DECISION TREE LEARNING

[read Chapter 3]

[recommended exercises 3.1, 3.4]

- Decision tree representation
- ID3 learning algorithm
- Entropy, Information gain
- Overfitting

Decision Tree

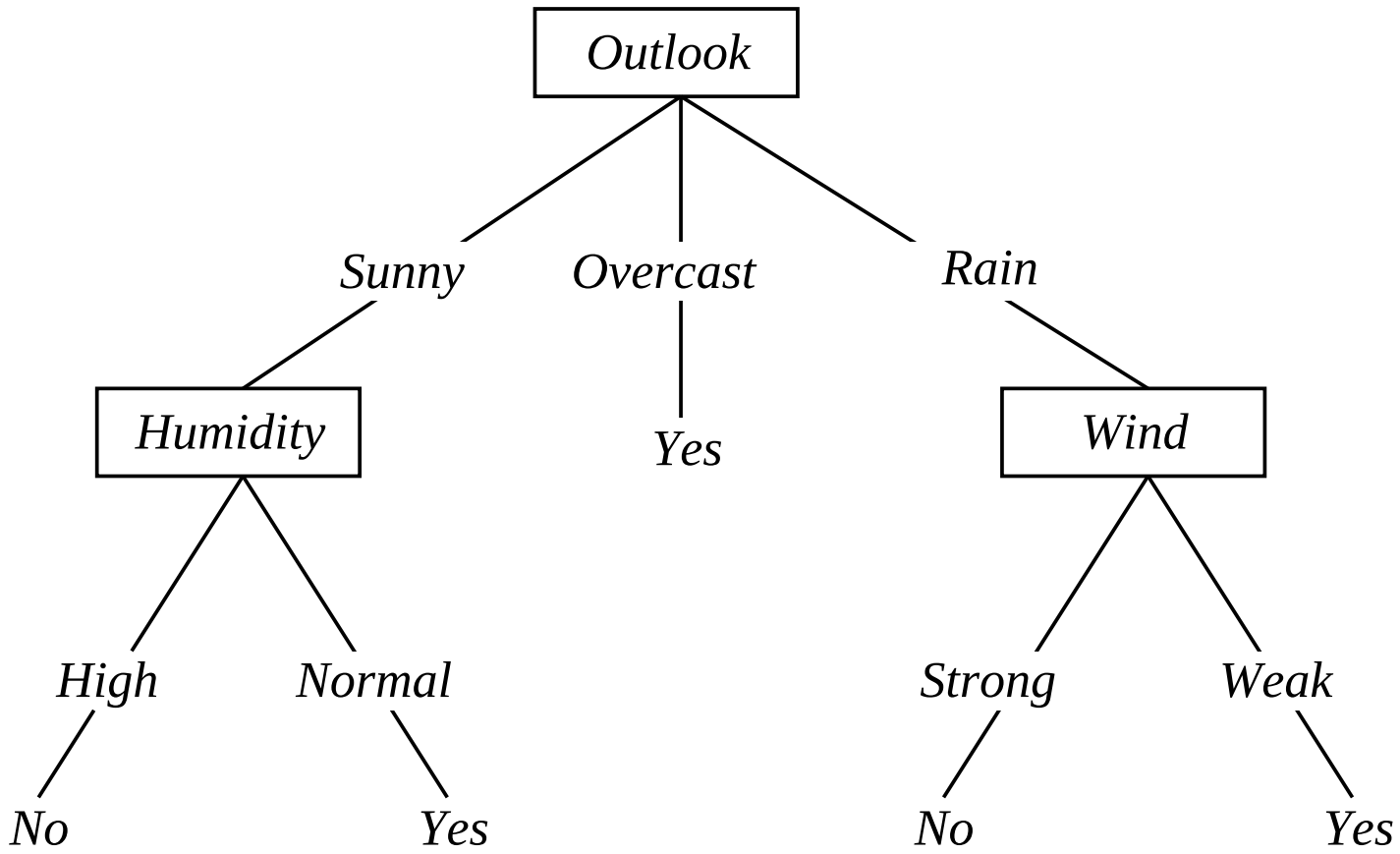
Representation: Tree-structured plan of a set of attributes to test in order to predict the output

1. In the simplest case:
   - Each internal node tests on a attribute
   - Each branch corresponds to an attribute
   - Each leaf corresponds to a class label
2. In general:
   - Each internal node corresponds to a test (on input instances) with mutually exclusive and exhaustive outcomes —test may be univariate or multivariate
   - Each branch corresponds to an outcome of a test
   - Each leaf corresponds to a class label

Learning: Build a DT consistent with a set of training examples T for a concept C with classes $C_1, \ldots, C_k$

Example: Decision Tree for $PlayTennis$

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



There are far too many DT's consistent with a training set

How many Decision Trees?

- Many DT's are consistent with the same training set

- Simple DT's are preferred over more complex DT's

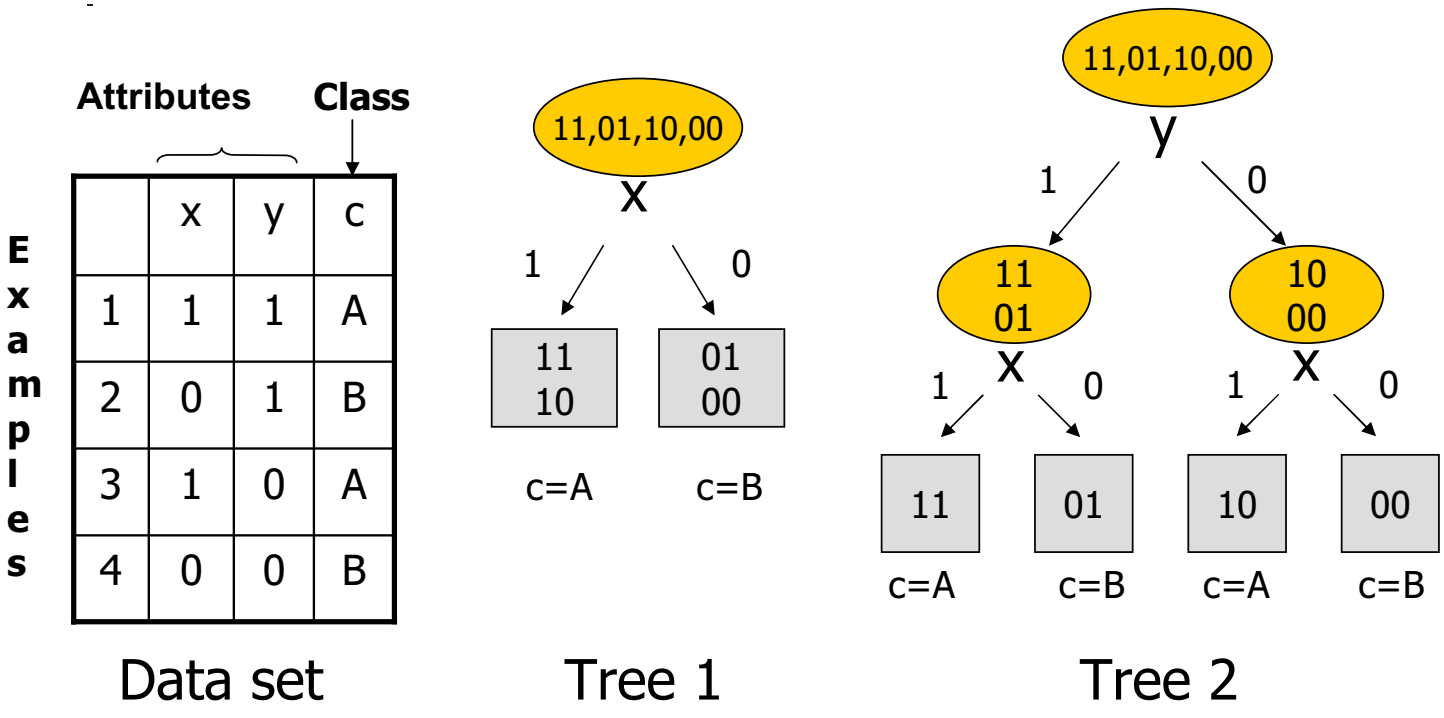
  1. The simplest DT is one that takes fewest bits to encode (least space, least memory)

  2. Searching for the simplest DT that is consistent with a training set is NP-hard. Solution:

     a Use a greedy heuristics, or

     b Restrict the hypothesis space to a subset of simple DT's

Any Boolean function can be represented by a DT



Data set Tree 1 Tree 2

When to Consider Decision Trees

- Instances describable by attribute–value pairs
- Target function is discrete valued
- Disjunctive hypothesis may be required
- Possibly noisy training data
- Examples of application domains
  1. Equipment or medical diagnosis
  2. Credit risk analysis
  3. Proteins function classifications and mutagenesis
  4. . . .
- DT learning algorithms
  1. Linear in size of the DT and the training set
  2. Produce comprehensible results
  3. Often among the first to be tried on a new data set

Building a Univariate Decision Tree

Given training set T for a concept with classes $C_1, \ldots, C_k$, there are 3 cases

1. T contains elements all belonging to the same class C_i: the DT for T is a leaf identifying class C_i

2. T contains no elements: the DT for T is a leaf, but the label is assigned heuristically, e.g. the majority class in the parent of this node

3. T contains elements from different classes: T is divided into subsets that seem to lead towards collections of elements. A test t based on a single attribute is chosen, and it partitions T into subsets $\{T_1, \ldots, T_n\}$. The DT consists of a decision node identifying the tested attribute, and one branch for each outcome of the test. Then, the same process is applied recursively to each T_j

Popular test: *Information Gain*

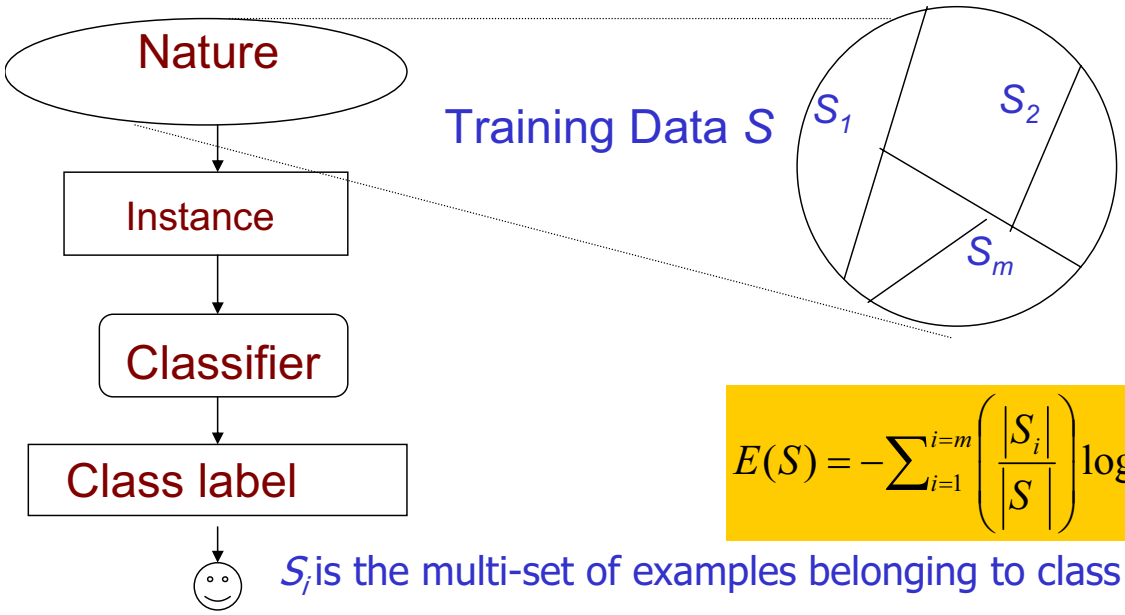
Information Gain Based DT Learner

- Learner's task is to extract needed information from a training set and store it in the form of a DT for classification

- Start with the entire training set at the root

  1. Recursively add nodes to the tree corresponding to tests that yield the greatest expected reduction in entropy (or the largest expected information gain)

  2. Until some termination criteria is met



$$E(S) = -\sum_{i=1}^{i=m} \left(\frac{|S_i|}{|S|} \right) \log_2 \left(\frac{|S_i|}{|S|} \right)$$

S_i is the multi-set of examples belonging to class C_i

On average, the information needed to convey the class membership of a random instance from S is $E(S)$

ID3: Top-Down Induction of Decision Trees

Main loop:

1. $A \leftarrow$ the "best" decision attribute for next $node$

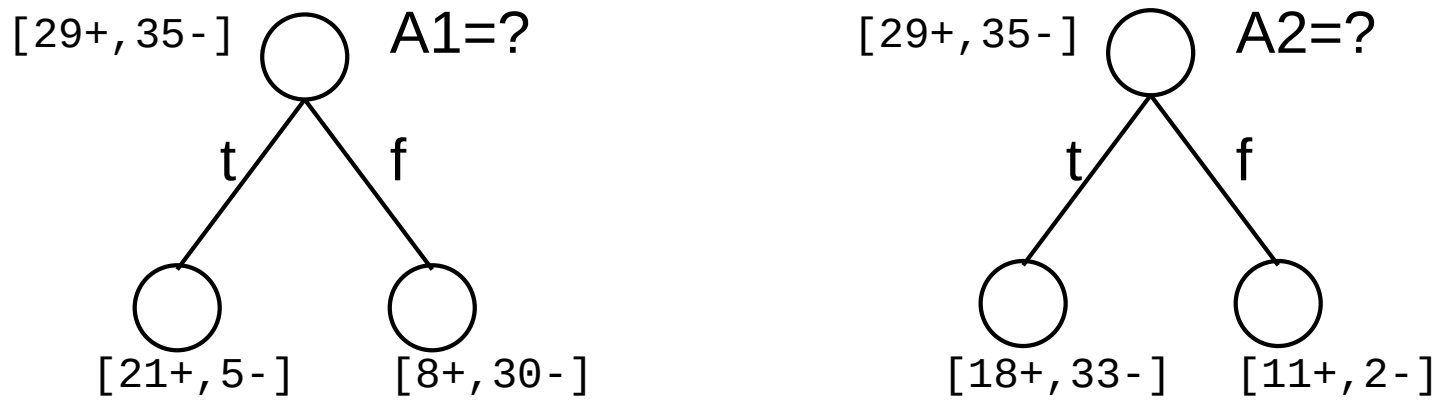
   e.g.: "best" = "highest information gain"

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

- 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

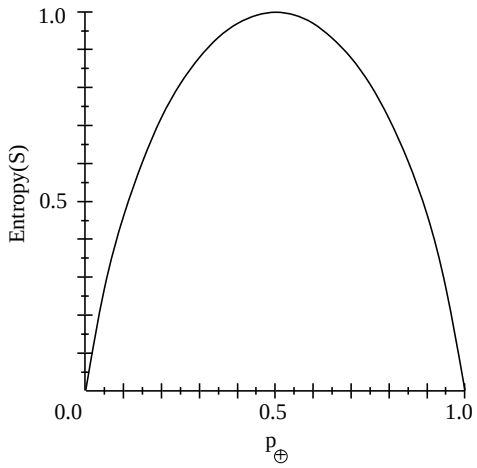
- Entropy measures the impurity of S

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

- Properties (for Boolean target):

  1. Entropy is 0 if S is totally unbalanced

  2. Entropy is 1 if S is totally balanced



Entropy

$Entropy(S)$ = expected number of bits needed to encode class ($\oplus$ or $\ominus$) of randomly drawn member of S (under the optimal, shortest-length code)

Why?

> Information theory: optimal length code assigns $-\log_2 p$ bits to message having probability p.

So, expected number of bits to encode $\oplus$ or $\ominus$ of random member of S:

$$p_\oplus(-\log_2 p_\oplus) + p_\ominus(-\log_2 p_\ominus)$$

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

In general, for a multi-class target with c classes

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

2. $0 \leq Entropy(S) \leq \log_2 c$

Entropy measures homogeneity of examples

Information Gain

- Expected entropy of S with respect to attribute A

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

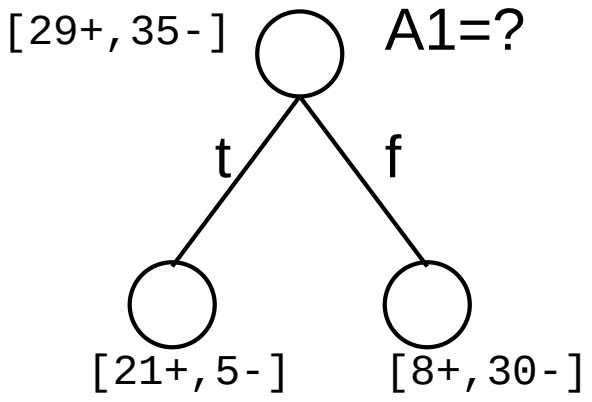
$Values(A)$ = set of all possible values of A

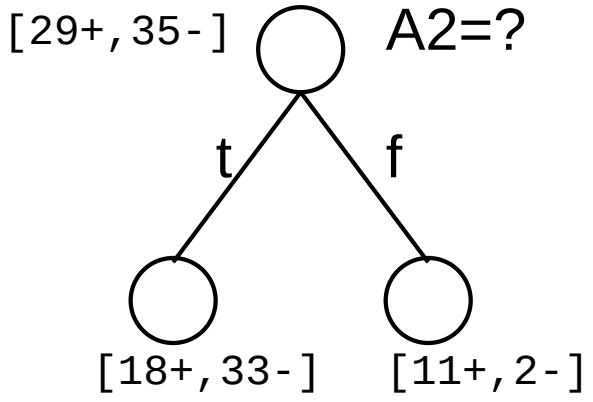
$S_v = \{s \in S | A(s) = v\}$, s = training instance

- Expected reduction in entropy of S given A

$$Gain(S, A) \equiv Entropy(S) - Entropy_A(S)$$

- To decide which attribute should be tested first, simply find the one with the highest information gain

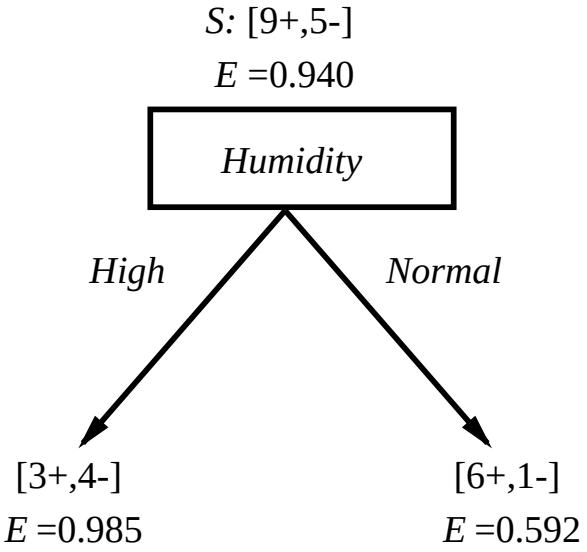


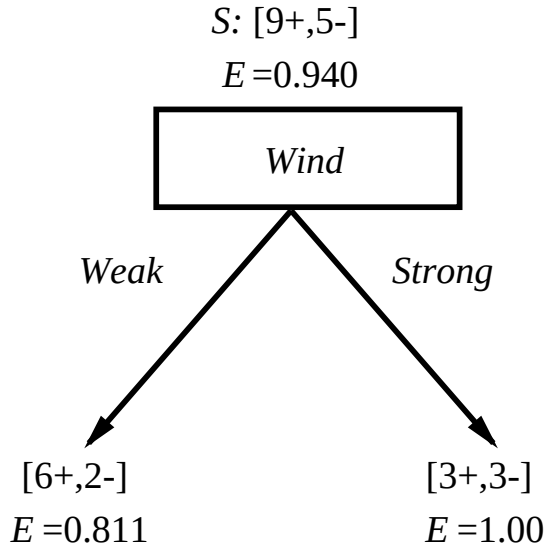


Decision Tree for $PlayTennis$?

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

Which attribute is the best classifier?





Gain (S, Humidity)

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

Gain (S, Wind)

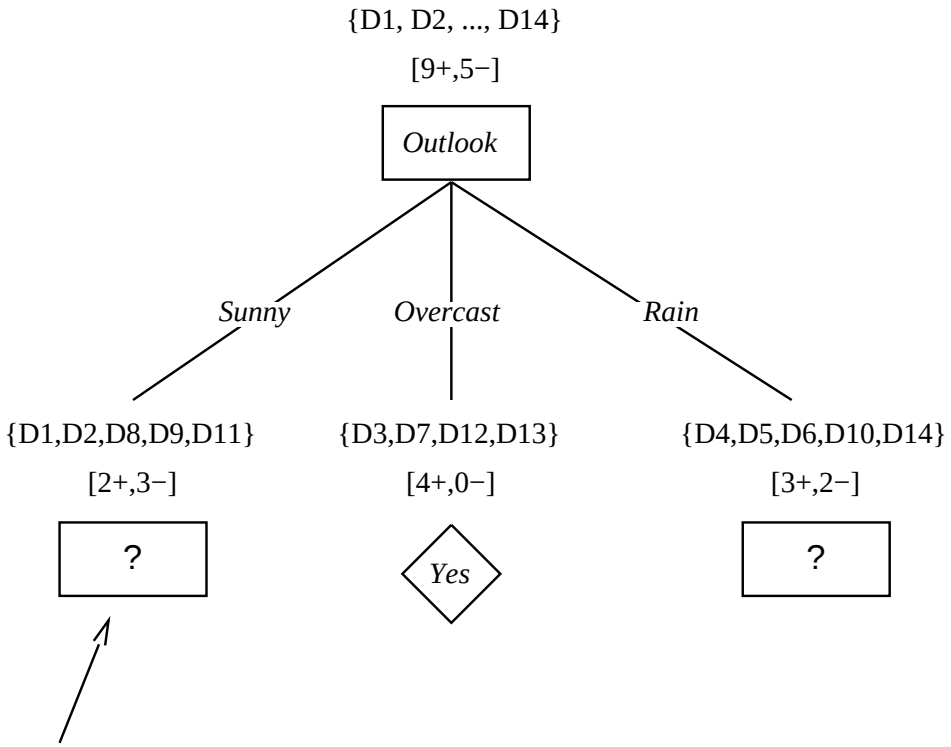
= .940 - (8/14).811 - (6/14)1.0
= .048

Selecting the Next Attribute for $PlayTennis$'s DT

Choose $Outlook$ as the top test, because

- $Gain(S, Outlook) = 0.246$
- $Gain(S, Humidity) = 0.151$
- $Gain(S, Wind) = 0.048$
- $Gain(S, Temperature) = 0.029$

Then repeat for each children of $Outlook$



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

Final Partition and Decision Tree for $PlayTennis$

Partition of cases:

outlook = sunny:

humidity $\leq$ 75:

Outlook	Temp (°F)	Humidity (%)	Windy?	Decision
sunny	75	70	true	Play
sunny	69	70	false	Play

humidity > 75:

Outlook	Temp (°F)	Humidity (%)	Windy?	Decision
sunny	80	90	true	Don't Play
sunny	85	85	false	Don't Play
sunny	72	95	false	Don't Play

outlook = overcast:

Outlook	Temp (°F)	Humidity (%)	Windy?	Decision
overcast	72	90	true	Play
overcast	83	78	false	Play
overcast	64	65	true	Play
overcast	81	75	false	Play

outlook = rain:

windy = true:

Outlook	Temp (°F)	Humidity (%)	Windy?	Decision
rain	71	80	true	Don't Play
rain	65	70	true	Don't Play

windy = false:

Outlook	Temp (°F)	Humidity (%)	Windy?	Decision
rain	75	80	false	Play
rain	68	80	false	Play
rain	70	96	false	Play

Corresponding decision tree:

```
outlook = sunny:
|   humidity <= 75: Play
|   humidity > 75: Don't Play
outlook = overcast: Play
outlook = rain:
|   windy = true: Don't Play
|   windy = false: Play
```

Summary: Basic ID3 Algorithm

BuildTree(*DataSet*, *Output*)

1. If all output values in *DataSet* are the same, return a leaf node that says

   "predict this unique output"

2. If all input values in *DataSet* are the same, return a leaf node that says

   "predict the majority output"

3. Else find attribute X with highest *Info Gain* (Suppose X has n_X distinct values)

   - Create and return a non-leaf node with n_X children

   - The i-th child should be built by calling

     BuildTree(DS_i, *Output*)

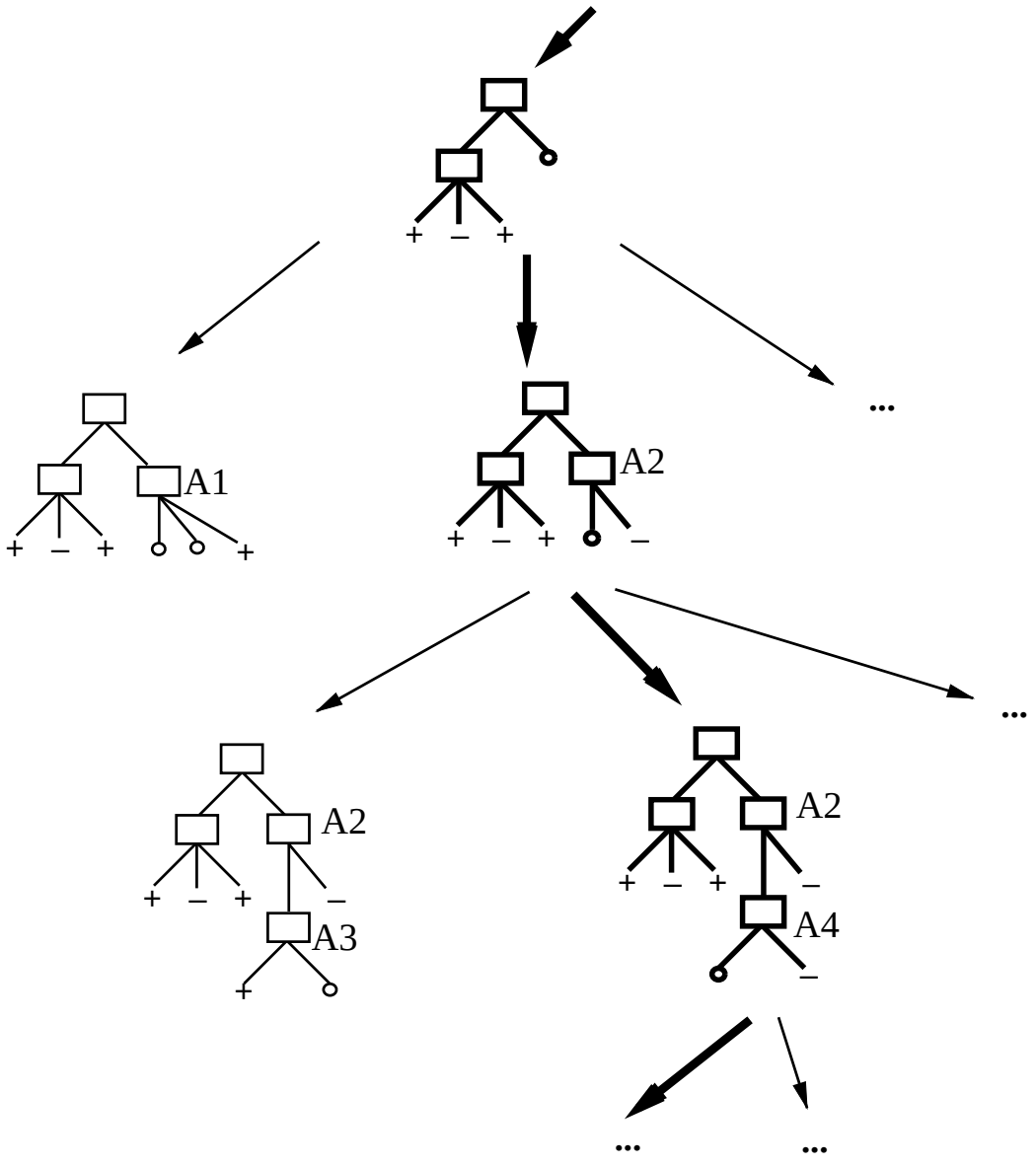
     Where DS_i set consists of all those records in *DataSet* for which $X = i$-th distinct value of X

Hypothesis Space Search by ID3

Hypothesis space of ID3: H = set of all possible DT's

ID3 is a search heuristic based on information gain

Simple-to-complex hill-climbing search through H



Hypothesis Space Search by ID3

- Hypothesis space is complete!
  - Target function surely in there . . .
- Outputs a single hypothesis (which one?)
  - Can't query the teacher . . .
  - Can't consider alternative DT's . . .
- Do not backtrack
  - May get stuck to local optima . . .
- Statistically-based search choices
  - Uses all training examples at each step
  - Less sensitive to errors in individual examples
  - Robust to noisy data . . .
- Approximate inductive bias: *prefer shorter trees*

Inductive Bias in ID3

- H = set of all consistent DT's from training set

  $\rightarrow$ Unbiased?

- Not really . . .

  1. Preference for short trees, and for those with high information gain attributes near the root

  2. Bias is a *preference* for some hypotheses, rather than a *restriction* of hypothesis space H

  3. *Occam's Razor Principle*: Prefer the shortest hypothesis that fits the data

- ID3 vs Candidate-Elimination

  1. ID3: Complete space, and, incomplete search

     $\rightarrow$ Preference bias

  2. C-E: Incomplete space, and, complete search

     $\rightarrow$ Restriction bias

Occam's Razor

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:

- 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??

Learning Errors

Training Set Error

- Apply learned DT on training examples
- Training error = Percentage of misclassification on training examples
- The smaller the better

Test Set Error

- Learning is not usually in order to predict the training data's output on data we have already seen
- It is more commonly in order to predict the output value for *future data* we have not yet seen
- Apply learned DT on test examples
- Test error = Percentage of misclassification on test examples
- The smaller the better

Two Definitions of Error

The **true error** of hypothesis h with respect to target function f and distribution $\mathcal{D}$ is the probability that h will misclassify an instance drawn at random according to $\mathcal{D}$.

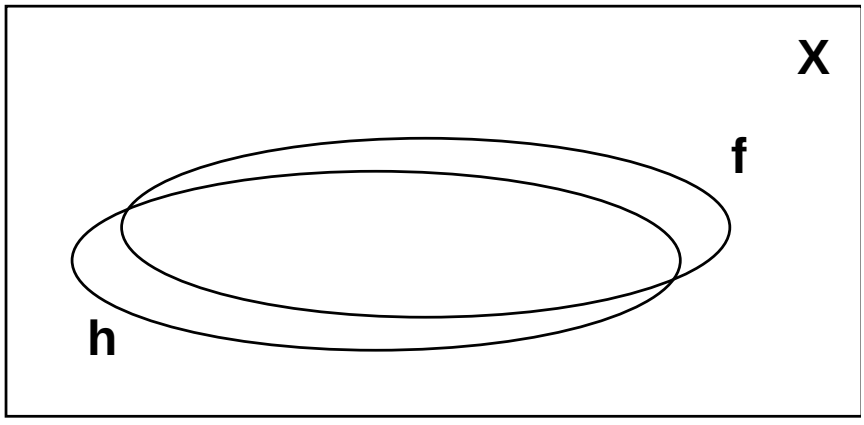
$$error_{\mathcal{D}}(h) \equiv \Pr_{x \in \mathcal{D}}[f(x) \neq h(x)]$$

The **sample error** of h with respect to target function f and data sample S is the proportion of examples h misclassifies

$$error_S(h) \equiv \frac{1}{|S|} \sum_{x \in S} \delta(f(x), h(x))$$

Where $\delta(a), b)$ is 1 if $a \neq b$, and 0 otherwise.

How well does $error_S(h)$ estimate $error_{\mathcal{D}}(h)$?



Problems Estimating Error

1 *Bias:* If S is training set, $error_S(h)$ is an optimistic estimate

$$bias \equiv E[error_S(h)] - error_{\mathcal{D}}(h)$$

For an *unbiased* estimate, h must be evaluated on an independent sample S (which is not the case if S is the training set!)

2 *Variance:* Even with unbiased S, $error_S(h)$ may still *vary* from $error_{\mathcal{D}}(h)$ and across samples!

Example: Hypothesis h misclassifies 12 of the 40 examples in S

$$error_S(h) = \frac{12}{40} = .30$$

- What is $error_{\mathcal{D}}(h)$?
- How close is $error_S(h)$ to $error_{\mathcal{D}}(h)$?
- Given observed $error_S(h)$ what can we conclude about $error_{\mathcal{D}}(h)$?

Confidence Intervals

There is an extensive literature on how to estimate a classifier's performance from samples and how to assign confidence to estimates (see Chapter 5). For instance

- If
  1. S contains n examples, drawn independently of h and each other
  2. $n \geq 30$
- Then

  With approximately N% probability, $error_{\mathcal{D}}(h)$ lies in interval

$$error_S(h) \pm z_N \sqrt{\frac{error_S(h)(1 - error_S(h))}{n}}$$

  where

N%:	50%	68%	80%	90%	95%	98%	99%
z_N:	0.67	1.00	1.28	1.64	1.96	2.33	2.58

Empirical Evaluation of a Classifier

Holdout Method (the usual approach):

1. Partition the set S into *training set* and *test set*
2. Use training set for learning, obtain an hypothesis H and set $Acc := 0$
3. For each element t in test set
   - Apply H on t
   - If $H(t) = label(t)$ then

     $Acc := Acc + 1$
4. $Accuracy := \frac{Acc}{\text{Size of test set}}$

We may be unlucky — training and test data may not be *representative*

Solution: Run multiple experiments with disjoint training and test data sets in which each class is represented in roughly the same proportion as in the entire data set

Empirical Evaluation of a Classifier

k-Fold Cross-Validation (recommended when data are limited):

1. Partition the set S into k equal parts, where each part has roughly the same class distribution as S
2. $Err := 0$
3. For $i = 1$ to k do
   - $S_{train} := S - S_i$ and $S_{test} := S_i$
   - $H := Learn(S_{train})$
   - $Err := Err + Error(H, S_{test})$
4. $Error := \frac{Err}{k}$

Better still: Repeat k-fold cross-validation r times and average the results

Leave-One-Out Cross-Validation :

- k-Fold cross-validation with $k = n$, where n is size of available data set
- Do n experiments — using $n - 1$ samples for training and the remaining sample for testing

Measuring Classifier's Performance

- Rigorous statistical evaluation is important
- How good is a learned hypothesis?
- Is one learning algorithm better than another?
- Different procedures for evaluation are appropriate under different conditions — Important to know when to use which evaluation method and be aware of pathological behavior

N: Total number of instances in the data set

TP_j: True positives for class j

FP_j : False positives for class j

TN_j: True Negatives for class j

FN_j: False Negatives for class j

$$Accuracy_j = \frac{TP_j + TN_j}{N}$$

$$Accuracy = \frac{\sum_j TP_j}{N}$$

$$Recall_j = \frac{TP_j}{TP_J + FN_j}$$

$$Precision_j = \frac{TP_j}{TP_J + FP_j}$$

$$FalseAlarm_j = \frac{FP_j}{TP_J + FP_j} = 1 - Precision_j$$

$$CorrelationCoeff_j = \frac{(TP_j \times TN_j) - (FP_j \times FN_j)}{\sqrt{(TP_J + FN_j)(TP_j + FP_j)(TN_j + FP_j)(TN_j + FN_j)}}$$

Measuring Classifier's Performance

$$(Micro)Average\ Precision = \frac{\sum_j TP_j}{\sum_j TP_j + \sum_j FP_j}$$

$$(Micro)Average\ Recall = \frac{\sum_j TP_j}{\sum_j TP_j + \sum_j FN_j}$$

Micro averaging gives equal importance to each instance $\Rightarrow$ classes with large number of instances dominate

$$(Micro)Average\ FalseAlarm == 1 - (Micro)Average\ Precision$$

$$(Micro)Average\ CorrelationCoeff = \frac{\left(\left(\sum_j TP_j\right)\times\left(\sum_j TN_j\right)\right) - \left(\left(\sum_j FP_j\right)\times\left(\sum_j FN_j\right)\right)}{\sqrt{\left(\sum_j TP_j + \sum_j FN_j\right)\left(\sum_j TP_j + \sum_j FP_j\right)\left(\sum_j TN_j + \sum_j FP_j\right)\left(\sum_j TN_j + \sum_j FN_j\right)}}$$

$$(Macro)Average\ Precision = \frac{1}{M}\sum_j Precision_j$$

$$(Macro)Average\ Recall = \frac{1}{M}\sum_j Recall_j$$

Macro averaging gives equal importance to class $\Rightarrow$ performance on classes with few instances is weighted as much as performance on classes with many instances

$$(Macro)Average\ FalseAlarm == 1 - (Macro)Average\ Precision$$

$$(Macro)Average\ CorrelationCoeff = \frac{1}{M}\sum_j CorrelationCoeff_j$$

Precision is sometimes called specificity

and Recall is sometimes called sensitivity

Measuring Classifier's Performance

Confusion Matrix: A matrix showing the predicted and actual classifications. A confusion matrix is of size $c \times c$ where c is the number of different label values (i.e. classes).

- Confusion matrix for $c = 2$:

		Predicted	
		$H+$	$H-$
Actual	$T+$	T_P	F_N
	$T-$	F_P	T_N

- Confusion matrix for $c = 3$ (not really!)

Guess / True	$\neg O_1$	O_1
$\neg C_1$	70	10
C_1	5	15

Guess / True	$\neg O_2$	O_2
$\neg C_2$	40	20
C_2	14	26

Guess / True	$\neg O_3$	O_3
$\neg C_3$	61	9
C_3	19	11

$$precision\ P_1 = \frac{15}{15+10} \qquad recall\ R_1 = \frac{15}{15+5} \qquad P_{micro} = \frac{15+26+11}{15+26+11+10+20+9}$$

$$precision\ P_2 = \frac{26}{26+20} \qquad precision\ P_2 = \frac{26}{26+14} \qquad R_{micro} = \frac{15+26+11}{15+26+11+5+14+19}$$

$$precision\ P_3 = \frac{11}{11+9} \qquad precision\ P_3 = \frac{11}{11+19} \qquad Accuracy = \frac{15+26+11}{100}$$

Measuring Classifier's Performance

Measures defined for a 2×2 confusion matrix

- Accuracy $= \frac{T_P+T_N}{T_P+F_N+F_P+T_N}$ = Efficiency
- Sensitivity $= \frac{T_P}{F_N+T_P}$ = True positive rate, Recall
- Specificity $= \frac{T_N}{T_N+F_P}$ = True negative rate
- Precision $= \frac{T_P}{F_P+T_P}$
- False positive rate $= \frac{F_P}{T_N+F_P}$
- False negative rate $= \frac{F_N}{F_N+T_P}$
- Coverage: The proportion of a data set for which a classifier makes a prediction. If a classifier does not classify all the instances, it may be important to know its performance on the set of cases for which it is *confident* enough to make a prediction

		Predicted	
		$H+$	$H-$
Actual	$T+$	T_P	F_N
	$T-$	F_P	T_N

Measuring Classifier's Performance

- The confusion matrix contains all the information needed to assess the performance of binary classifiers

- Measures like *accuracy*, *sensitivity*, *specificity* and *precision* summarize this information in a single scalar. Any such summary necessarily loses information

- Each measure is useful in its own way, but must be used with care; for example, accuracy is misleading when data set has uneven proportion of examples of different classes

- If a single measure of performance is to be reported, perhaps one of the least biased and the most useful measure is the *correlation coefficient*; value of 1 corresponds to the perfect classifier , 0 correspond to random predictions

- *correlation coefficient* can be defined for the case of classifiers with c classes

- It is often possible to trade off different measures

Overfitting

- Consider error of hypothesis h over

  1. Training data: $error_{train}(h)$

  2. Entire distribution $\mathcal{D}$ of data: $error_{\mathcal{D}}(h)$

  Hypothesis $h \in H$ **over fits** training data if there is an alternative hypothesis $h' \in H$ such that

  $error_{train}(h) < error_{train}(h')$ and $error_{\mathcal{D}}(h) > error_{\mathcal{D}}(h')$

- Causes of overfitting

  1. Noisy samples

  2. Irrelevant variables

  3. Complex hypothesis (e.g./ VS, DT, NN, . . .)

  4. As we move further away from the root, the data set used to choose the best test becomes smaller $\rightarrow$ poor estimate of entropy

- Example: Add the noisy training example #15:

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

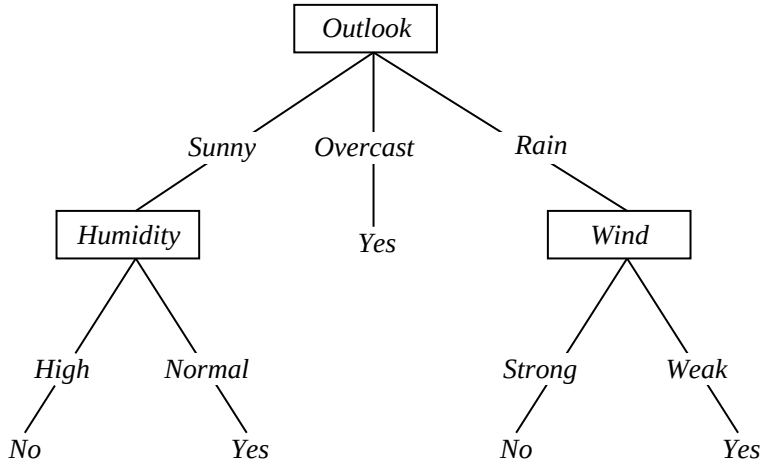
  What effect on earlier tree?

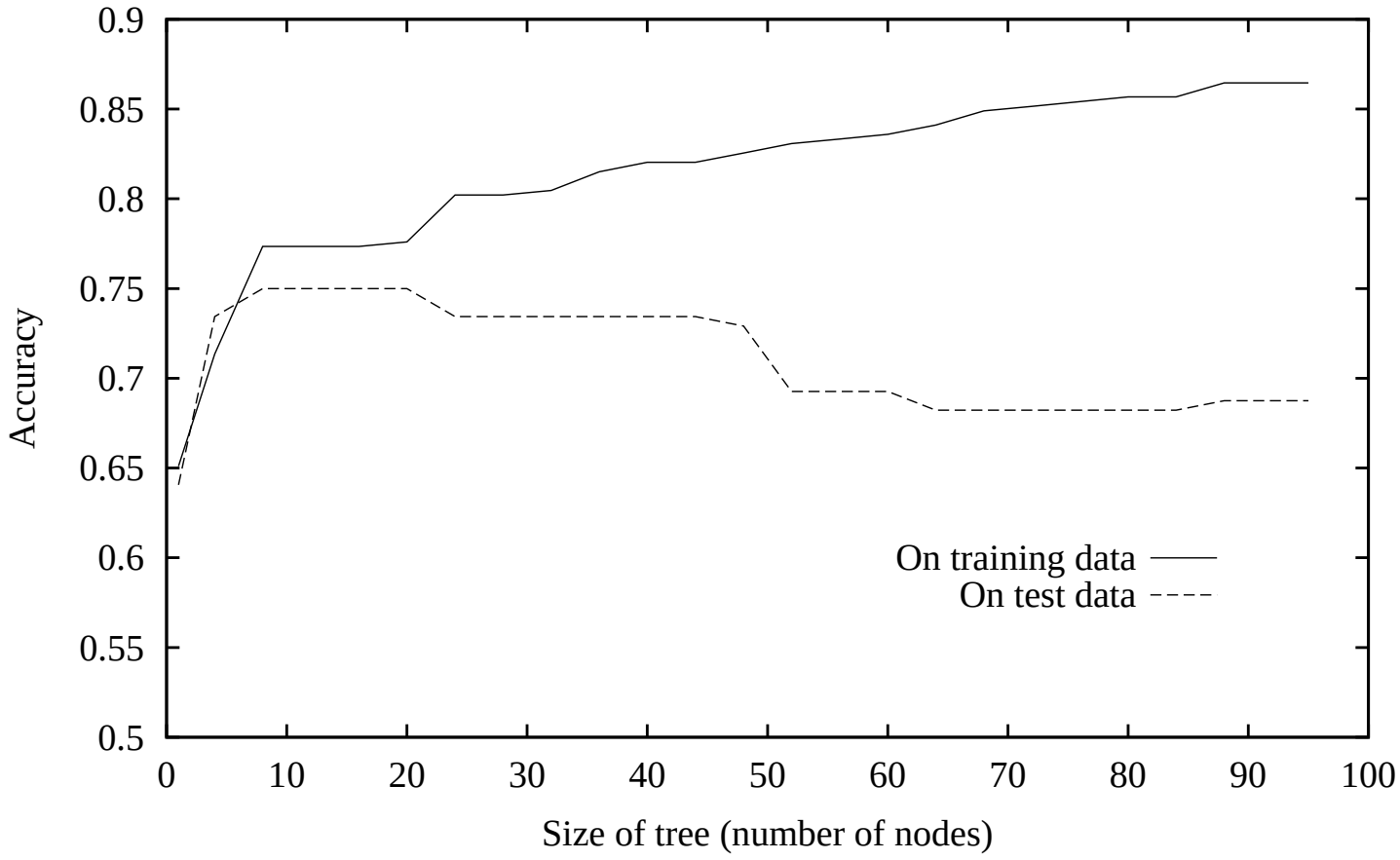
Overfitting in Decision Tree Learning

Consider adding noisy training example #15:

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

What effect on earlier tree below?





Avoiding Overfitting in DT learning

- How can we avoid overfitting?

  1. Early stopping: Stop growing when data split not statistically significant

     That is: When further split fails to yield statistically significant information gain estimated from validation set

  2. Pruning: Grow full tree, then post-prune

  Use roughly the same size sample at every node to estimate the entropy — when there is a large data set from which we can sample

- How to select "best" tree (that minimizes overfitting)?

  1. Measure performance over training data
  2. Measure performance over separate validation data set

     MDL: minimize $size(tree) + size(misclassifications(tree))$

Reduced-Error Pruning

Each node is a candidate for pruning
Pruning a DT node consists of

- Removing the sub-tree rooted at that node
- Making it a leaf
- Assigning it the most common label at that node

Reduced-Error Pruning

Split data into *training* and *validation* set and
Do until further pruning is harmful:

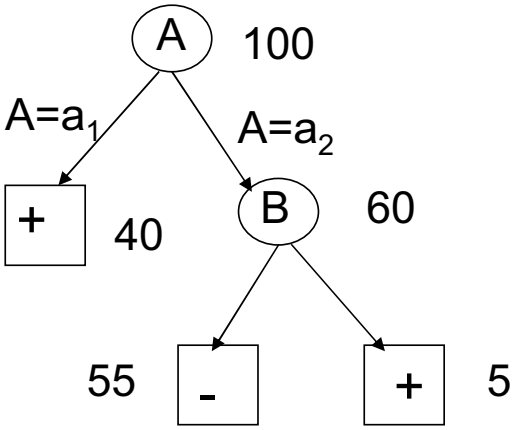
1. Evaluate impact on *validation* set of pruning each candidate node (plus those below it)

2. Greedily select a node which most improves the performance on the *validation* set when the sub-tree rooted at that node is pruned

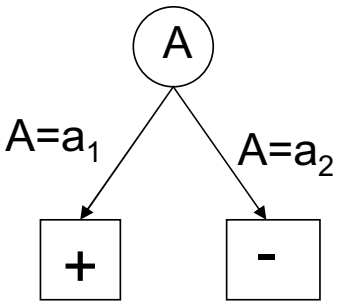
Produces smallest version of most accurate subtree

Drawback: Holding back the validation set limits the amount of training data available; not desirable when data set is small

Reduced-Error Pruning — Example



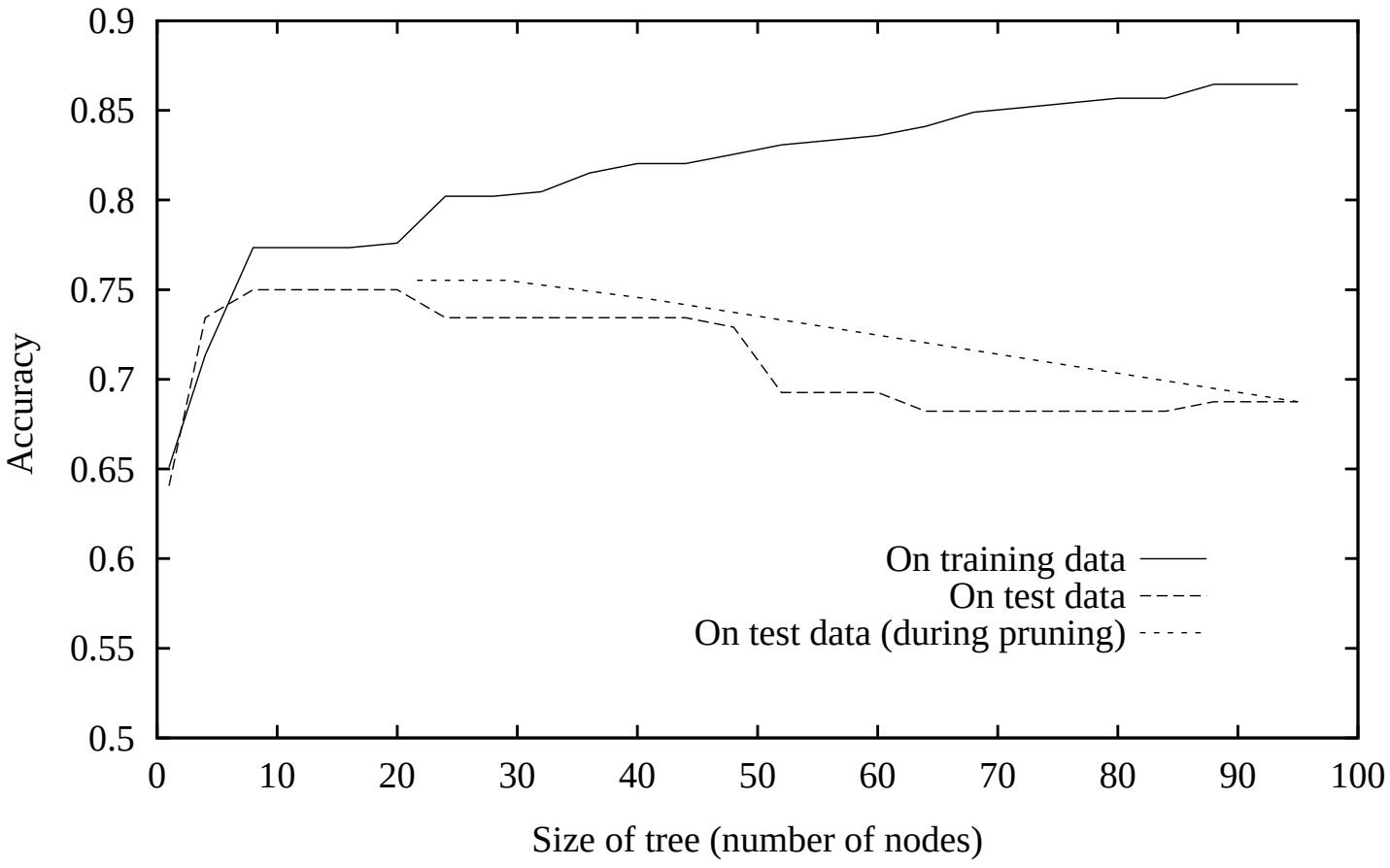
Node	Accuracy gain by Pruning
A	-20%
B	+10%



Before Pruning

After Pruning

Effect of Reduced-Error Pruning

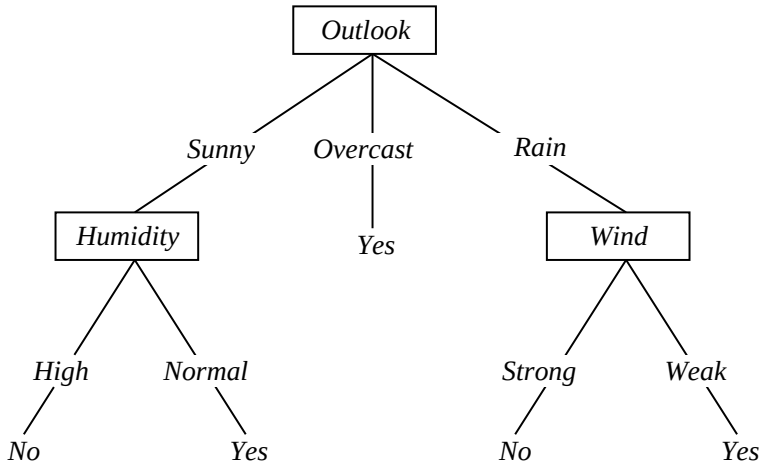


Rule Post-Pruning

1. Convert tree to equivalent set of rules

2. Prune each rule independently of others

3. Sort final rules in order of lowest to highest error



If $(Outlook = Sunny) \wedge (Humidity = High)$
Then $PlayTennis = No$
If $(Outlook = Sunny) \wedge (Humidity = Normal)$
Then $PlayTennis = Yes$

Advantage: can potentially correct bad choices made close to the root

Post pruning based on validation set is the most commonly used method in practice (e.g., C4.5)

Development of *pre-pruning* methods with comparable performance that do not require a validation set is an open problem

Continuous Valued Attributes

Create a discrete attribute to test continuous

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

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

Candidate splits $T > \frac{48+50}{2}$? or $\frac{60+70}{2}$?

1 Sort instances by value of continuous attribute under consideration, and obtain all candidate thresholds; e.g. $T_{>54}$ and $T_{>85}$ are the candidate discrete attributes

2 Compute the information gain for each of the candidate discrete attribute

3 Chose the best candidate discrete attribute and . . .

. . . use it as any normal discrete attribute for growing a DT

Attributes with Many Values

Problem:

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

One solution: use $GainRatio$ instead of $Gain$

- $GainRatio$ shows the proportion of information generated by the split that is useful for classification

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

- $SplitInformation$ measures potential information generated by dividing S into c classes

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

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

Attributes with Costs

Not all attributes are equally costly or risky

- In medical diagnosis: $BloodTest$ has cost \$150

Goal: Learn a DT which minimizes cost of classification

Solutions:

1. Replace gain by
   - Tan and Schlimmer (1990)
   $$\frac{Gain^2(S, A)}{Cost(A)}$$
   - Nunez (1988)
   $$\frac{2^{Gain(S,A)} - 1}{(Cost(A) + 1)^w}$$
   Where $w \in [0, 1]$ determines importance of cost
2. Use a cost matrix: Not all misclassifications are equally costly (e.g. a false alarm is less costly than a failure in a nuclear power plant)
   - Cost matrix is like confusion matrix, except cost of errors are assigned to the elements outside the main diagonal (misclassifications). Used for diagnosis problems

Unknown/Missing Attribute Values

- Sometimes, the fact that an attribute value is missing might itself be informative

  Missing blood sugar level might imply that the physician had reason not to measure it

- Solution 1:

  1. Introduce a new value (one per attribute) to denote a missing value

  2. DT construction and its use for classification proceed as before

- Solution 2:

  During DT construction: Replace a missing value in a training example with the most frequent value found among the instances at the node

  During use of DT for classification: Replace a missing value in an instance to be classified with the most frequent value found among the training instances at the node

Unknown/Missing Attribute Values

- Solution 3:

  During DT construction: Replace a missing value in a training example with the most frequent value found among the instances at the node that have the same class label as the training example

  During use of DT for classification:

  1. Assign to a missing value the most frequent value found at the node (based on the training sample)

  2. Sort the instance through the DT to generate the class label

- Solution 4:

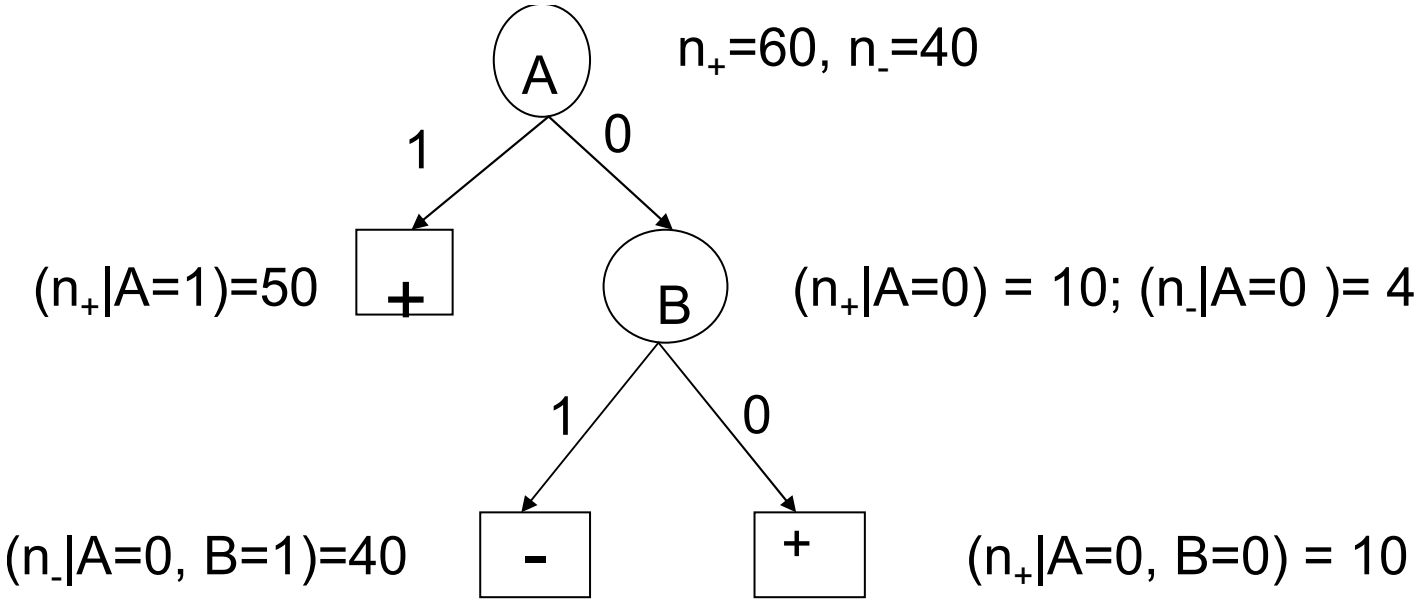
  During DT construction: Generate several fractionally weighted training examples based on the distribution of values for the corresponding attribute at the node

  During use of DT for classification:

  1. Generate multiple *instances* by assigning candidate values for the missing attribute based on the distribution of instances at the node

  2. Sort each instance through the DT to generate candidate labels and assign the most probable class label or probabilistically assign class label

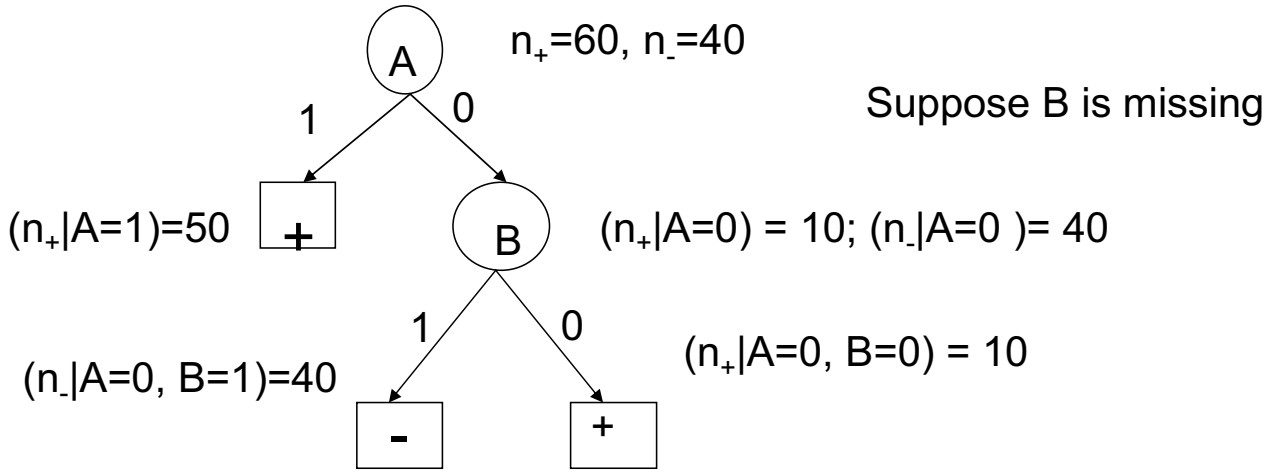
Unknown/Missing Attribute Values



Suppose B is missing

Replacement with most frequent value at the node → B=1

Replacement with most frequent value if class is + → B=0



Fractional instance based on distribution at the node .. 4/5 for B=1, 1/5 for B=0

Fractional instance based on distribution at the node for class + ..

1/5 for B=0. 0 for B=1

PETs — Probability Estimation Trees

Accuracy does not consider the probability of prediction, so in PETs

- Instead of predicting a class, the leaves give a probability
- Very useful when we do not want just the class, but examples most likely to belong to a class
- No additional effort in learning PETs compared to DTs
- Require different evaluation methods
- Bayesian DTs