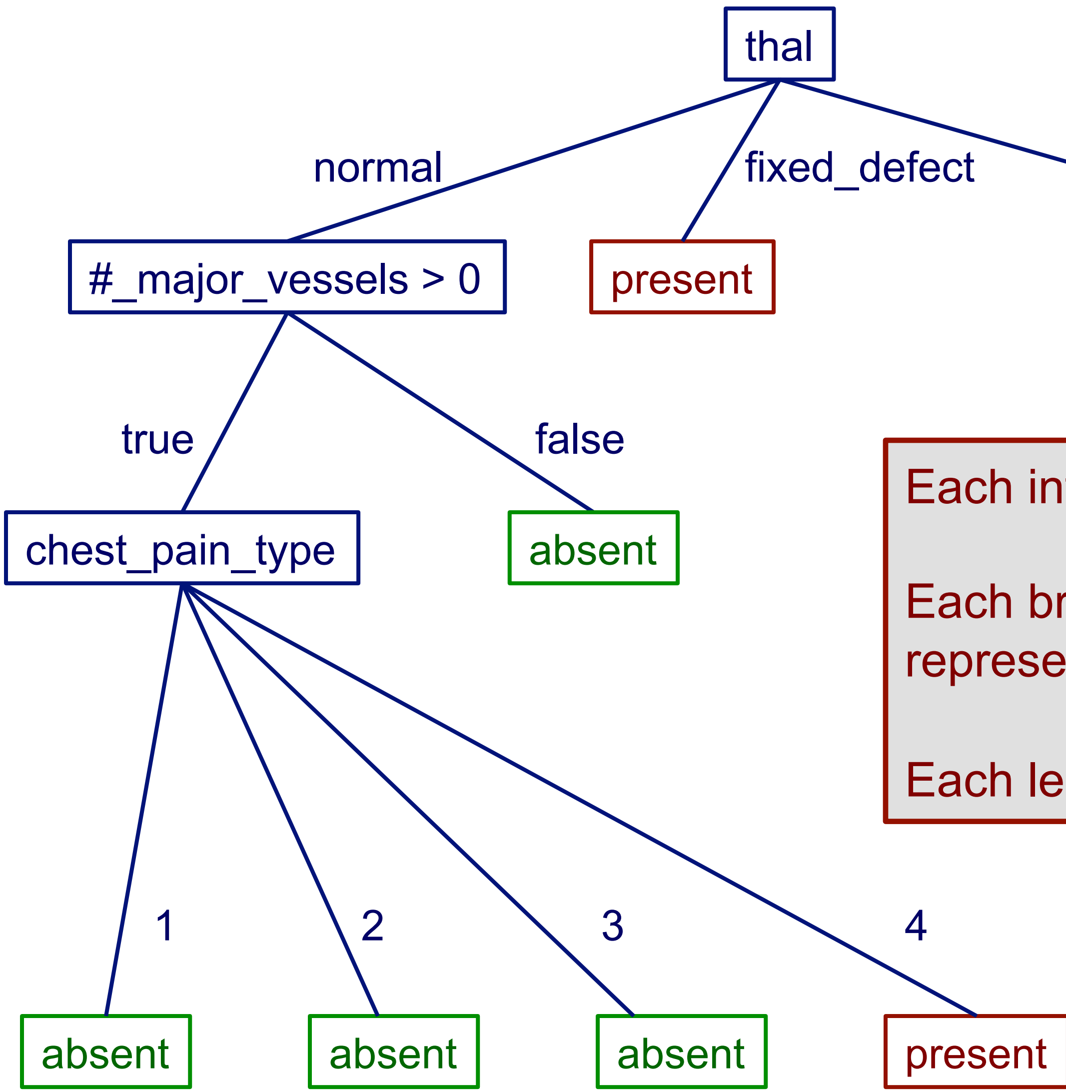
Decision Tree Learning

Goals for the lecture

you should understand the following concepts

- the decision tree representation
- the standard top-down approach to learning a tree
- Occam's razor
- entropy and information gain
- types of decision-tree splits
- test sets and unbiased estimates of accuracy
- overfitting
- early stopping and pruning
- tuning (validation) sets
- regression trees
- m-of-n splits
- using lookahead in decision tree search

A decision tree to predict heart disease



Each internal node tests one feature x_i

Each branch from an internal node represents one outcome of the test

Each leaf predicts y or $P(y \mid \boldsymbol{x})$

Decision tree exercise

Suppose $x_1 \ldots x_5$ are Boolean features, and y is also Boolean

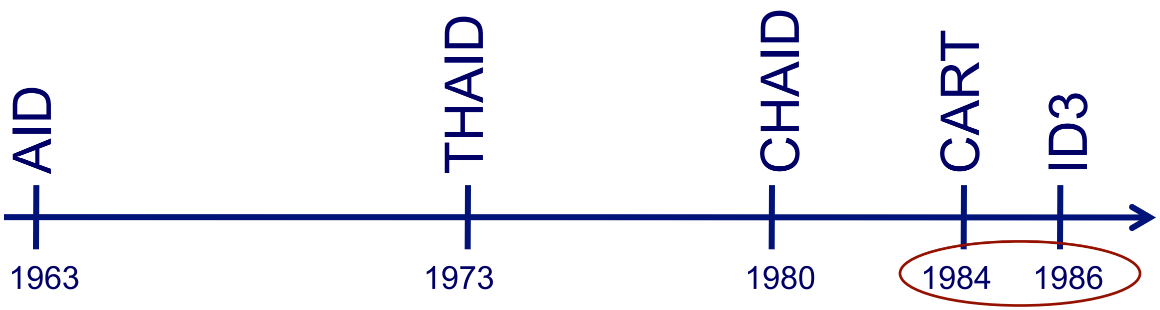
How would you represent the following with decision trees?

$$y = x_2 x_5 \qquad \text{(i.e. } y = x_2 \wedge x_5 \text{)}$$

$$y = x_2 \vee x_5$$

$$y = x_2 x_5 \vee x_3 \neg x_1$$

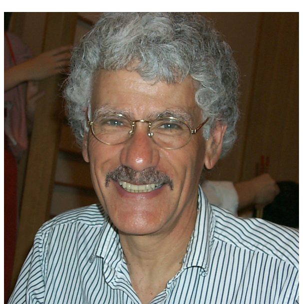
History of decision tree learning

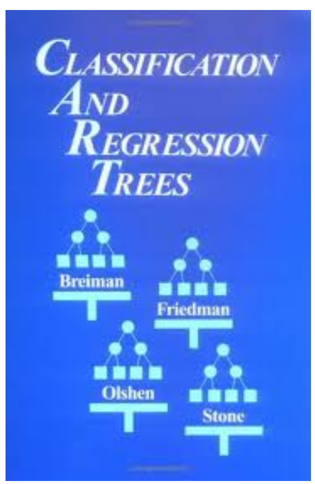


many DT variants have been developed since CART and ID3

dates of seminal publications: work on these 2 was contemporaneous

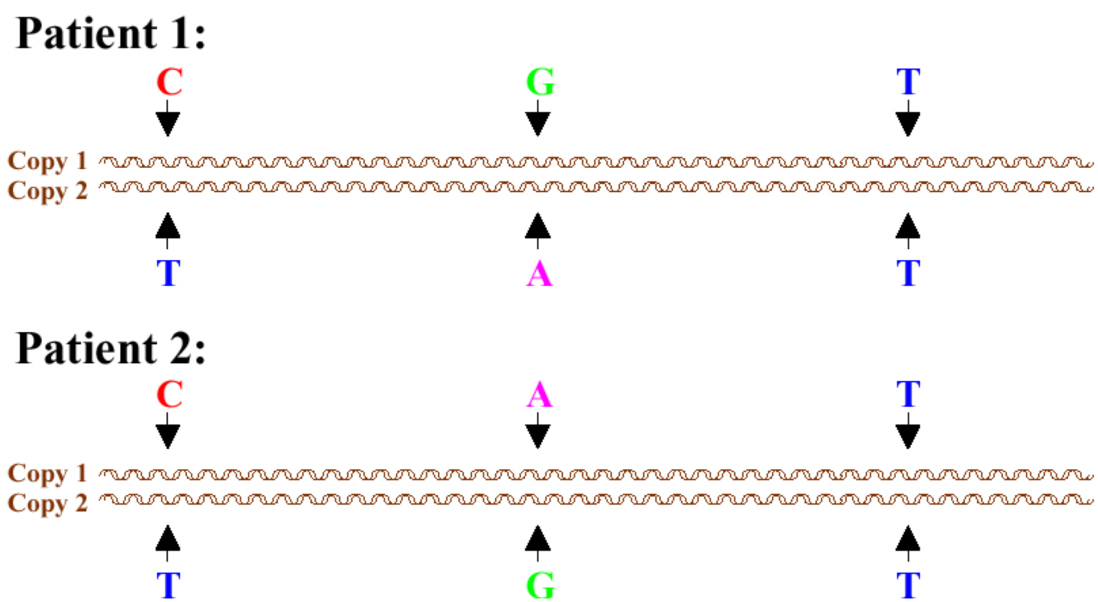
CART developed by Leo Breiman, Jerome Friedman, Charles Olshen, R.A. Stone





ID3, C4.5, C5.0 developed by Ross Quinlan

An Example: Genetic Data



	SNP 1		SNP 2		SNP 3		. . .	class
Patient 1	C	T	A	G	T	T	. . .	Diseased
Patient 2	C	T	A	G	T	T	. . .	Healthy
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .

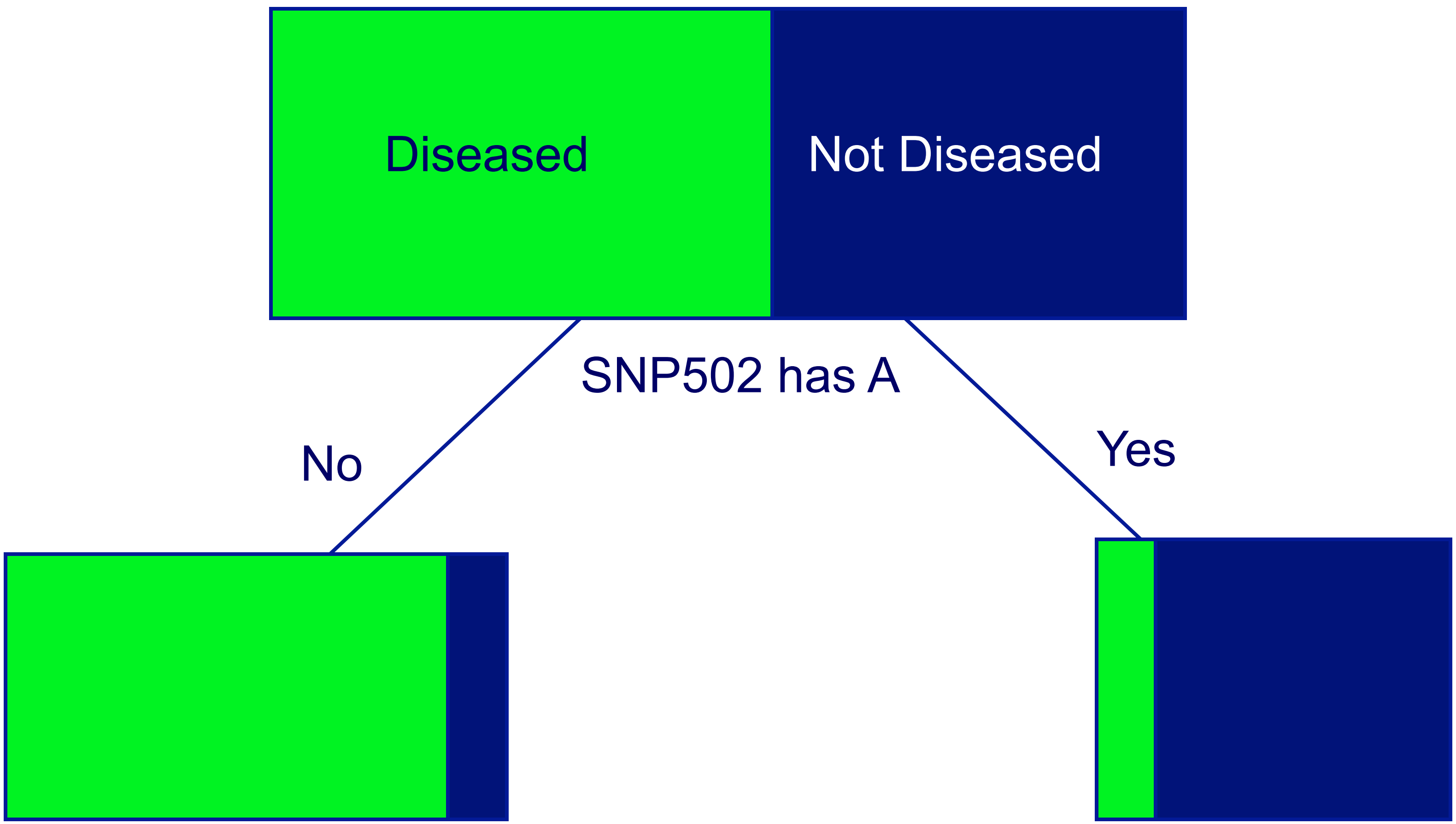
A Supervised Learning Task

- **Given**: a set of SNP profiles, each from a different patient.

  Details: **unordered pair** of DNA bases at each SNP position constitute the features, and patient's disease constitutes the class

- **Do**: Learn a model that accurately predicts class based on features

Decision Trees in One Picture



Top-down decision tree learning

MakeSubtree(set of training instances D)

C = DetermineCandidateSplits(D)

if stopping criteria met

make a leaf node N

determine class label/probabilities for N

else

make an internal node N

S = FindBestSplit(D, C)

for each outcome k of S

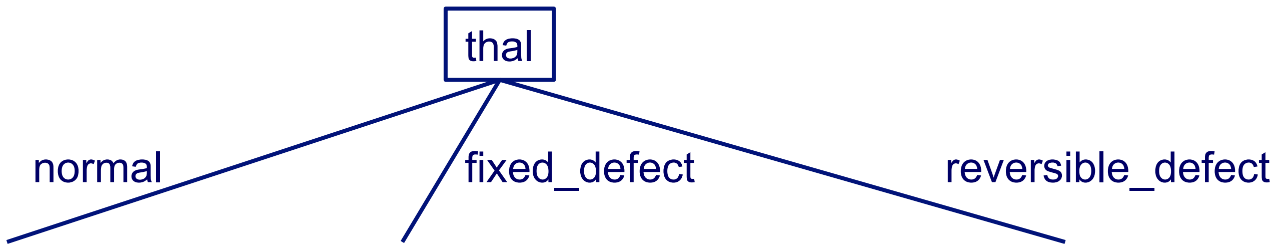
D_k = subset of instances that have outcome k

k^{th} child of N = MakeSubtree(D_k)

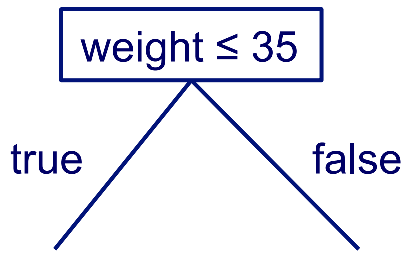
return subtree rooted at N

Candidate splits in ID3, C4.5

- splits on nominal features have one branch per value



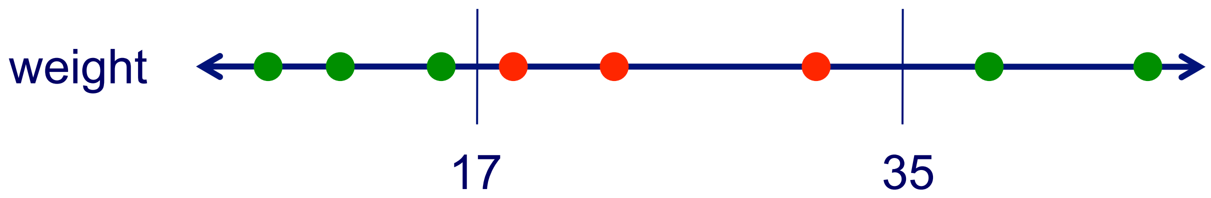
- splits on continuous features use a threshold

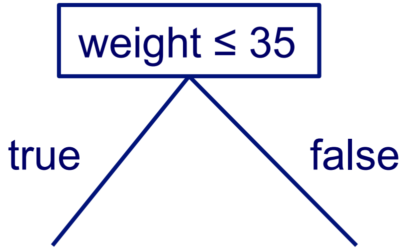


Candidate splits on continuous features

given a set of training instances D and a specific feature F

- sort the values of F in D
- evaluate split thresholds in intervals between instances of different classes

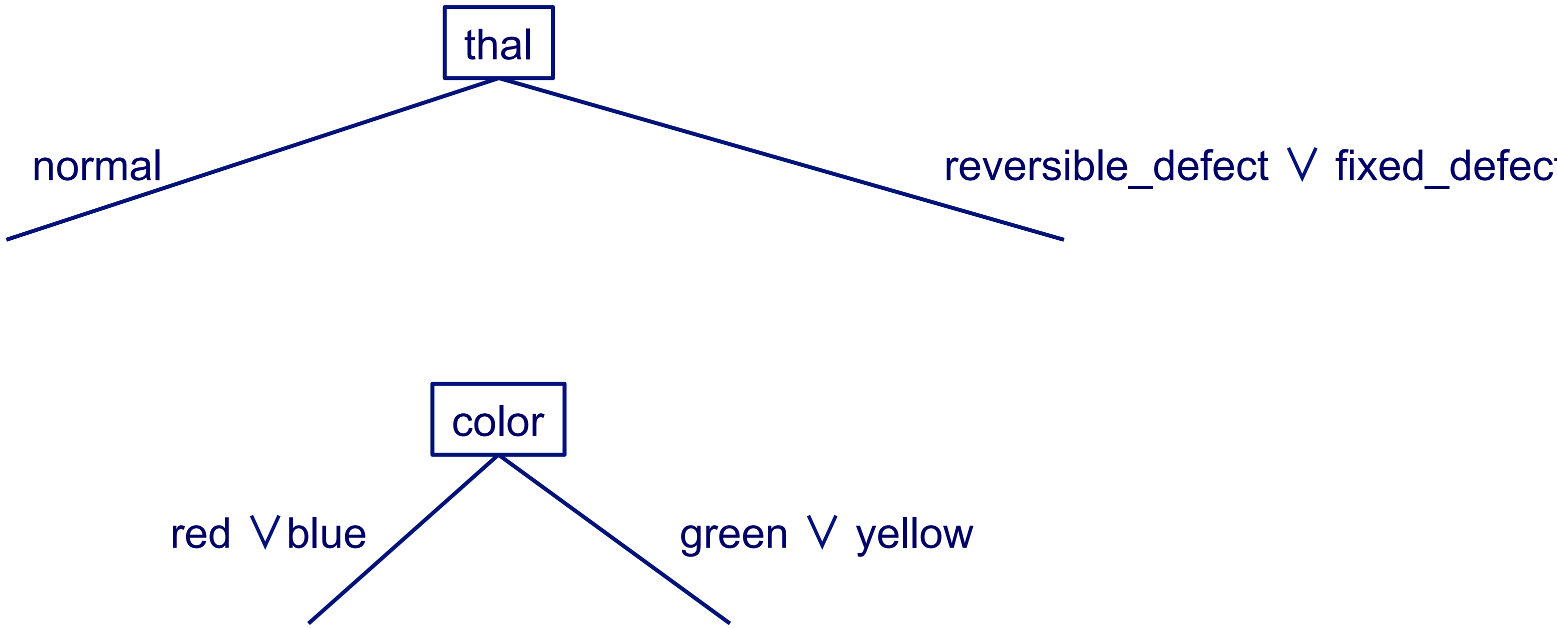




- could use midpoint of each considered interval as the threshold
- C4.5 instead picks the largest value of F in the entire training set that does not exceed the midpoint

Candidate splits

- instead of using k-way splits for k-valued features, could require binary splits on all discrete features (CART does this)



- Breiman et al. proved for the 2-class case, the optimal binary partition can be found considered only $O(k)$ possibilities instead of $O(2^k)$

Finding the best split

- How should we select the best feature to split on at each step?
- Key hypothesis: the simplest tree that classifies the training examples accurately will work well on previously unseen examples

Occam's razor

- attributed to 14th century William of Ockham
- "Nunquam ponenda est pluralitis sin necesitate"
- "Entities should not be multiplied beyond necessity"
- "should proceed to simpler theories until simplicity can be traded for greater explanatory power"
- "when you have two competing theories that make exactly the same predictions, the simpler one is the better"

Ptolemy

But a thousand years earlier, I said, “We consider it a good principle to explain the phenomena by the simplest hypothesis possible.”

Occam's razor and decision trees

Why is Occam's razor a reasonable heuristic for decision tree learning?

- there are fewer short models (i.e. small trees) than long ones
- a short model is unlikely to fit the training data well by chance
- a long model is more likely to fit the training data well coincidentally

Finding the best splits

- Can we return the smallest possible decision tree that accurately classifies the training set?

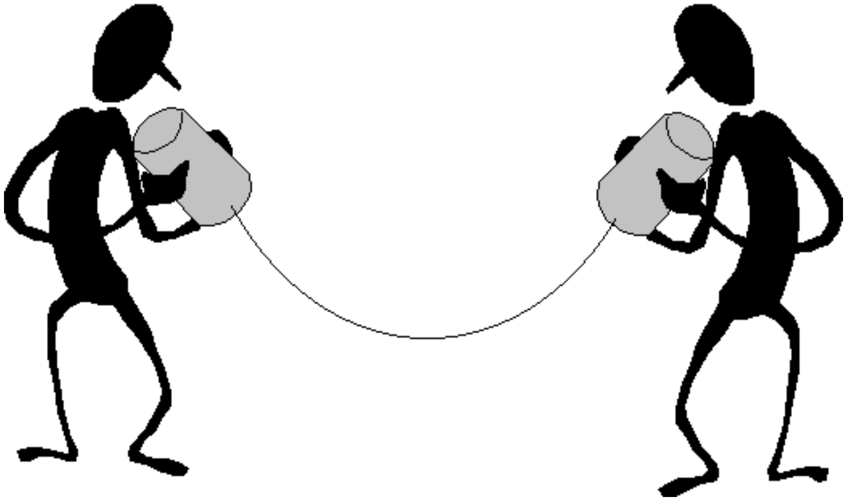
  NO! This is an NP-hard problem
  [Hyafil & Rivest, *Information Processing Letters, 1976]*

- Instead, we'll use an information-theoretic heuristic to greedily choose splits

Information theory background

- consider a problem in which you are using a code to communicate information to a receiver

- example: as bikes go past, you are communicating the manufacturer of each bike



Information theory background

- suppose there are only four types of bikes
- we could use the following code

type	code
Trek	11
Specialized	10
Cervelo	01
Serrota	00

- expected number of bits we have to communicate:
  2 bits/bike

Information theory background

- we can do better if the bike types aren't equiprobable
- optimal code uses $-\log_2 P(y)$ bits for event with probability $P(y)$

Type/probability	# bits	code
$P(\text{Trek}) = 0.5$	1	1
$P(\text{Specialized}) = 0.25$	2	01
$P(\text{Cervelo}) = 0.125$	3	001
$P(\text{Serrota}) = 0.125$	3	000

- expected number of bits we have to communicate: 1.75 bits/bike

$$-\sum_{y \in \text{values}(Y)} P(y) \log_2 P(y)$$

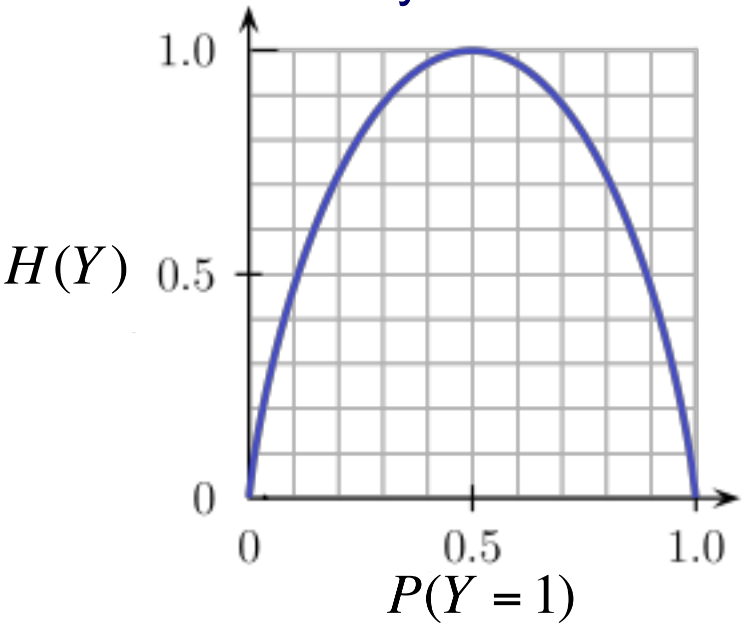
Entropy

- entropy is a measure of uncertainty associated with a random variable

- defined as the expected number of bits required to communicate the value of the variable

$$H(Y) = -\sum_{y \in \text{values}(Y)} P(y) \log_2 P(y)$$

entropy function for binary variable



Conditional entropy

- What's the entropy of Y if we condition on some other variable X?

$$H(Y \mid X) = \sum_{x \in \text{values}(X)} P(X = x)\, H(Y \mid X = x)$$

where

$$H(Y \mid X = x) = -\sum_{y \in \text{values}(Y)} P(Y = y \mid X = x) \log_2 P(Y = y \mid X = x)$$

Information gain (a.k.a. mutual information)

- choosing splits in ID3: select the split S that most reduces the conditional entropy of Y for training set D

$$\text{InfoGain}(D,S) = H_D(Y) - H_D(Y \mid S)$$

D indicates that we're calculating probabilities using the specific sample D

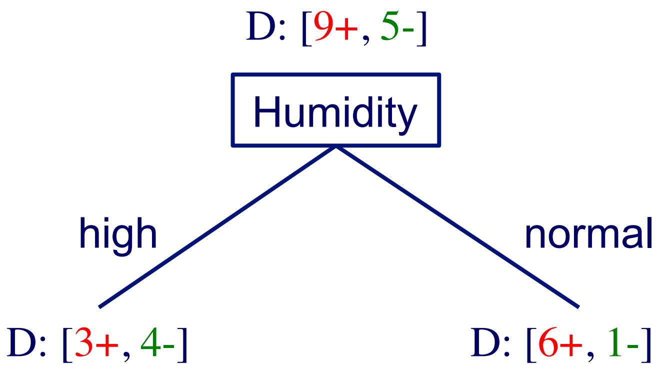
Information gain example

PlayTennis: training examples

Day	Outlook	Temperature	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

Information gain example

- What's the information gain of splitting on Humidity?



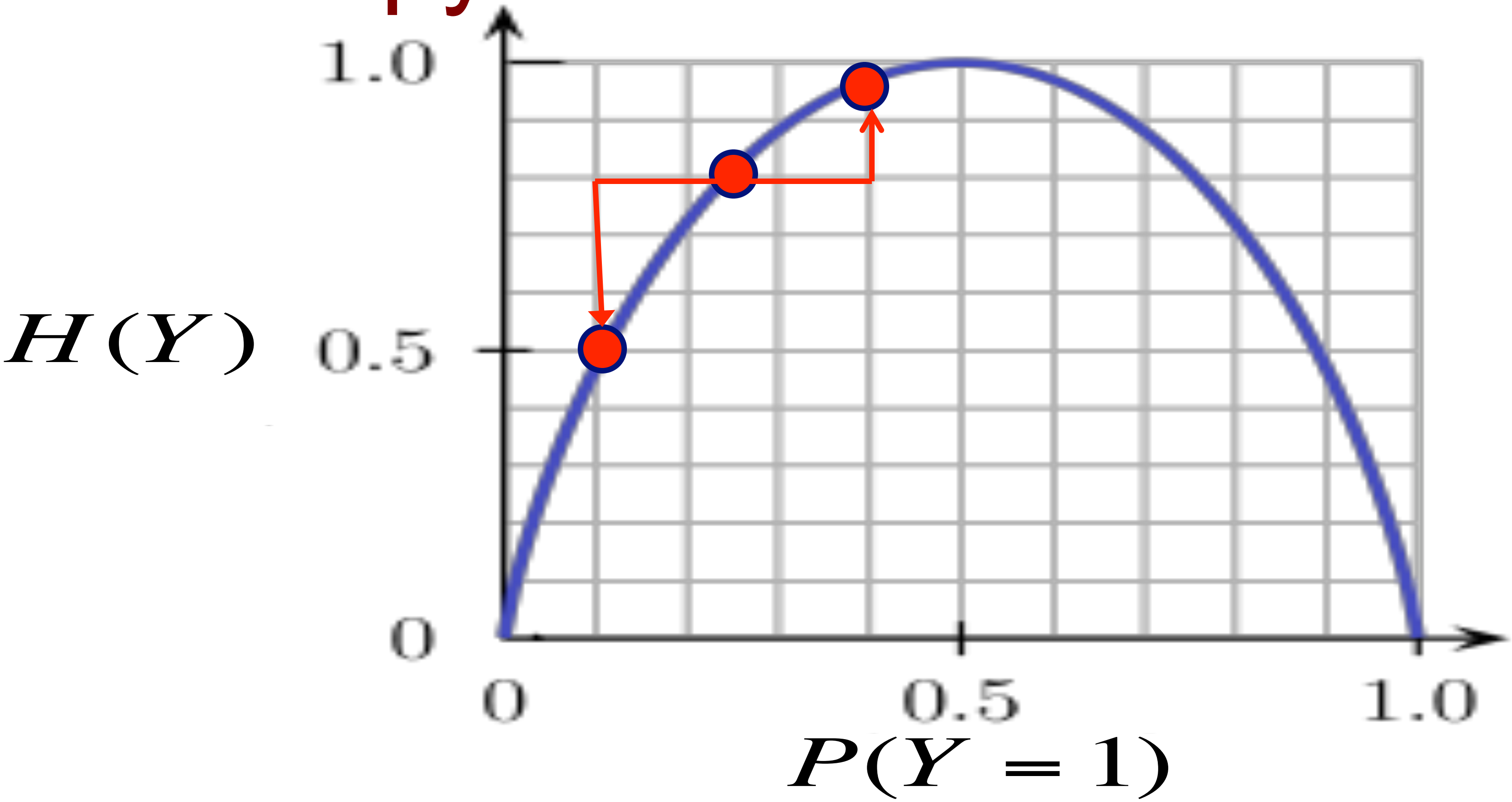
$$H_D(Y) = -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) = 0.940$$

$$H_D(Y \mid \text{high}) = -\frac{3}{7}\log_2\left(\frac{3}{7}\right) - \frac{4}{7}\log_2\left(\frac{4}{7}\right) = 0.985$$

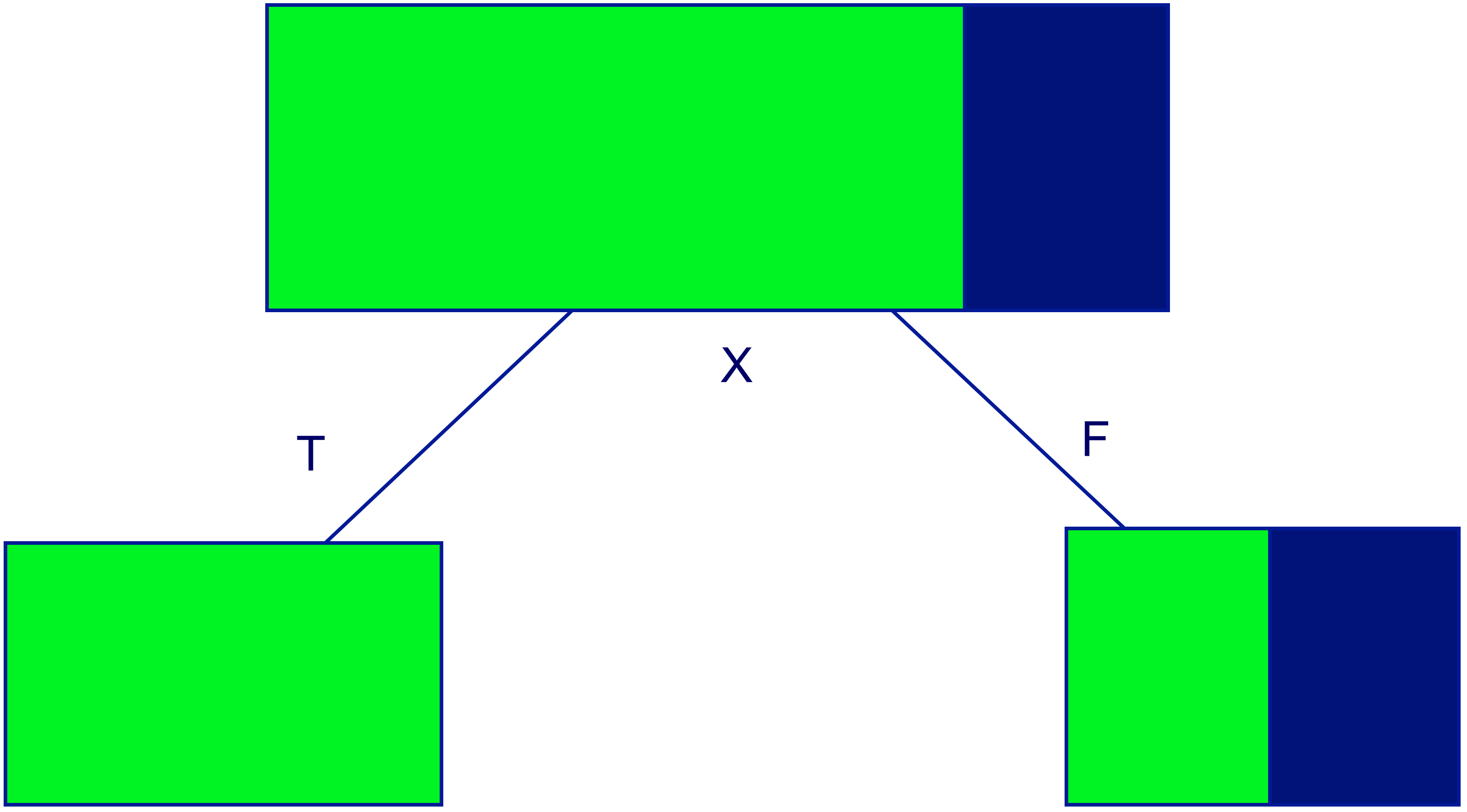
$$H_D(Y \mid \text{normal}) = -\frac{6}{7}\log_2\left(\frac{6}{7}\right) - \frac{1}{7}\log_2\left(\frac{1}{7}\right) = 0.592$$

$$\begin{aligned}\text{InfoGain}(D, \text{Humidity}) &= H_D(Y) - H_D(Y \mid \text{Humidity}) \\ &= 0.940 - \left[\frac{7}{14}(0.985) + \frac{7}{14}(0.592)\right] \\ &= 0.151\end{aligned}$$

Key Property: Equal change in P(Y) yields bigger change in entropy if toward an extreme

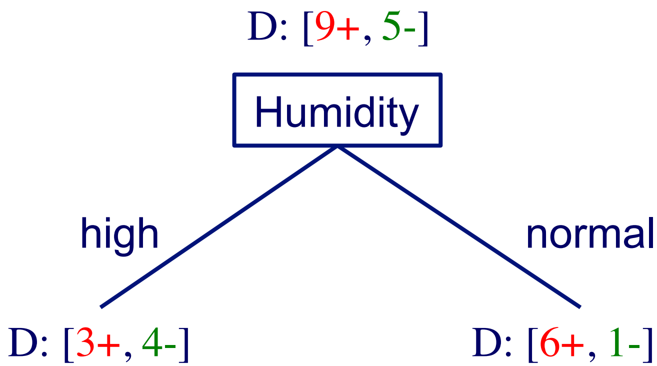


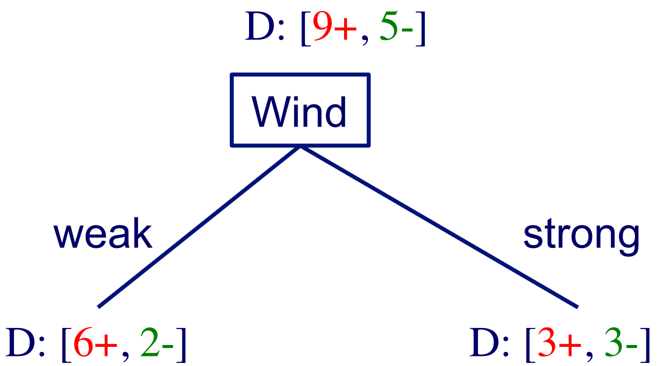
Means there is InfoGain in this split, though no gain in accuracy



Information gain example

- Is it better to split on Humidity or Wind?





$H_D(Y \mid \text{weak}) = 0.811$ $\quad$ $H_D(Y \mid \text{strong}) = 1.0$

✔ $$\text{InfoGain}(D, \text{Humidity}) = 0.940 - \left[\frac{7}{14}(0.985) + \frac{7}{14}(0.592)\right]$$
$$= 0.151$$

$$\text{InfoGain}(D, \text{Wind}) = 0.940 - \left[\frac{8}{14}(0.811) + \frac{6}{14}(1.0)\right]$$
$$= 0.048$$

One limitation of information gain

- information gain is biased towards tests with many outcomes
- e.g. consider a feature that uniquely identifies each training instance
  - splitting on this feature would result in many branches, each of which is “pure” (has instances of only one class)
  - maximal information gain!

Gain ratio

- To address this limitation, C4.5 uses a splitting criterion called *gain ratio*

- consider the potential information generated by splitting on S

$$\text{SplitInfo}(D,S) = -\sum_{k \in \text{outcomes}(S)} \frac{|D_k|}{|D|} \log_2\left(\frac{D_k}{D}\right)$$

use this to normalize information gain

$$\text{GainRatio}(D,S) = \frac{\text{InfoGain}(D,S)}{\text{SplitInfo}(D,S)}$$

Stopping criteria

We should form a leaf when

- all of the given subset of instances are of the same class
- we've exhausted all of the candidate splits

Is there a reason to stop earlier, or to prune back the tree?

How can we assess the accuracy of a tree?

- Can we just calculate the fraction of training examples that are correctly classified?

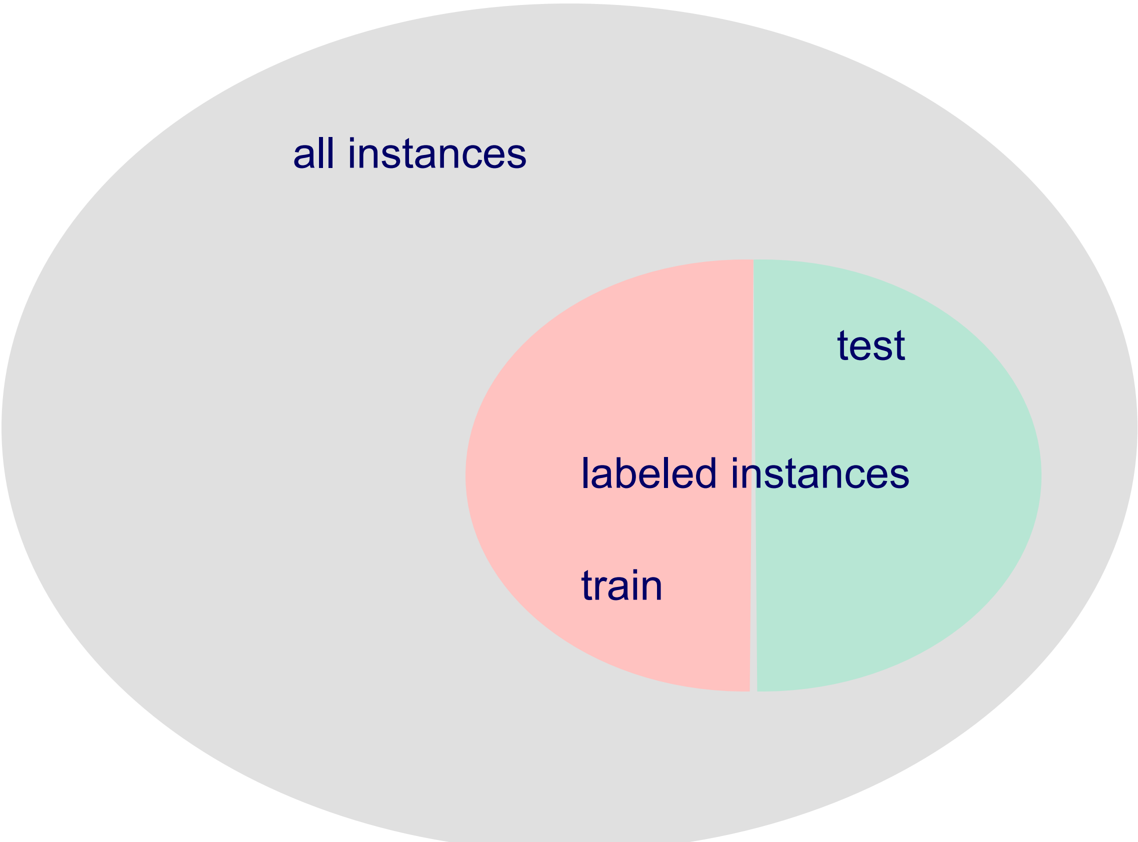
- Consider a problem domain in which instances are assigned labels at random with $P(Y = \text{T}) = 0.5$

  - How accurate would a learned decision tree be on previously unseen instances?

  - How accurate would it be on its training set?

How can we assess the accuracy of a tree?

- to get an unbiased estimate of a learned model's accuracy, we must use a set of instances that are held-aside during learning
- this is called a *test set*



Overfitting

- consider error of model h over
  - training data: $error_D(h)$
  - entire distribution of data: $error(h)$

- model $h \in H$ *overfits* the training data if there is an alternative model $h' \in H$ such that

$$error(h) > error(h')$$

$$error_D(h) < error_D(h')$$

Overfitting with noisy data

suppose

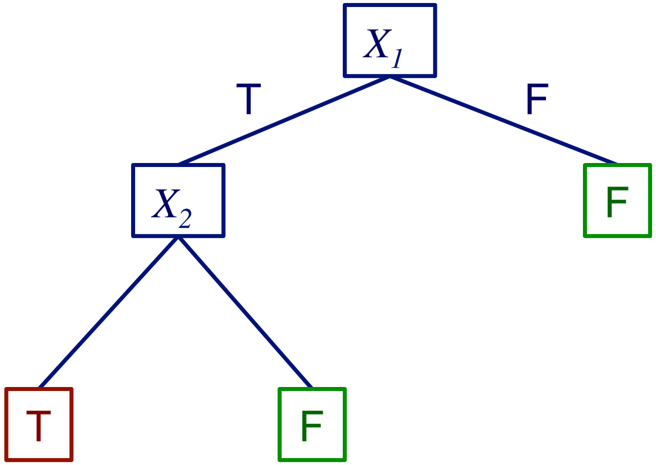
- the target concept is $Y = X_1 \wedge X_2$
- there is noise in some feature values
- we're given the following training set

X_1	X_2	X_3	X_4	X_5	…	Y
T	T	T	T	T	…	T
T	T	F	F	T	…	T
T	F	T	T	F	…	T
T	F	F	T	F	…	F
T	F	T	F	F	…	F
F	T	T	F	T	…	F

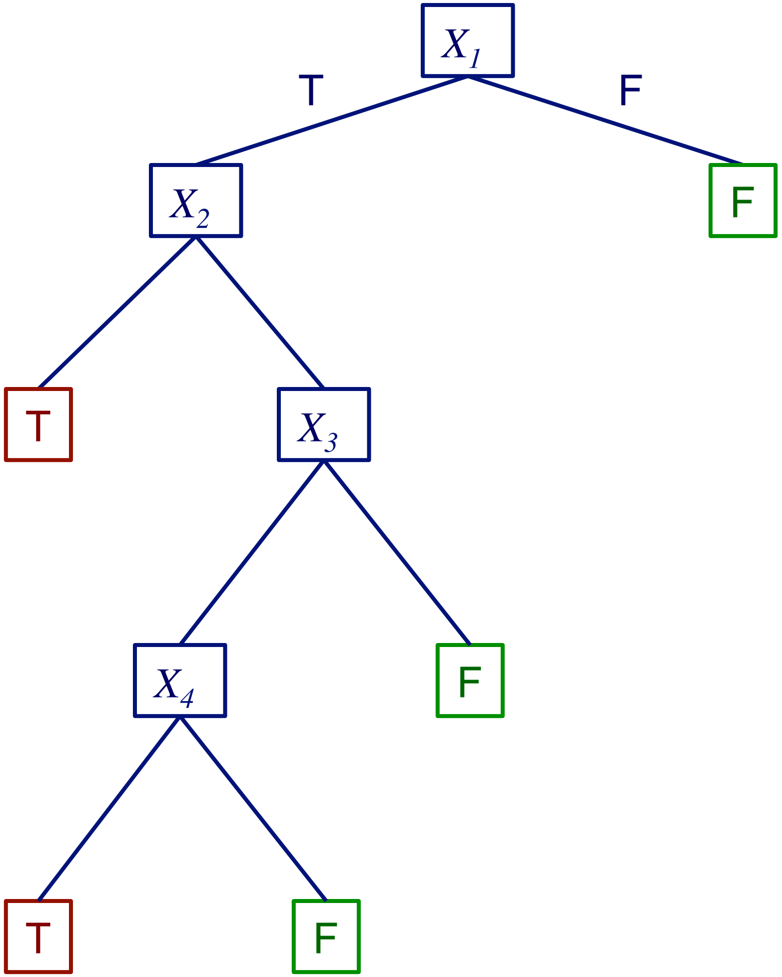
noisy value

Overfitting with noisy data

correct tree



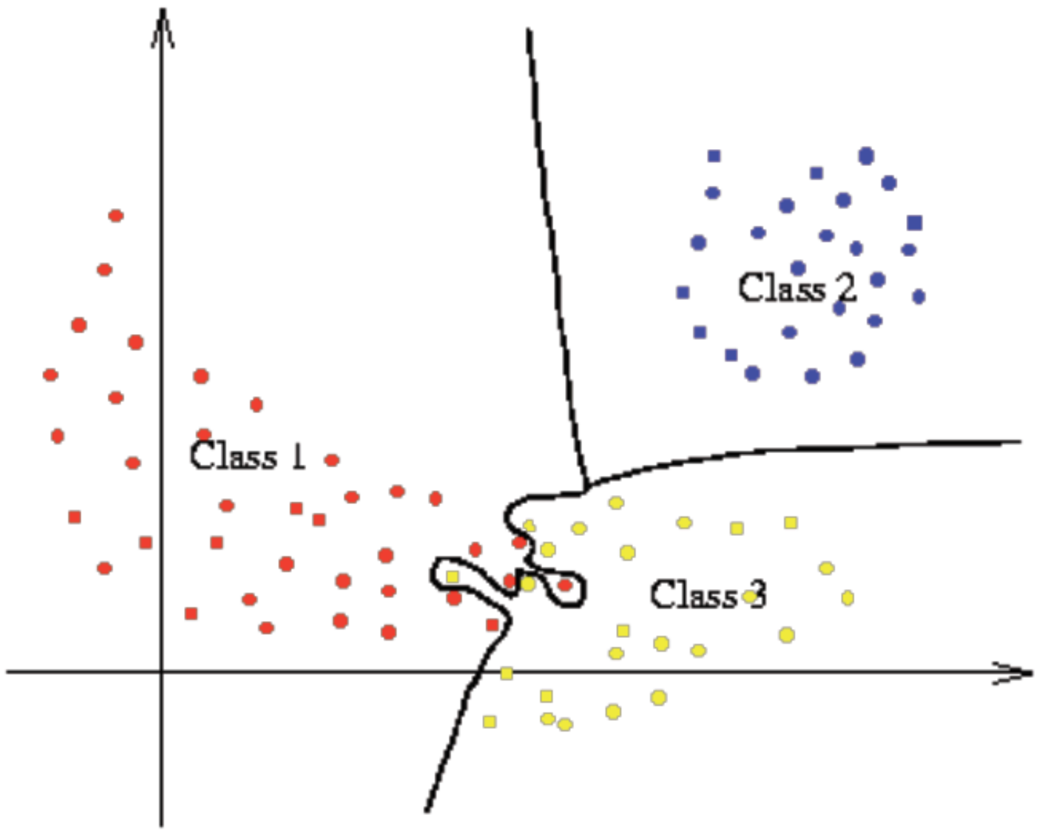
tree that fits noisy training data



Overfitting visualized

consider a problem with

- 2 continuous features
- 3 classes
- some noisy training instances



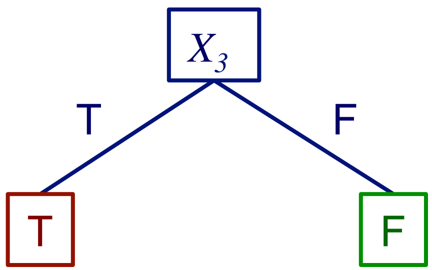
Overfitting with noise-free data

suppose

- the target concept is $Y = X_1 \wedge X_2$
- P(X_3 = T) = 0.5 for both classes
- P(Y = T) = 0.67
- we're given the following training set

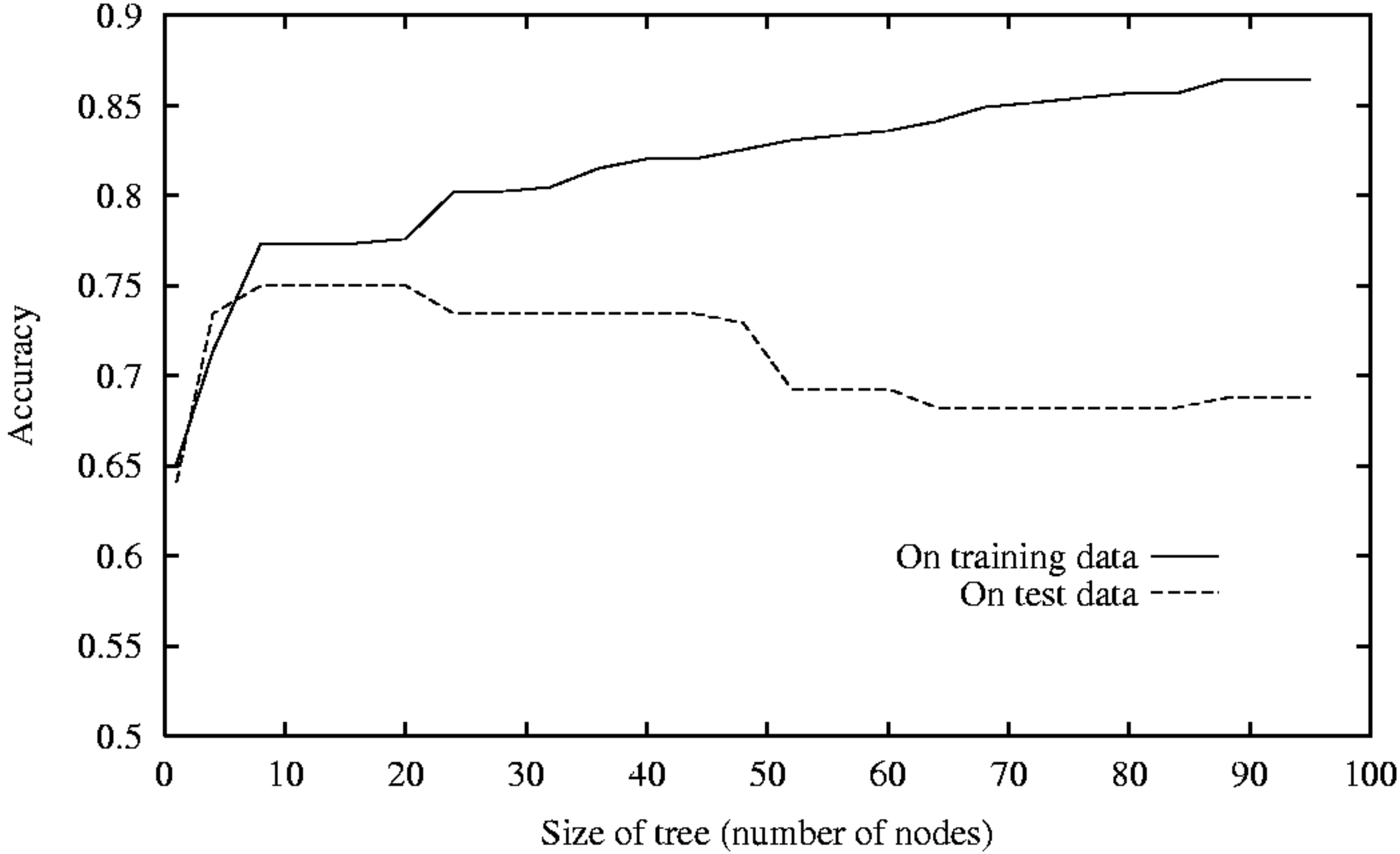
X_1	X_2	X_3	X_4	X_5	…	Y
T	T	T	T	T	…	T
T	T	T	F	T	…	T
T	T	T	T	F	…	T
T	F	F	T	F	…	F
F	T	F	F	T	…	F

Overfitting with noise-free data

	training set accuracy	test set accuracy
	100%	50%
	66%	66%

- because the training set is a limited sample, there might be (combinations of) features that are correlated with the target concept by chance

Overfitting in decision trees



Avoiding overfitting in DT learning

two general strategies to avoid overfitting

1. *early stopping*: stop if further splitting not justified by a statistical test
   - Quinlan's original approach in ID3

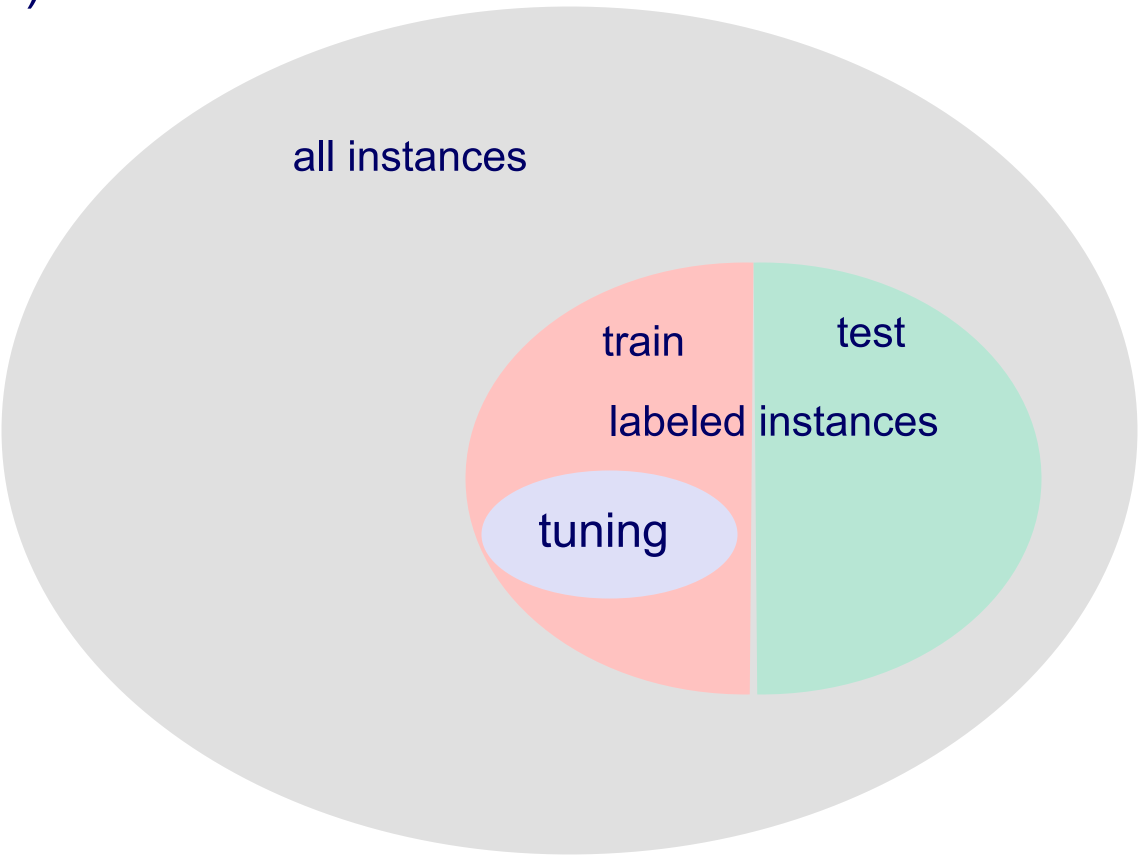
2. *post-pruning*: grow a large tree, then prune back some nodes
   - more robust to myopia of greedy tree learning

Pruning in ID3, C4.5

1. split given data into training and *tuning* (*validation*) sets
2. grow a complete tree
3. do until further pruning is harmful
   - evaluate impact on tuning-set accuracy of pruning each node
   - greedily remove the one that most improves tuning-set accuracy

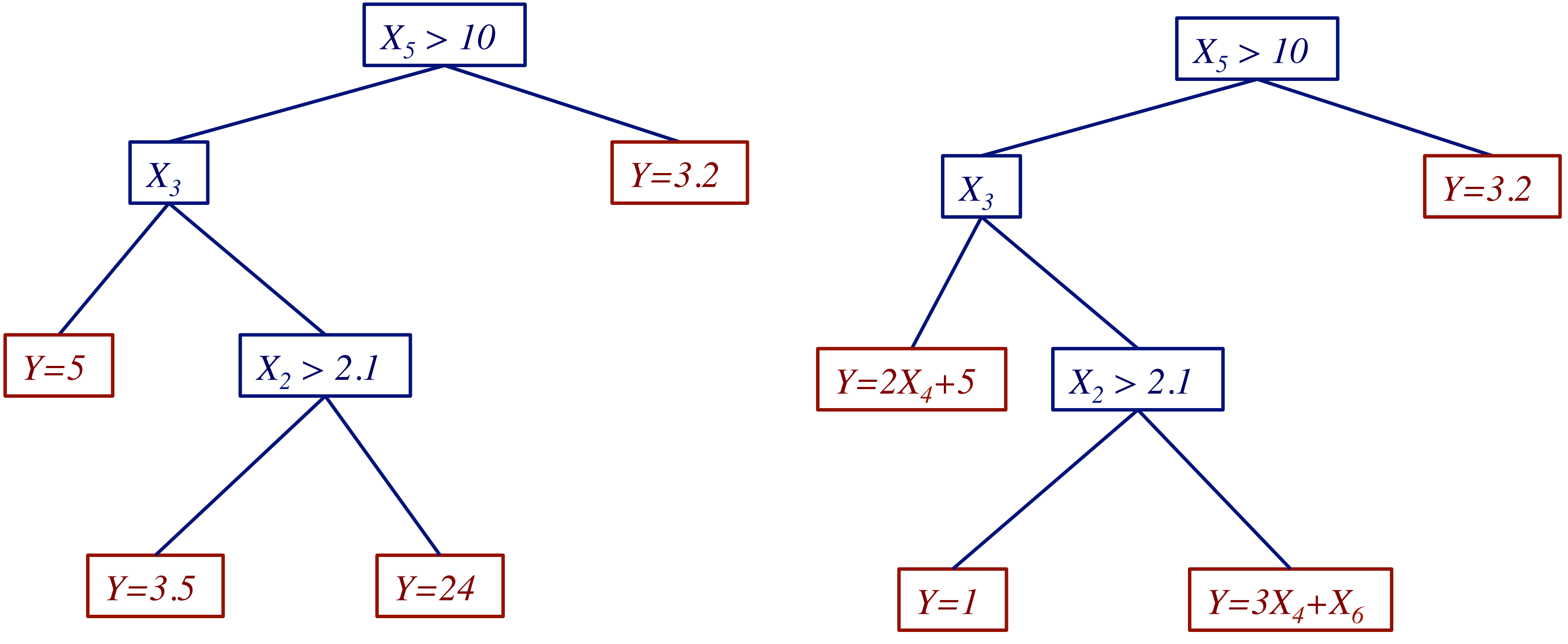
Tuning sets

- a *tuning set* (a.k.a. *validation set*) is a subset of the training set that is held aside
  - not used for primary training process (e.g. tree growing)
  - but used to select among models (e.g. trees pruned to varying degrees)



Regression trees

- in a regression tree, leaves have functions that predict numeric values instead of class labels
- the form of these functions depends on the method
  - CART uses constants: regression trees
  - some methods use linear functions: model trees



Regression trees in CART

- CART does *least squares regression* which tries to minimize

$$\sum_{i=1}^{|D|} \left(y_i - \hat{y}_i\right)^2$$

target value for i^{th} training instance

value predicted by tree for i^{th} training instance (average valueof y for training instances reaching the leaf)

$$= \sum_{L \in \text{leaves}} \sum_{i \in L} \left(y_i - \hat{y}_i\right)^2$$

- at each internal node, CART chooses the split that most reduces this quantity
- if *D* is data at node, minimize *variance* $1/|D| \sum_{i=1}^{|D|} \left(y_i - \hat{y}_i\right)^2$

Lookahead

- most DT learning methods use a hill-climbing search
- a limitation of this approach is myopia: an important feature may not appear to be informative until used in conjunction with other features
- can potentially alleviate this limitation by using a *lookahead* search [Norton ‘89; Murphy & Salzberg ‘95]
- empirically, often doesn’t improve accuracy or tree size

Choosing best split in ordinary DT learning

OrdinaryFindBestSplit(set of training instances D, set of candidate splits C)

$maxgain = -\infty$

for each split S in C

$gain$ = InfoGain(D, S)

if $gain > maxgain$

$maxgain = gain$

$S_{best} = S$

return S_{best}

Choosing best split with lookahead (part 1)

LookaheadFindBestSplit(set of training instances D, set of candidate splits C)

$maxgain$ = -∞

for each split S in C

$gain$ = EvaluateSplit(D, C, S)

if $gain > maxgain$

$maxgain = gain$

$S_{best} = S$

return S_{best}

Choosing best split with lookahead (part 2)

EvaluateSplit(D, C, S)

if a split on S separates instances by class (i.e. $H_D(Y \mid S) = 0$)

// no need to split further

return $H_D(Y) - H_D(Y \mid S)$

else

for outcomes $k \in \{1, 2\}$ of S // let's assume binary splits

// see what the splits at the next level would be

D_k = subset of instances that have outcome k

S_k = OrdinaryFindBestSplit($D_k, C - S$)

// return information gain that would result from this 2-level subtree

return $H_D(Y) - H_D(Y \mid S, S_1, S_2)$

Correlation Immune (CI) Function

Female	*Sxl* gene active	Survival
0	0	0
0	1	1
1	0	1
1	1	0

Drosophila survival based on gender and *Sxl* gene activity

Learning CI Functions

- Standard method of learning hard functions with TDIDT: depth-*k* lookahead
  - $O(mn^{2^k-1})$ for *m* examples in *n* variables

- Can we devise a technique that allows TDIDT algorithms to *efficiently* learn hard functions?

Key Idea

Hard functions aren't – if the data distribution is significantly different from uniform

Example

- Uniform distribution can be sampled by setting each variable (feature) independently of all others, with probability 0.5 of being set to 1.

- Consider same distribution, but with each variable having probability 0.75 of being set to 1.

Example

x_1	x_2	$x_3 \dots x_{100}$	f
0	0	0…0000000 0…0000001 0…0000010 … 1…1111111	0
0	1	0…0000000 0…0000001 0…0000010 … 1…1111111	1
1	0	0…0000000 0…0000001 0…0000010 … 1…1111111	1
1	1	0…0000000 0…0000001 0…0000010 … 1…1111111	0

$$GINI(f) = 0.25$$

$$GINI(f; x_i = 0) = 0.25$$

$$GINI(f; x_i = 1) = 0.25$$

$$\Downarrow$$

$$GAIN(x_i) = 0$$

Example

x_1	x_2	$x_3 \ldots x_{100}$	f	*Weight*
0	0	0…0000000 0…0000001 0…0000010 … 1…1111111	0	$\frac{1}{16}$
0	1	0…0000000 0…0000001 0…0000010 … 1…1111111	1	$\frac{3}{16}$
1	0	0…0000000 0…0000001 0…0000010 … 1…1111111	1	$\frac{3}{16}$
1	1	0…0000000 0…0000001 0…0000010 … 1…1111111	0	$\frac{9}{16}$

$$GINI(f) = \frac{60}{256}$$

$$GINI(f; x_1 = 0) = \frac{48}{256}$$

$$GINI(f; x_1 = 1) = \frac{48}{256}$$

$$\Downarrow$$

$$GAIN(x_1) = \frac{(60-48)}{256} = \frac{12}{256}$$

$$GINI(f; x_4 = 0) = \frac{60}{256}$$

$$GINI(f; x_4 = 1) = \frac{60}{256}$$

$$\Downarrow$$

$$GAIN(x_4) = 0$$

Example

x_1	x_2	$x_3 \ldots x_{100}$	f	*Weight*
0	0	0…0000000 0…0000001 0…0000010 … 1…1111111	0	$\frac{1}{16}$
0	1	0…0000000 0…0000001 0…0000010 … 1…1111111	1	$\frac{3}{16}$
1	0	0…0000000 0…0000001 0…0000010 … 1…1111111	1	$\frac{3}{16}$
1	1	0…0000000 0…0000001 0…0000010 … 1…1111111	0	$\frac{9}{16}$

$$GINI(f) = \frac{6}{16}\frac{10}{16} = \frac{60}{256}$$

Example

x_1	x_2	$x_3 \ldots x_{100}$	f	*Weight*
0	0	0…0000000 0…0000001 0…0000010 … 1…1111111	0	$\frac{1}{16}$
0	1	0…0000000 0…0000001 0…0000010 … 1…1111111	1	$\frac{3}{16}$
1	0	0…0000000 0…0000001 0…0000010 … 1…1111111	1	$\frac{3}{16}$
1	1	0…0000000 0…0000001 0…0000010 … 1…1111111	0	$\frac{9}{16}$

$$GINI(f; x_1 = 0) = \frac{1}{4}\frac{3}{4} = \frac{48}{256}$$

$$GINI(f; x_1 = 1) = \frac{1}{4}\frac{3}{4} = \frac{48}{256}$$

Example

x_4	x_1	$x_2 x_3 x_5 \ldots x_{100}$	f	*Weight*
0	0	0…0000000 0…0000001 0…0000010 … 1…1111111	.25:0 .75:1	$\frac{1}{16}$
0	1	0…0000000 0…0000001 0…0000010 … 1…1111111	.75:0 .25:1	$\frac{3}{16}$
1	0	0…0000000 0…0000001 0…0000010 … 1…1111111	.25:0 .75:1	$\frac{3}{16}$
1	1	0…0000000 0…0000001 0…0000010 … 1…1111111	.75:0 .25:1	$\frac{9}{16}$

$$GINI(f; x_4 = 0) = \left[\frac{1}{4}\frac{1}{4} + \frac{3}{4}\frac{3}{4}\right]\left[\frac{1}{4}\frac{3}{4} + \frac{3}{4}\frac{1}{4}\right] = \frac{10}{16}\frac{6}{16} = \frac{60}{256}$$

$$GINI(f; x_4 = 1) = \frac{60}{256}$$

Can Show

- Given
  - a *large enough sample* and
  - a second distribution *sufficiently different from the first*,

  we can learn functions that are hard for TDIDT algorithms under the original distribution.

Issues to Address

- How can we get a “sufficiently different” distribution?
  - Our approach: “skew” the given sample by choosing “favored settings” for the variables

- Not-large-enough sample effects?
  - Our approach: Average “goodness” of any variable over multiple skews

Skewing Algorithm

- For *T* trials do
  - Choose a favored setting for each variable
  - Reweight the sample
  - Calculate entropy of each variable split under this weighting
  - For each variable that has sufficient gain, increment a counter

- Split on the variable with the highest count

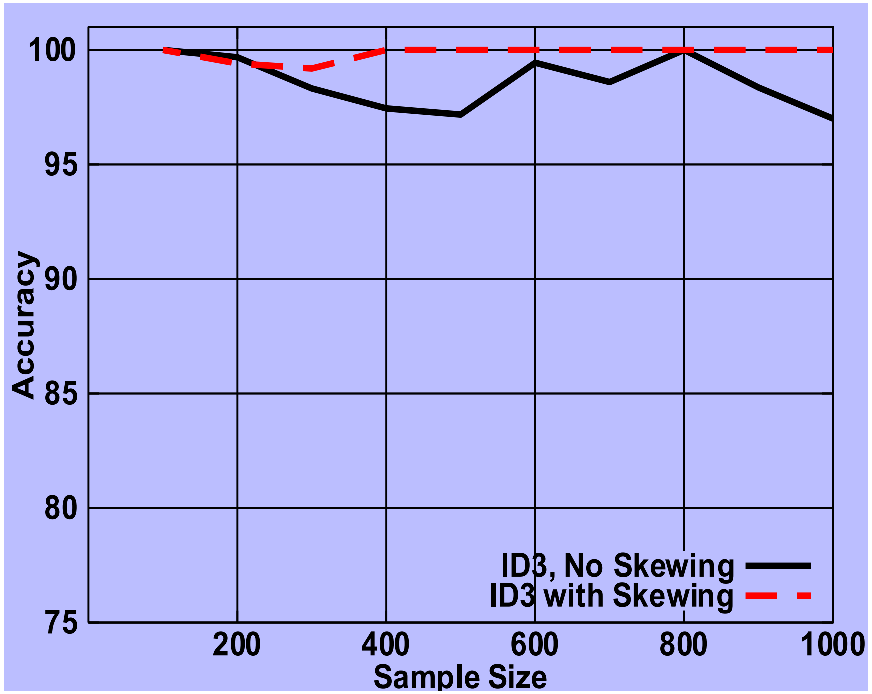
Experiments

- ID3 vs. ID3 with Skewing (ID3 to avoid issues to do with parameters, pruning, etc.)

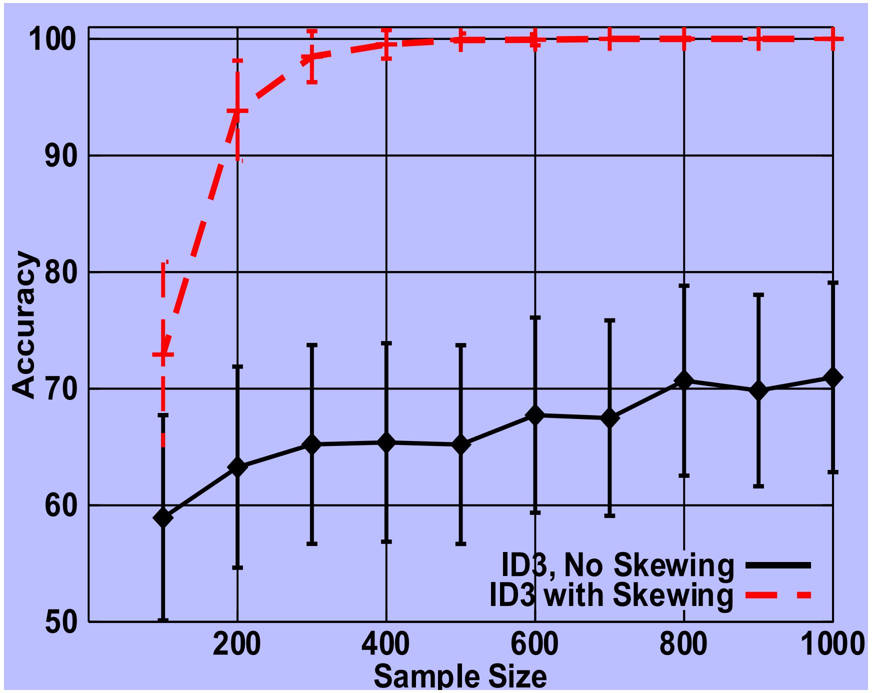
- Several UCI Datasets

- Synthetic Propositional Data
  - Examples of 30 Boolean variables.
  - Target Boolean functions of 2-6 of these variables.
  - Randomly chosen targets and randomly chosen hard targets.

Results (3-variable Boolean functions)

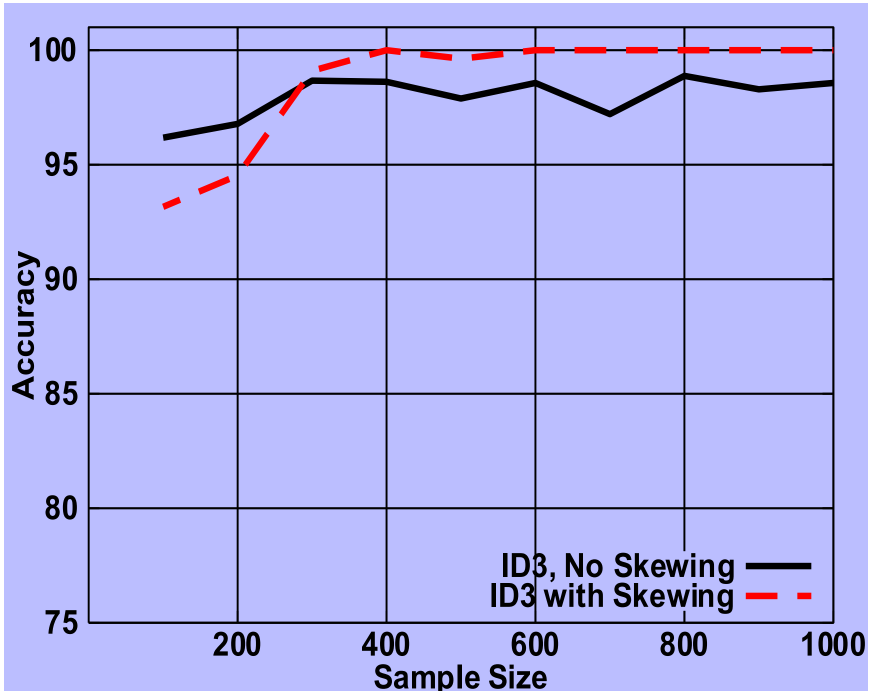


Random functions

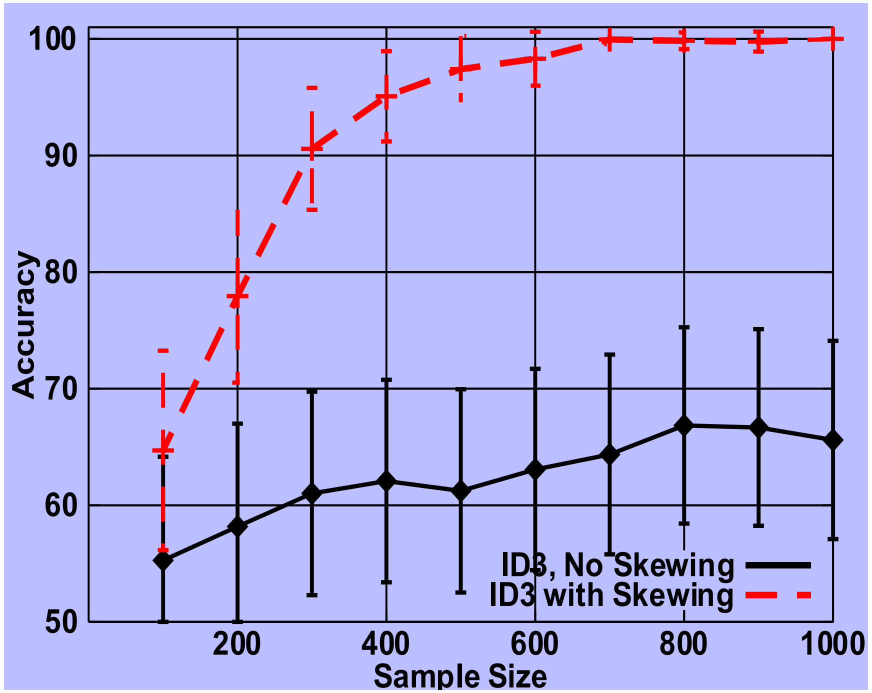


CI functions

Results (4-variable Boolean functions)

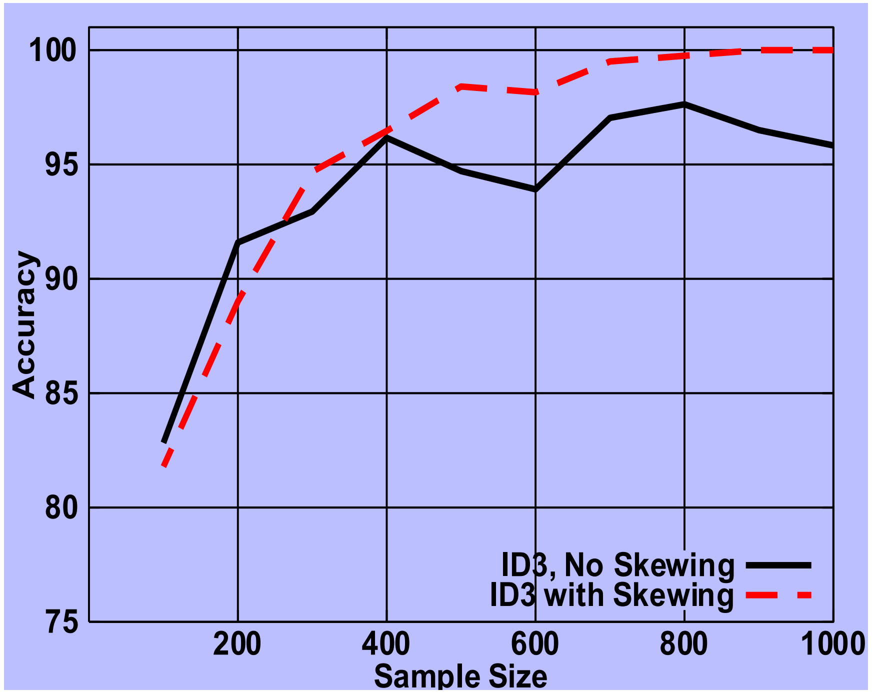


Random functions

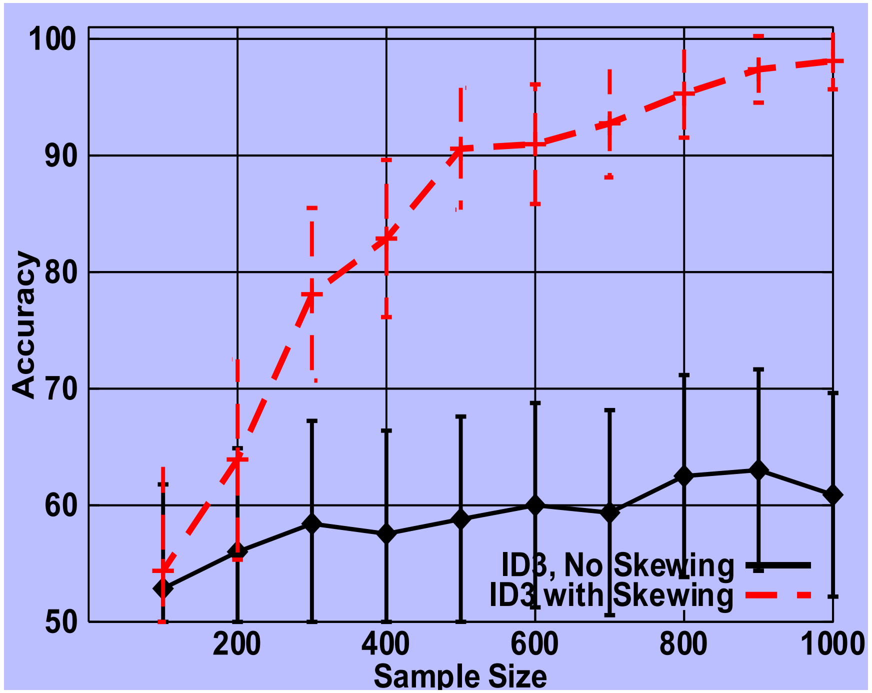


CI functions

Results (5-variable Boolean functions)

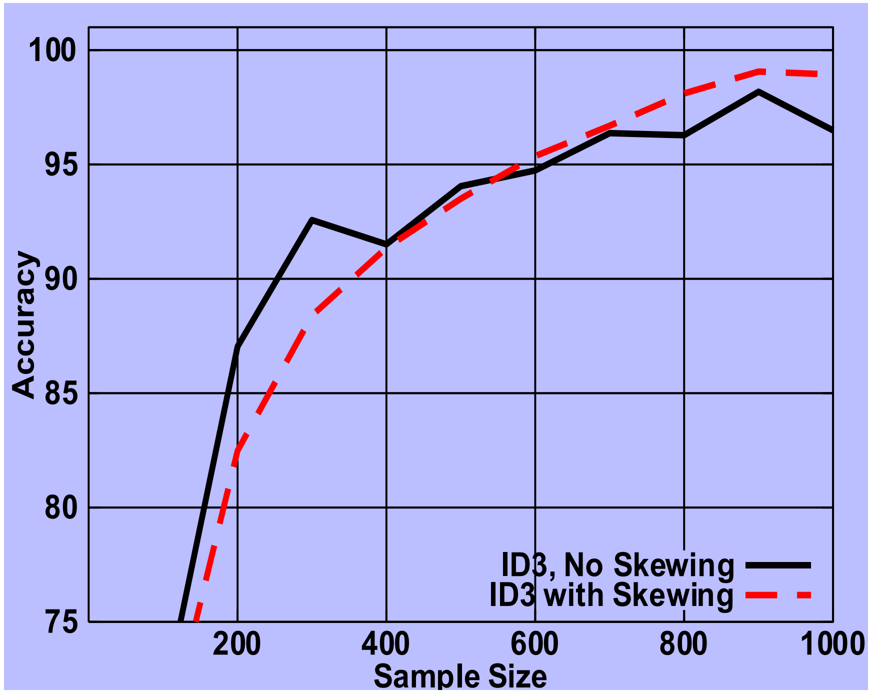


Random functions

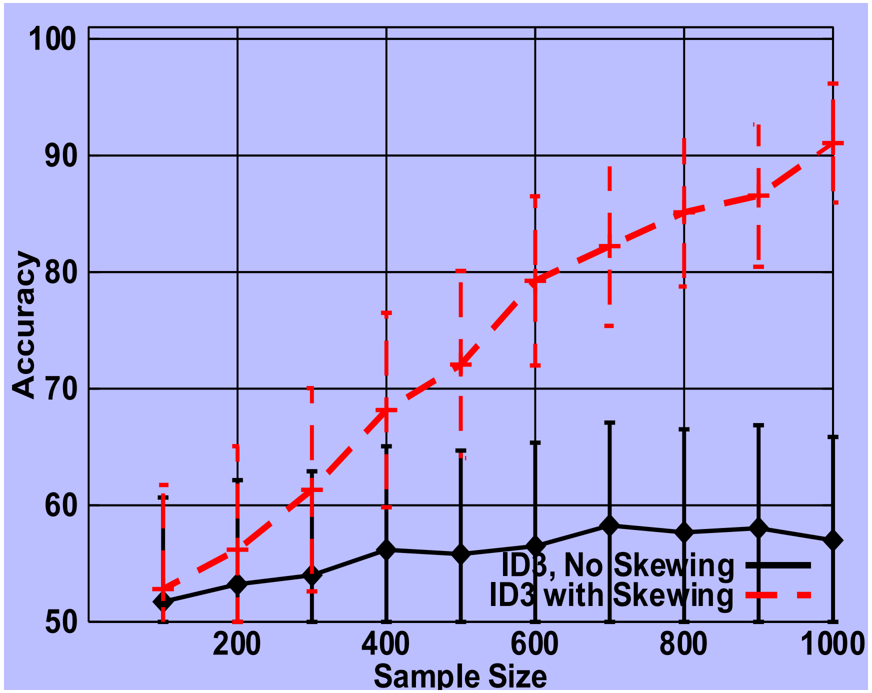


CI functions

Results (6-variable Boolean functions)



Random functions



CI functions

Accuracy Results (UCI datasets)

Data Set	ID3	ID3 with Skewing
Heart	71.9	74.5
Voting	94.0	94.2
Contra	60.4	61.5
Monks-1	92.6	100.0
Monks-2	86.5	89.3
Monks-3	89.8	91.7

Empirical Conclusions

- Skewing rarely hurts (hurts only at very small sample sizes for the tasks we investigated).
- Skewing helps dramatically when the target is hard.
- Hard functions appear to be relatively uncommon in UCI database.

Comments on decision tree learning

- widely used approach
- many variations
- fast in practice
- provides humanly comprehensible models when trees not too big
- insensitive to monotone transformations of numeric features
- standard methods learn axis-parallel hypotheses*
- standard methods not suited to on-line setting*
- usually not among most accurate learning methods

* although variants exist that are exceptions to this