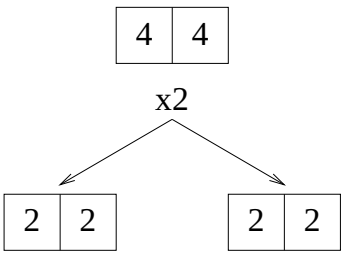
```
Score(x, S)

   S_0 = all ⟨x, y⟩ ∈ S with x = 0
   S_1 = all ⟨x, y⟩ ∈ S with x = 1
   y_0 = most common y-value in S_0
   y_1 = most common y-value in S_1
   J_0 = number of ⟨x, y⟩ ∈ S_0 with y ≠ y_0
   J_1 = number of ⟨x, y⟩ ∈ S_1 with y ≠ y_1
   J = J_0 + J_1 (total errors if split on feature x)
   return J
```

Choosing Best Attribute: An Example

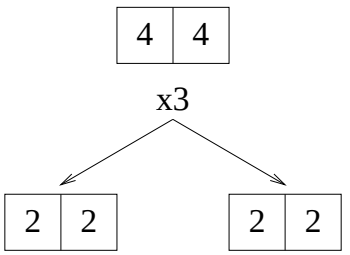
x_1	x_2	x_3	y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0



J=2



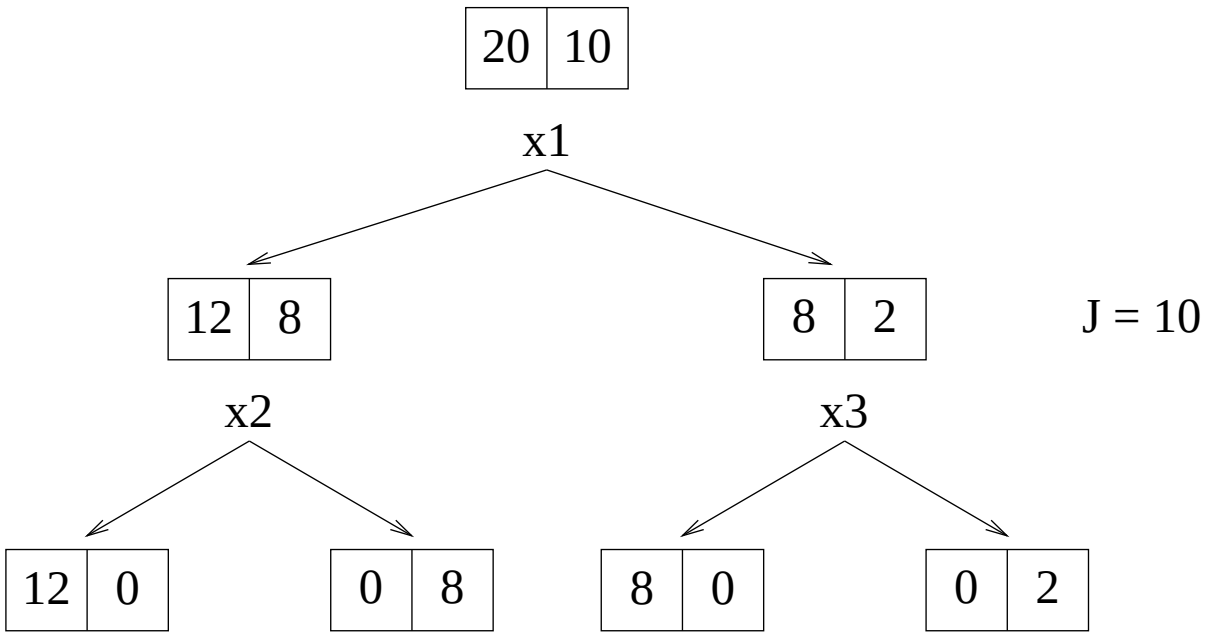
J=4



J=4

Problem with Simple ChoiceFunction

This measure does not always work well
because it does not detect when making "progress" toward a good tree:



Idea2: Entropy

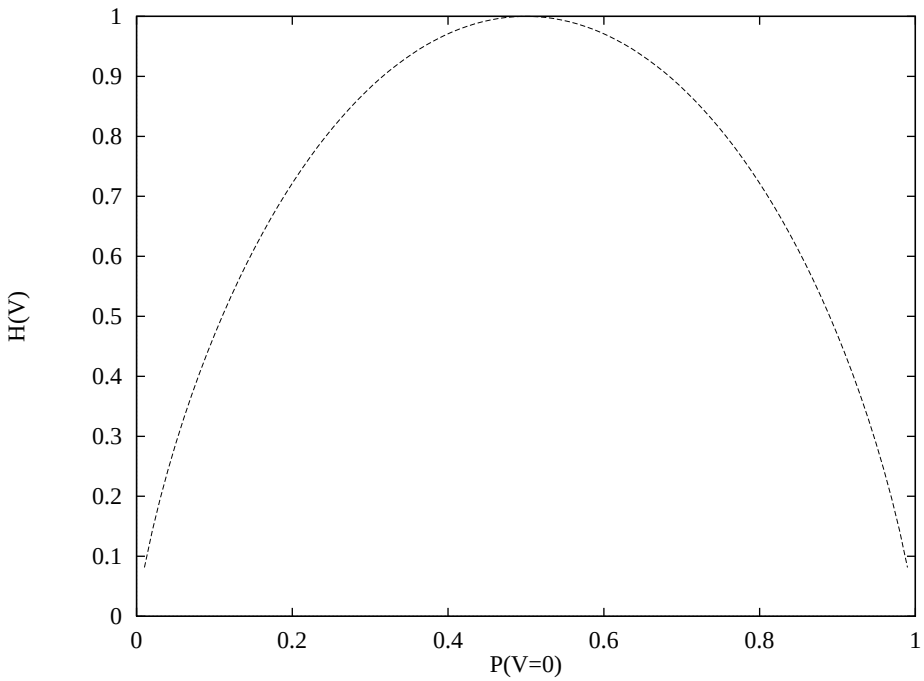
- *Entropy* of V:

$$H(V) \quad = \quad -\sum_{v_i} P(\, V = v_i \,) \log_2 P(\, V = v_i \,)$$

$\equiv$ # of **bits** needed to obtain full info

average surprise of result of one "trial" of V

- Entropy $\approx$ measure of uncertainty



Examples of Entropy

- Fair coin:

  $H(\frac{1}{2},\ \frac{1}{2})\ =\ -\frac{1}{2}\ \log_2(\frac{1}{2}) - \frac{1}{2}\ \log_2(\frac{1}{2})\ =\ 1\ \text{bit}$

  (ie, need 1 bit to convey the outcome of coin flip)

- Biased coin:

  $H(\frac{1}{100},\ \frac{99}{100})\ =\ -\frac{1}{100}\ \log_2(\frac{1}{100}) - \frac{99}{100}\ \log_2(\frac{99}{100}) = 0.08\ \text{bit}$

- As $P(\ heads\ )\ \rightsquigarrow\ 1$,

  info of actual outcome $\rightsquigarrow\ 0$

  $H(0,1)\ =\ H(1,0)\ =\ 0\ \text{bits}$

  ie, no uncertainty left in source

  $(0 \cdot log_2(0) = 0)$

Prefer Low Entropy Leaves

- Spse use decision tree h
  to classify (unlabeled) test example x

  Follow path down to leaf ℓ.
  What classification?

- Consider *training examples* that reached ℓ:

  If all had same class c
  $\Rightarrow$ label x as c (ie, entropy is 0)

  If 1/2 are $+$; 1/2 are $-$
  $\Rightarrow$ label x as ??? (ie, entropy is 1)

- On reaching leaf ℓ with entropy H_ℓ,
  uncertainty w/label is H_ℓ
  (ie, need H_ℓ more bits to decide on class)

  $\Rightarrow$ prefer leaf with **LOW entropy**

Entropy of Collection of Examples

- Don't have **exact** probabilities
  but **training data** provides estimates of probabilities:

Given training set with $\left\{ \begin{array}{c} p \text{ positive} \\ n \text{ negative} \end{array} \right\}$ examples:

$$H(\tfrac{p}{p+n}, \tfrac{n}{p+n}) \;=\; -\tfrac{p}{p+n}\ \log_2(\tfrac{p}{p+n}) \;-\; \tfrac{n}{p+n}\ \log_2(\tfrac{n}{p+n})$$

Eg: wrt 12 restaurant examples, S:

$$p = n = 6 \quad \Rightarrow \quad H(\tfrac{1}{2}, \tfrac{1}{2}) = 1 \ \ bit$$

so need 1 bit of info to classify
randomly picked example from S

Training Examples

Day	Outlook	Temper.	Humidity	Wind	PlayTennis
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

- 4 discrete attributes
- "`Yes/No`" classification

Task: Learn $f_{PT}(Out,\ Temp,\ Humid,\ Wind) \quad \in \{\texttt{Yes}, \texttt{No}\}$

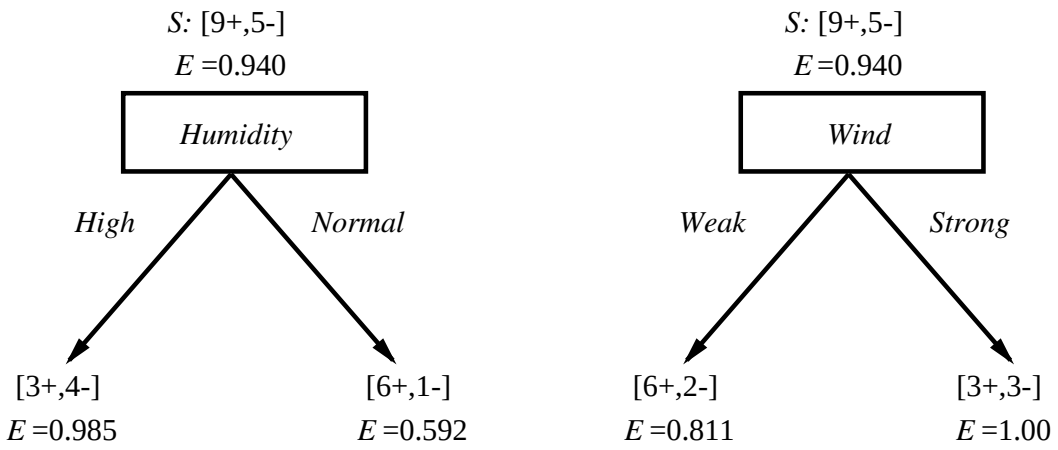
Information Gain

$$Gain(S, A) \quad = \quad \text{expected reduction in entropy due to splitting on } A$$

$$\equiv \; Entropy(S) \; - \sum_{v_i \in Values(A)} \frac{|S_{v_i}|}{|S|} Entropy(S_{v_i})$$

$$\equiv \; H\left(\frac{p}{p+n}, \frac{n}{p+n}\right) - \sum_{i=1}^{v} \frac{p_i^{(A)} + n_i^{(A)}}{p+n} \; H\left(\frac{p_i^{(A)}}{p_i^{(A)} + n_i^{(A)}}, \frac{n_i^{(A)}}{p_i^{(A)} + n_i^{(A)}}\right)$$

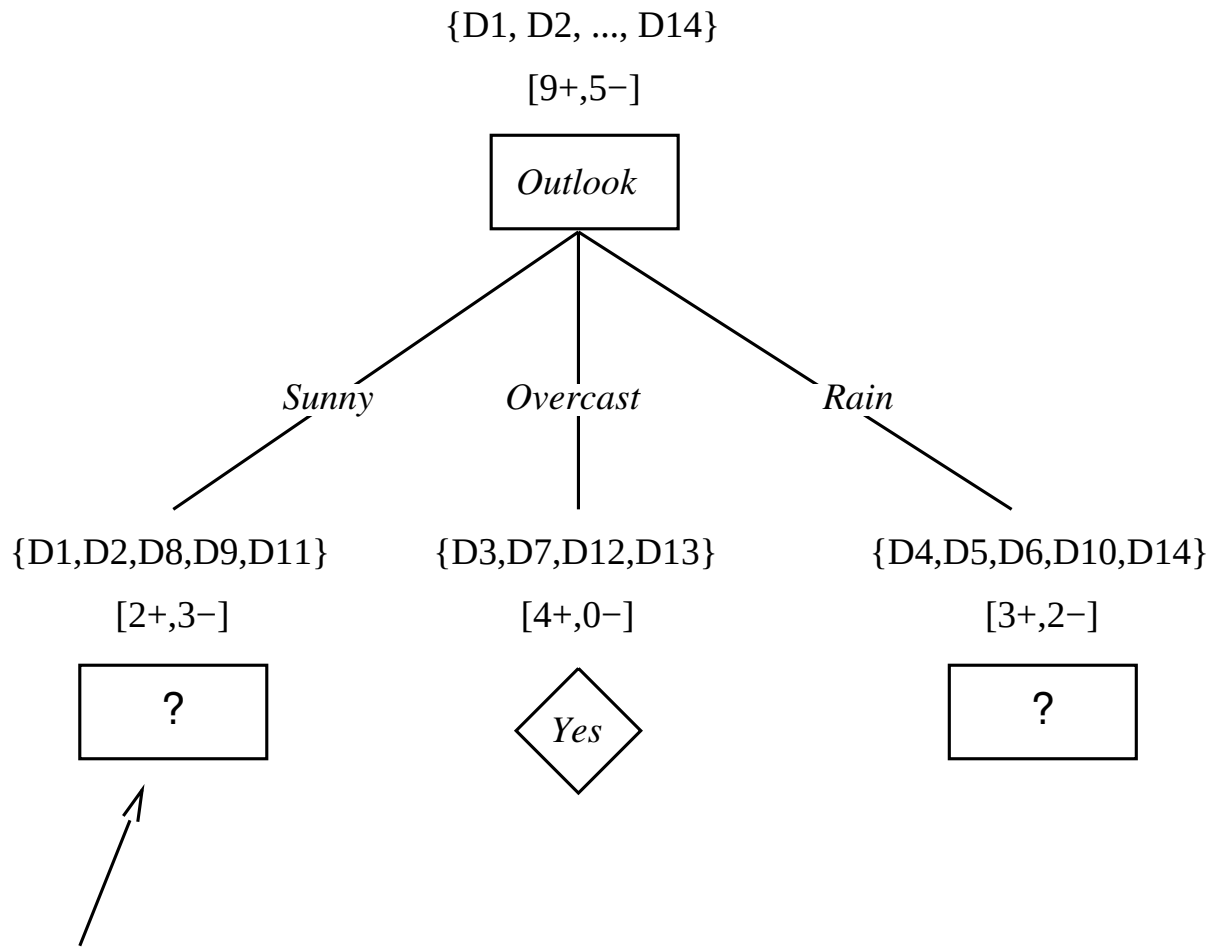
Which attribute is the best classifier?



Gain (S, Humidity)
= .940 - (7/14).985 - (7/14).592
= .151

Gain (S, Wind)
= .940 - (8/14).811 - (6/14)1.0
= .048

. . . as tree is built . . .



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

Maximize Information Gain

Greedy: Split on attribute that **most** reduces **entropy** (uncertainty) of class, over training examples that reach there.

$\Rightarrow$ Consider information **gain**ed by testing attribute A:

$Gain(A) = H(\frac{p}{p+n}, \frac{n}{p+n}) - Uncert(A)$

$Uncert(A)$ = uncertainty remaining after getting info on attribute A

- Assume A divides training set E into $E_1, \ldots E_v$, where A has v distinct values.

E_i has $\left\{ \begin{array}{l} p_i^{(A)} \text{ positive} \\ n_i^{(A)} \text{ negative} \end{array} \right\}$ examples.

- Entropy of each E_i is $H(\frac{p_i^{(A)}}{p_i^{(A)}+n_i^{(A)}}, \frac{n_i^{(A)}}{p_i^{(A)}+n_i^{(A)}})$
- $Uncert(A)$ = *expected* information content $\Rightarrow$ weigh contribution of each E_i

$$Uncert(A) = \sum_{i=1}^{v} \frac{p_i^{(A)} + n_i^{(A)}}{p+n} \; H\left(\frac{p_i^{(A)}}{p_i^{(A)} + n_i^{(A)}}, \frac{n_i^{(A)}}{p_i^{(A)} + n_i^{(A)}}\right)$$

Comments of Learner

- Hypothesis space is complete!
  $\Rightarrow$ contains target function. . .

- No back tracking

  – Local minima...

- Statisically-based search choices

  – Robust to noisy data...

- Inductive bias: $\approx$ "prefer shortest tree"

Inductive Bias in C4.5

Note H = DecisionTreeClassifiers
is power set of instances X

$\Rightarrow$ Unbiased?

- Not really...
  - Preference for
    short trees,
    trees w/ high info gain attributes near root
  - Here: Bias is `preference` for some hypotheses,
    rather than *restriction* of hypothesis space H
  - **Occam's razor:**
    Prefer shortest hypothesis that fits data

Occam's Razor

Q: Why prefer short hypotheses?

- Argument in favor:
  - Fewer short hyps. than long hyps.
  - $\Rightarrow$ a short hyp that fits data unlikely to be coincidence
  - $\Rightarrow$ a long hyp that fits data might be coincidence

- Argument opposed:
  - $\exists$ many ways to define small sets of hyps

    Eg, all trees with prime number of nodes whose attributes all begin with "Z"

  - What's so special about small sets based on *size* of hypothesis??

Quality of Learned Decision Tree

Good News: Learned decision tree is
PERFECT over training set

Bad News: . . . but not perfect over all instances
Eg, wrong for `WaitEst`$>$60, or `WaitEst`$<$10, or . . .

Q: How bad is this?

A: Probably ok. . .
Goodness based on **Expected Performance**
over distribution of examples

$$err_{P,f}(\,h\,) \;=\; P(\,h(x) \neq f(x)\,)$$

Why Learning May Succeed

Learner L produces classifier $h = L(S)$
that does well on training data S

Why?

---

1. *If example x appears rarely*
($P(x) \approx 0$)
then $h(\cdot)$ not penalized for being wrong.

2. *If x appears alot*
then x probably occurs in training data S

As h does well on S,
$h(x)$ is probably correct on x

Assumption: Distribution is "stationary"
distr. for testing = distr. for training

Evaluating Performance

- Want to estimate $err_{P,f}(h)$

  h's **Generalization Error** over distribution P:

  + to evaluate learner
  + to evaluate classifier
  + to decide among possible classifiers

- Algorithm

  - Collect large set of examples, $S = \{\langle x_i, f(x_i)\rangle\}_i$
  - Randomly divide S into two disjoint sets:
    training set S_1
    test set S_2
  - Run learning algorithm on training set S_1
    to generate hypothesis h_{S_1}
  - Estimate $err_P(h_{S_1})$ using $err_{S_2}(h_{S_1})$
    % of ex's in test set S_2 that h_{S_1} mis-classifies

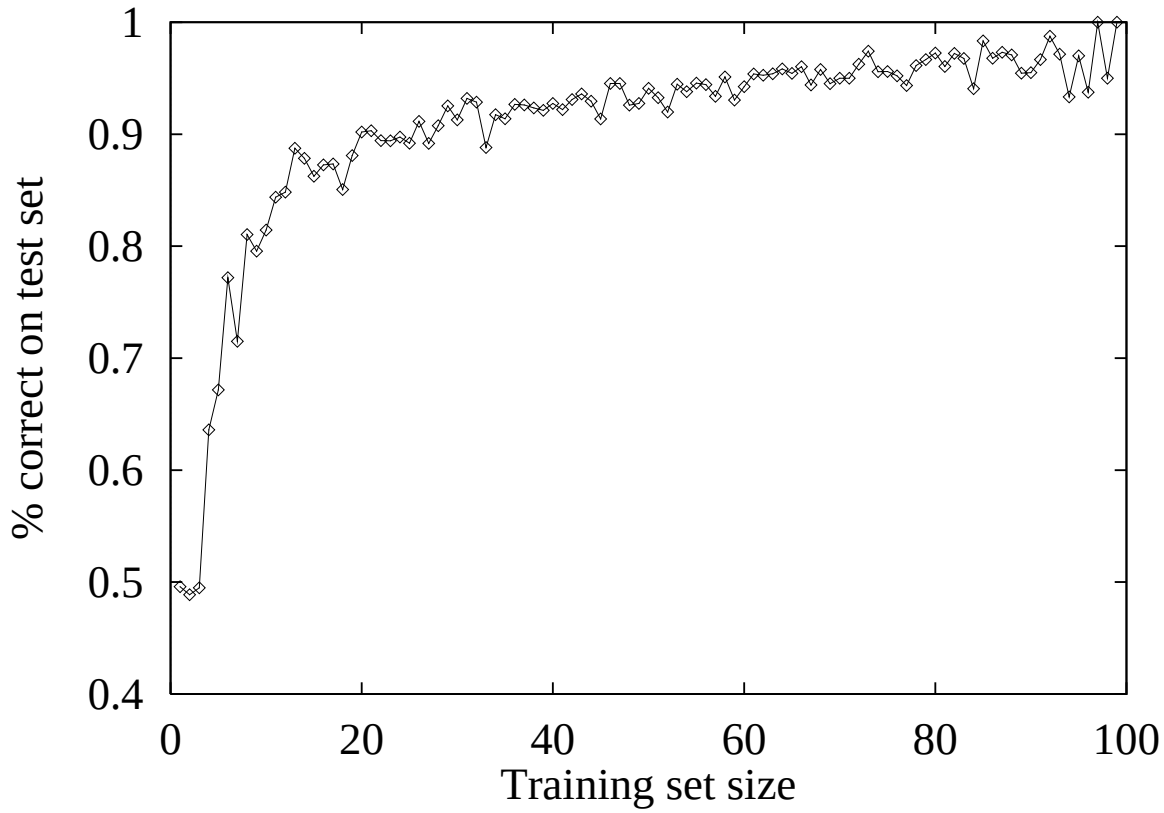
- As we seek "generalization" to *unseen* data,
  want to know $err_P(h_{S_1})$

  Evaluating h_{S_1} on training set S_1 is too optimistic!
  $err_{S_2}(h_{S_1})$ better estimate of $err_P(h_{S_1})$
  than $err_{S_1}(h_{S_1})$ "resubstitution error"

  $\Rightarrow$ test set must be independent!

Learning Curves

Accuracy usually improves with more training examples



Estimating Error

SetUp: Learner L,
using labeled training data S
produces classifer $h = L(S)$.

Estimating $err_D(h)$
$\equiv$ (true) error of h over distribution D

Option1: Use h's empirical error on S
$\widehat{err}_S(h)$ (Resubstitution Error)

$\Rightarrow$ Very **Optimistic**

Option2: Divide S into disjoint S_1, S_2
Train on S_1: computing $h_{S_1} = L(S_1)$
Test on S_2: $\widehat{err}_{S_2}(h_{S_1})$

$\Rightarrow$ Kinda **Pessimistic**
as train on only $|S_1|/|S|$ of the data

Estimating Error: Cross Validation

Option3: "Cross-Validation"

Divide S into k disjoint sets $\{S_1, S_2, \ldots, S_k\}$

For $i = 1..k$ **do**
  Let $TestS_i$ = S_i
  Let $TrainS_i$ = $S - S_i$
  Run L on $TrainS_i$, obtain h_i
  Evaluate h_i on $TestS_i$: $\widehat{err}_{TestS_i, f}(h_i)$
Return Average of $\widehat{err}_{TestS_i, f}(h_i)$

$\Rightarrow$ Less **Pessimistic**
as train on $\frac{k-1}{k}|S|$ of the data

Notes: Every point used as
Test 1 time, Training $k-1$ times

Computational cost for k-forld Cross-validation
linear in k

Can also use "balanced CV"

If class c_i appears in m_i instances,
Force each S_k to include $\frac{k-1}{k}\frac{m_i}{|S|}$ such instances