Decision tree learning

Biplab Banerjee

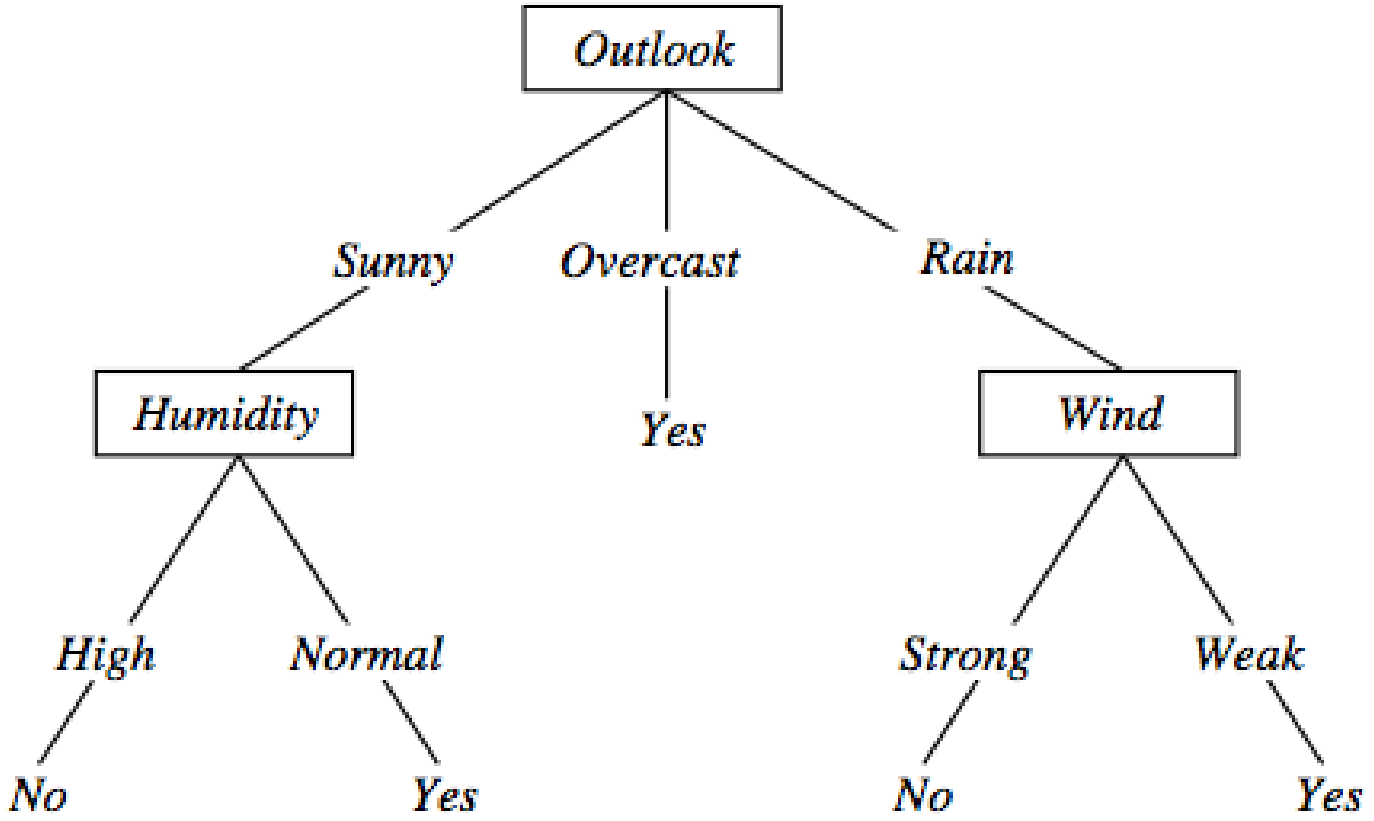
Inductive inference with decision trees

- *Decision Trees* is one of the most widely used *inductive inference.*
- Pros:
  - Method for approximating *discrete-valued* functions (including boolean)
  - Learned functions are represented as *decision trees* (or *if-then-else* rules*)*
  - Expressive hypotheses space

Example

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

Decision tree representation (PlayTennis)

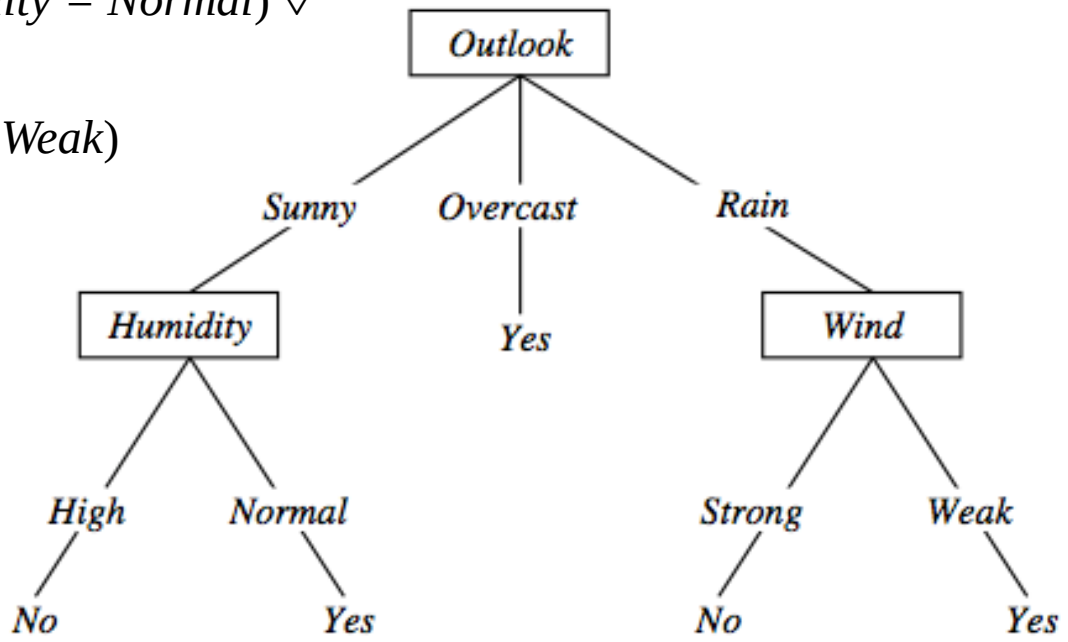


⟨*Outlook=Sunny, Temp=Hot, Humidity=High, Wind=Strong*⟩ No

Decision trees expressivity

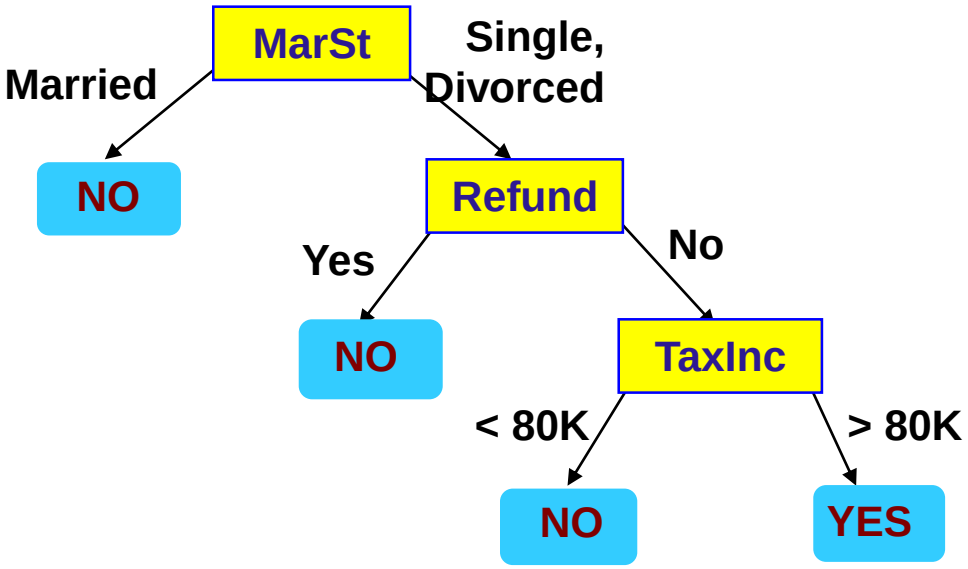
- Decision trees represent a disjunction of conjunctions on constraints on the value of attributes:

$(Outlook = Sunny \wedge Humidity = Normal) \vee$
$(Outlook = Overcast) \vee$
$(Outlook = Rain \wedge Wind = Weak)$



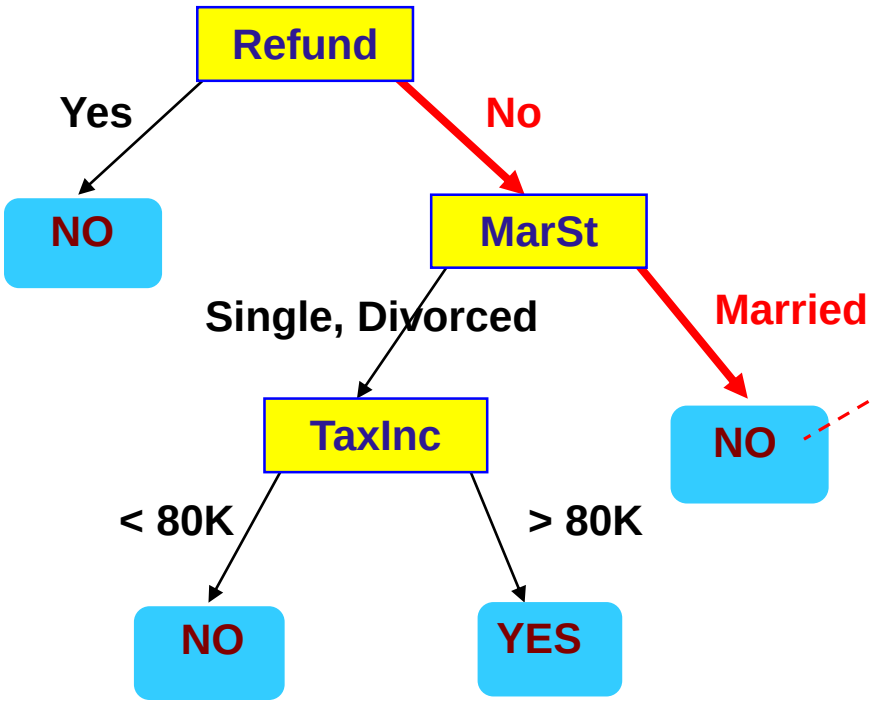
Another Example of Decision Tree

	categorical	categorical	continuous	class
Tid	**Refund**	**Marital Status**	**Taxable Income**	**Cheat**
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes



There could be more than one tree that fits the same data!

Apply Model to Test Data



Test Data

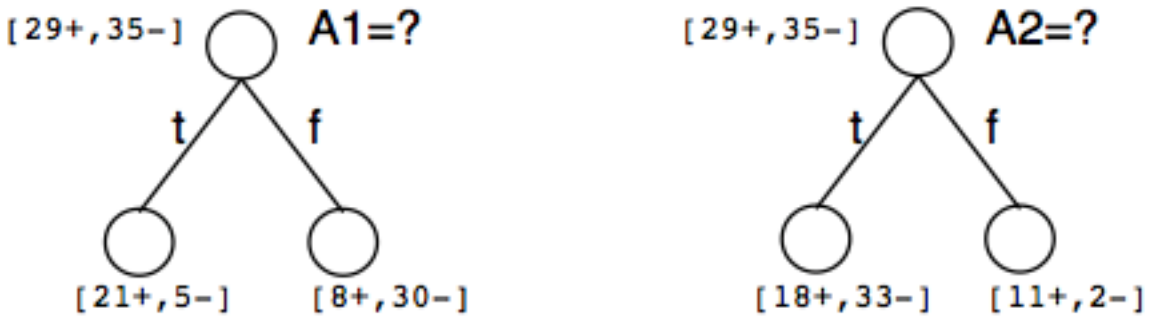
Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

Assign Cheat to "No"

Top-down induction of Decision Trees

- ID3 (Quinlan, 1986) is a basic algorithm for learning DT's
- Given a training set of examples, the algorithms for building DT performs search in the space of decision trees
- The construction of the tree is top-down. The algorithm is greedy.
- The fundamental question is "which attribute should be tested next? Which question gives us more information?"
- A descendent node is then created for each possible value of this attribute and examples are partitioned according to this value
- The process is repeated for each successor node until all the examples are classified correctly or there are no attributes left

Which attribute is the best classifier?



- A statistical property called *information gain*, measures how well a given attribute separates the training examples
- Information gain uses the notion of *entropy*, commonly used in information theory
- *Information gain = expected reduction of entropy*

Entropy in binary classification

- Entropy measures the *impurity* of a collection of examples. It depends from the distribution of the random variable *p.*
  - S is a collection of training examples
  - p_+ the proportion of positive examples in S
  - p_- the proportion of negative examples in S

$$Entropy\ (S) \equiv\ -p_+\ log_2\ p_+ - p_-log_2 p_- \quad [0\ log_2 0 = 0]$$

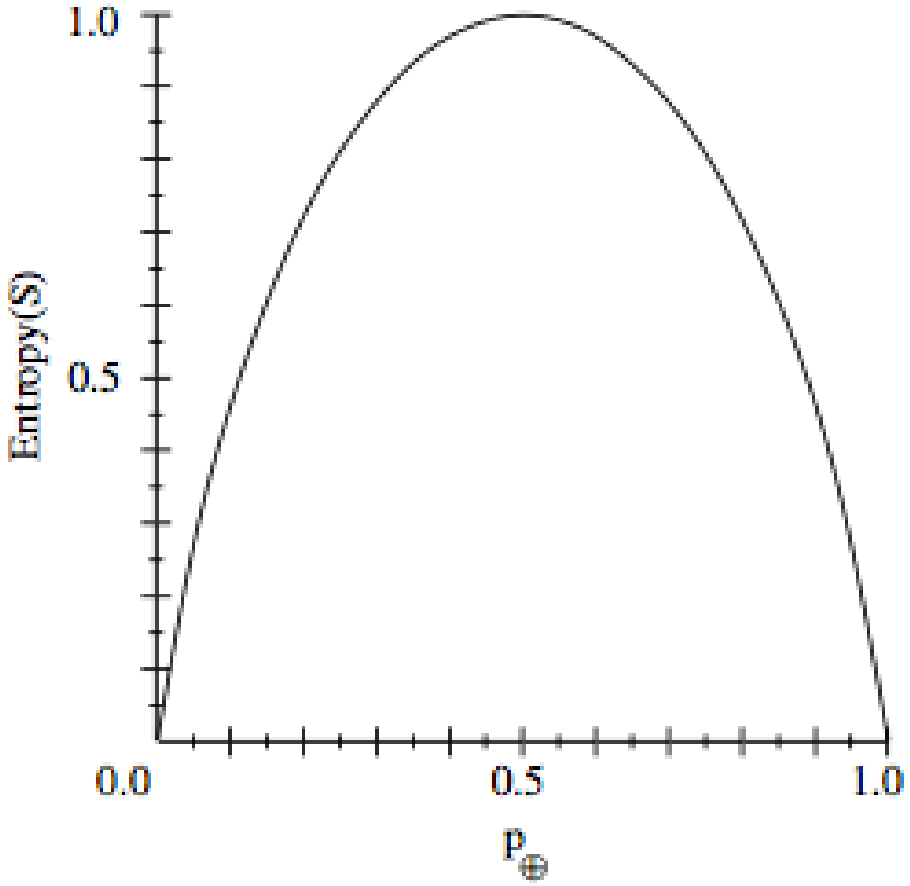
$$Entropy\ ([14+,\ 0-]) = -\ 14/14\ log_2\ (14/14) -\ 0\ log_2\ (0) = 0$$

$$Entropy\ ([9+,\ 5-]) = -\ 9/14\ log_2\ (9/14) -\ 5/14\ log_2\ (5/14) = 0{,}94$$

$$Entropy\ ([7+,\ 7-]) = -\ 7/14\ log_2\ (7/14) -\ 7/14\ log_2\ (7/14) =$$

$$= 1/2 + 1/2 = 1 \qquad [log_2 1/2 = -1]$$

Entropy



Information gain as *entropy reduction*

- *Information gain* is the *expected* reduction in entropy caused by partitioning the examples on an attribute.
- The higher the information gain the more effective the attribute in classifying training data.
- Expected reduction in entropy knowing *A*

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

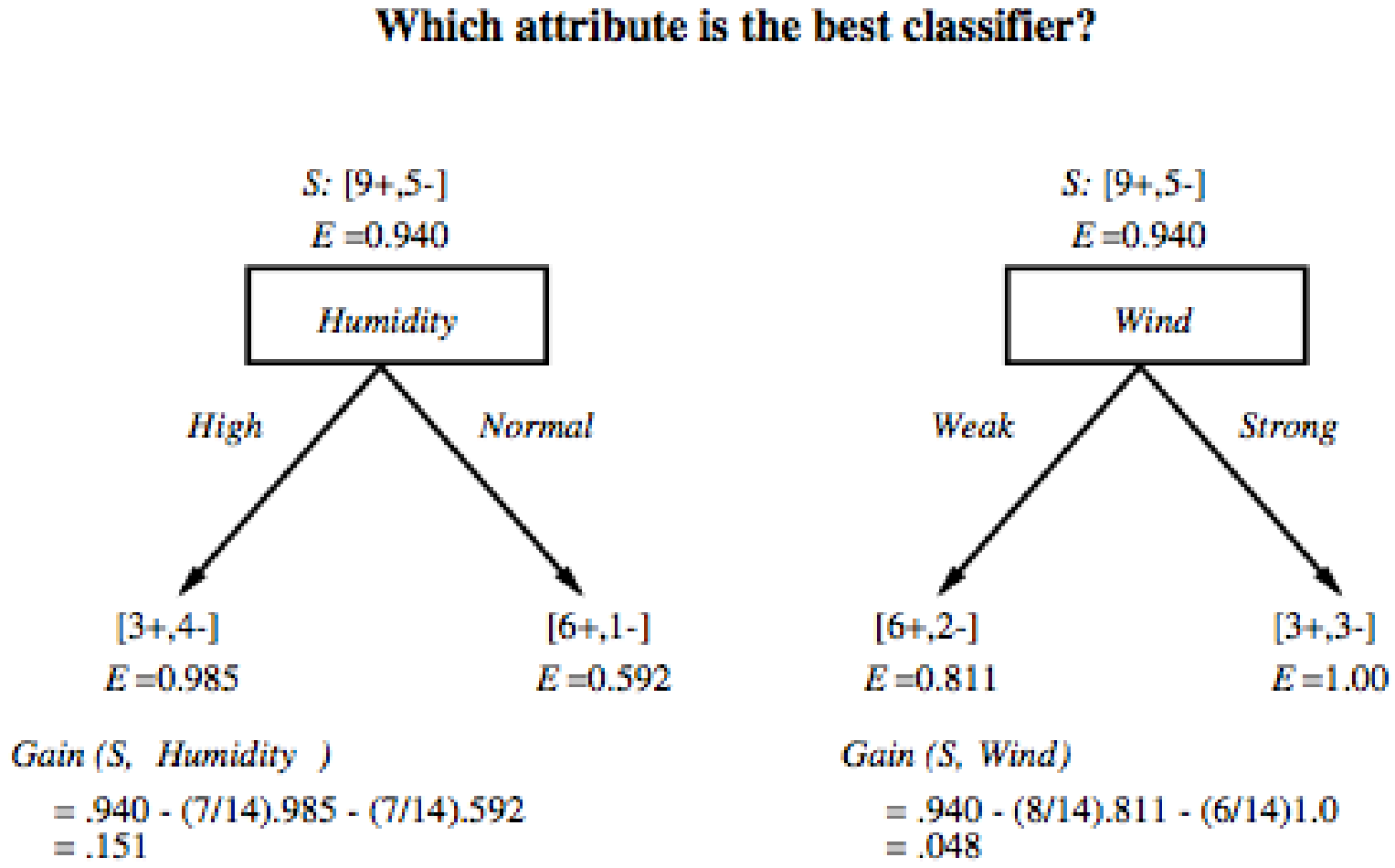
$Values(A)$ possible values for A
Sv subset of S for which A has value v

Example: expected information gain

- Let
  - $Values(Wind) = \{Weak, Strong\}$
  - $S = [9+, 5-]$
  - $S_{Weak} = [6+, 2-]$
  - $S_{Strong} = [3+, 3-]$
- Information gain due to knowing *Wind*:

$$Gain(S, Wind) = Entropy(S) - 8/14\ Entropy(S_{Weak}) - 6/14\ Entropy(S_{Strong})$$
$$= 0{,}94 - 8/14 \times 0{,}811 - 6/14 \times 1{,}00$$
$$= 0{,}048$$

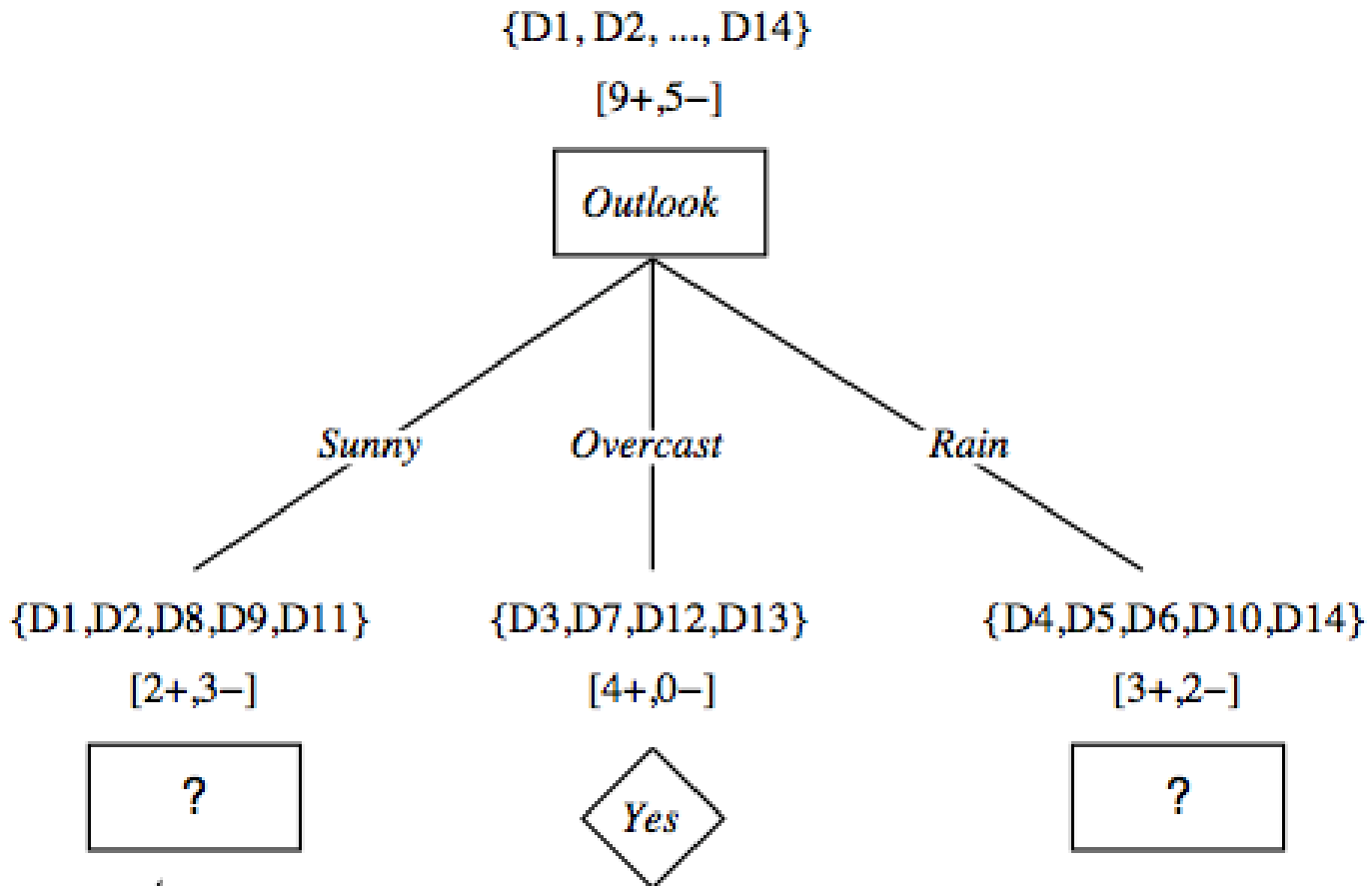
Which attribute is the best classifier?



First step: which attribute to test at the root?

- Which attribute should be tested at the root?
  - $Gain(S, Outlook) = 0.246$
  - $Gain(S, Humidity) = 0.151$
  - $Gain(S, Wind) = 0.084$
  - $Gain(S, Temperature) = 0.029$
- *Outlook* provides the best prediction for the target
- Lets grow the tree:
  - add to the tree a successor for each possible value of *Outlook*
  - partition the training samples according to the value of *Outlook*

After first step



Second step

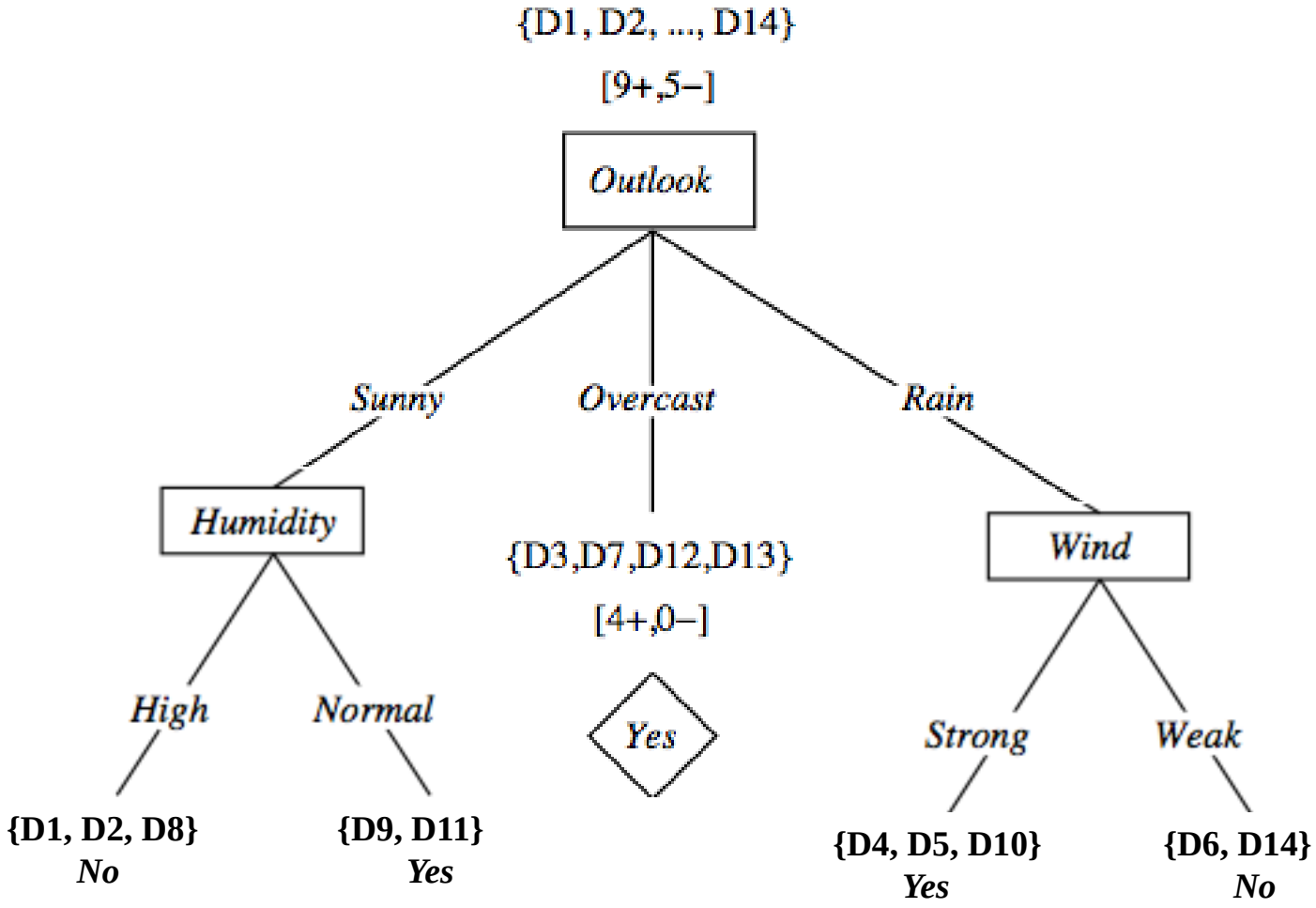
- Working on $Outlook{=}Sunny$ node:

  $Gain(S_{Sunny}, Humidity) = 0.970 - 3/5 \times 0.0 - 2/5 \times 0.0 = 0.970$

  $Gain(S_{Sunny}, Wind) = 0.970 - 2/5 \times 1.0 - 3.5 \times 0.918 = 0\ .019$

  $Gain(S_{Sunny}, Temp.) = 0.970 - 2/5 \times 0.0 - 2/5 \times 1.0 - 1/5 \times 0.0 = 0.570$
- $Humidity$ provides the best prediction for the target
- Lets grow the tree:
  - add to the tree a successor for each possible value of $Humidity$
  - partition the training samples according to the value of $Humidity$

Second and third steps



Measure of Impurity: GINI

- Gini Index for a given node t :

$$GINI(t) = 1 - \sum_j [p(j \mid t)]^2$$

(NOTE: $p(j \mid t)$ is the relative frequency of class j at node t).

  - Maximum ($1 - 1/n_c$) when records are equally distributed among all classes, implying least interesting information
  - Minimum (0.0) when all records belong to one class, implying most interesting information

C1	**0**
C2	**6**
Gini=0.000	

C1	**1**
C2	**5**
Gini=0.278	

C1	**2**
C2	**4**
Gini=0.444	

C1	**3**
C2	**3**
Gini=0.500	

Examples for computing GINI

$$GINI(t) = 1 - \sum_j [p(j \mid t)]^2$$

C1	**0**
C2	**6**

P(C1) = 0/6 = 0 **P(C2) = 6/6 = 1**

Gini = 1 – P(C1)^2 – P(C2)^2 = 1 – 0 – 1 = 0

C1	**1**
C2	**5**

P(C1) = 1/6 **P(C2) = 5/6**

Gini = 1 – (1/6)^2 – (5/6)^2 = 0.278

C1	**2**
C2	**4**

P(C1) = 2/6 **P(C2) = 4/6**

Gini = 1 – (2/6)^2 – (4/6)^2 = 0.444

Splitting Based on GINI

- Used in CART, SLIQ, SPRINT.
- When a node p is split into k partitions (children), the quality of split is computed as,

$$GINI_{split} = \sum_{i=1}^{k} \frac{n_i}{n} GINI(i)$$

where, n_i = number of records at child i,

n = number of records at node p.

https://blog.quantinsti.com/gini-index/

ID3: algorithm

ID3(*X, T, Attrs*) *X*: training examples:

T: target attribute (e.g. $PlayTennis$),

Attrs: other attributes, initially all attributes

Create *Root* node

If all X's are +, *return Root* with class +

If all X's are –, *return Root* with class –

If Attrs is empty *return Root* with class most common value of *T* in *X*

else

$A \leftarrow$ best attribute; decision attribute for Root $\leftarrow A$

For each possible value v_i of *A:*

- add a new branch below *Root,* for test $A = v_i$
- $X_i \leftarrow$ subset of *X* with $A = v_i$
- *If* X_i is empty *then* add a new leaf with class the most common value of *T* in *X*

else add the subtree generated by ID3(X_i, *T, Attrs* – {*A*})

return Root

Prefer shorter hypotheses: Occam's razor

- Why prefer shorter hypotheses?
- Arguments in favor:
  - There are fewer short hypotheses than long ones
  - If a short hypothesis fits data unlikely to be a coincidence
  - Elegance and aesthetics
- Arguments against:
  - Not every short hypothesis is a reasonable one.

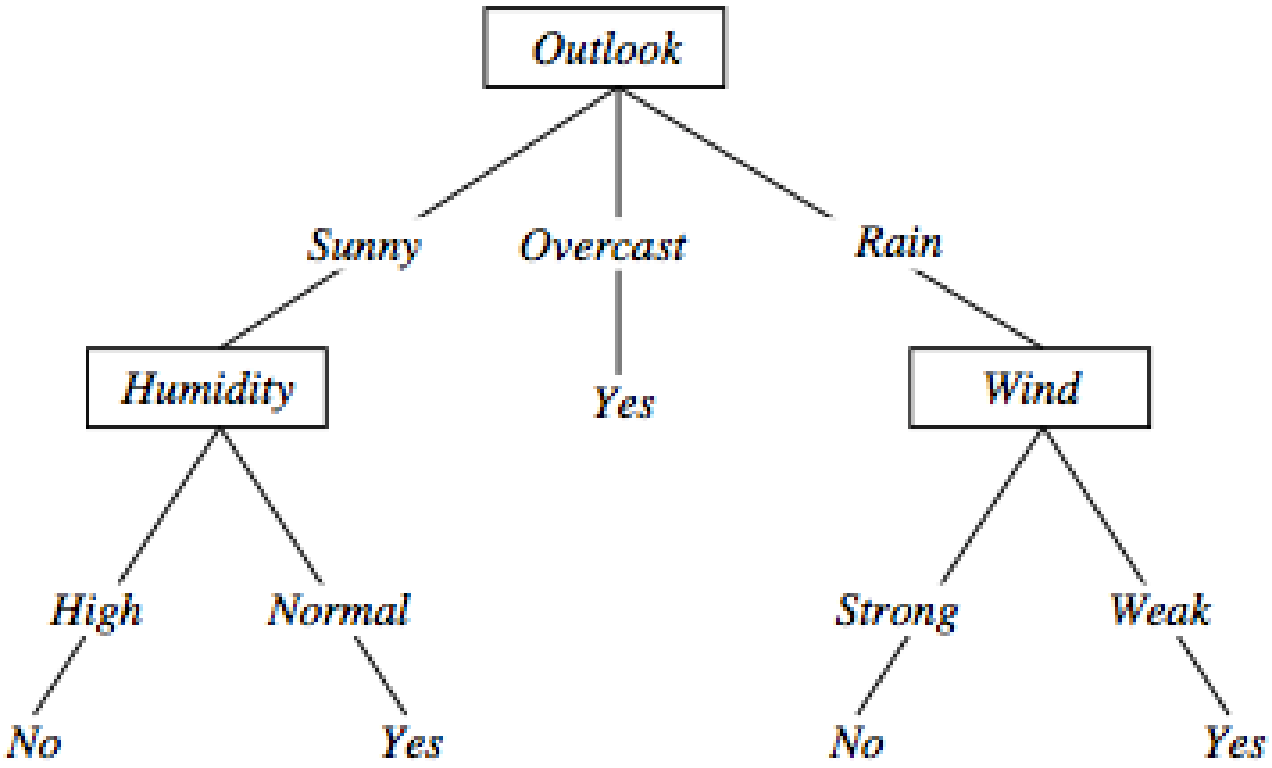
Issues in decision trees learning

- Overfitting
  - Reduced error pruning
  - Rule post-pruning
- Extensions
  - Continuous valued attributes
  - Alternative measures for selecting attributes
  - Handling training examples with missing attribute values
  - Handling attributes with different costs
  - Improving computational efficiency
  - Most of these improvements in C4.5 (Quinlan, 1993)

Example

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
D15	Sunny	Hot	Normal	Strong	No

Overfitting in decision trees



⟨*Outlook=Sunny, Temp=Hot, Humidity=Normal, Wind=Strong, PlayTennis=No* ⟩

New noisy example causes splitting of second leaf node.

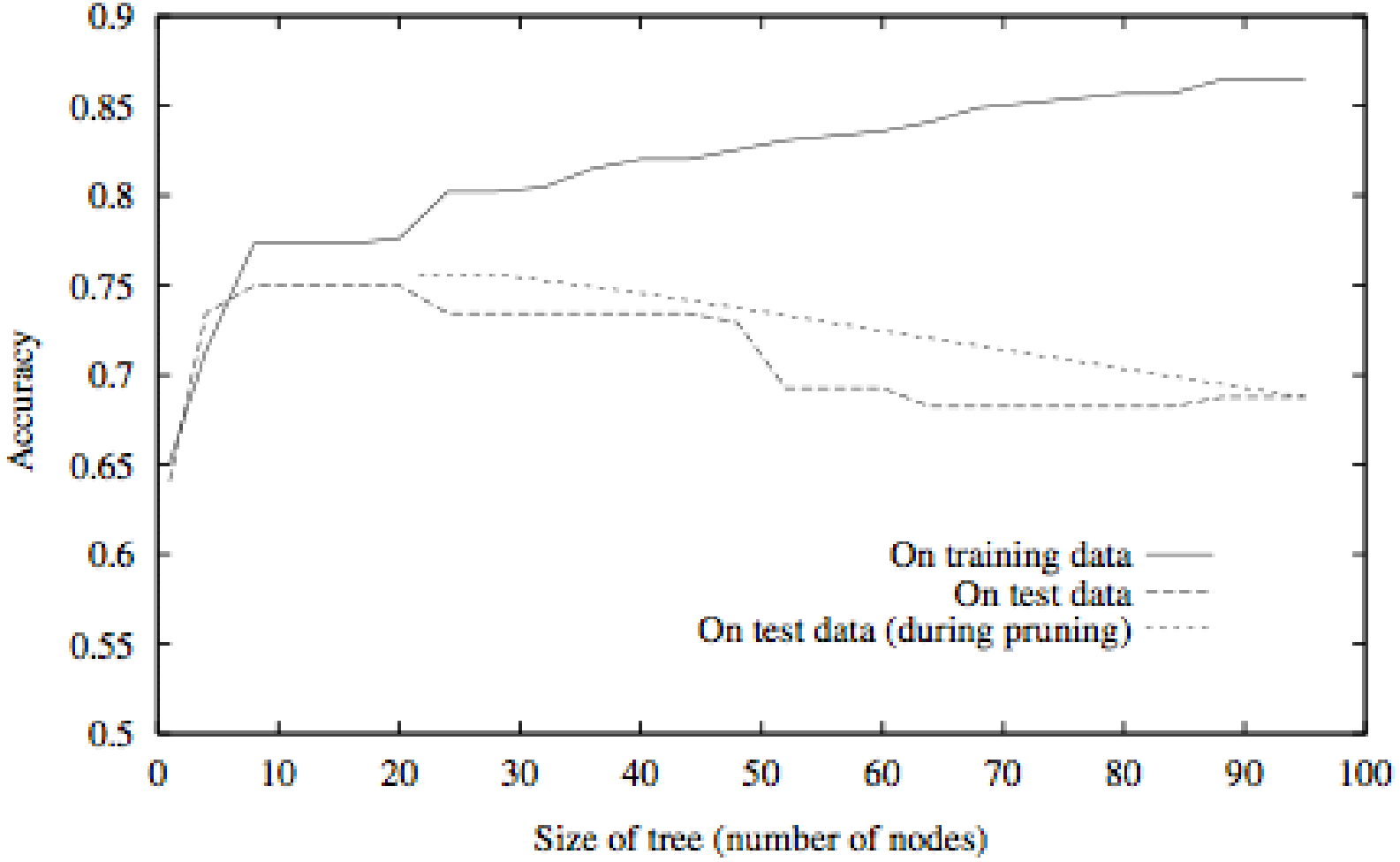
Avoid overfitting in Decision Trees

- Two strategies:
  1. Stop growing the tree earlier, before perfect classification
  2. Allow the tree to *overfit* the data, and then *post-prune* the tree
- Training and validation set
  - split the training in two parts (training and validation) and use validation to assess the utility of *post-pruning*
    - *Reduced error pruning*
    - *Rule pruning*

Reduced-error pruning (Quinlan 1987)

- Each node is a candidate for pruning
- *Pruning* consists in removing a subtree rooted in a node: the node becomes a leaf and is assigned the most common classification
- Nodes are removed only if the resulting tree performs no worse on the validation set.
- Nodes are pruned iteratively: at each iteration the node whose removal most increases accuracy on the validation set is pruned.
- Pruning stops when no pruning increases accuracy

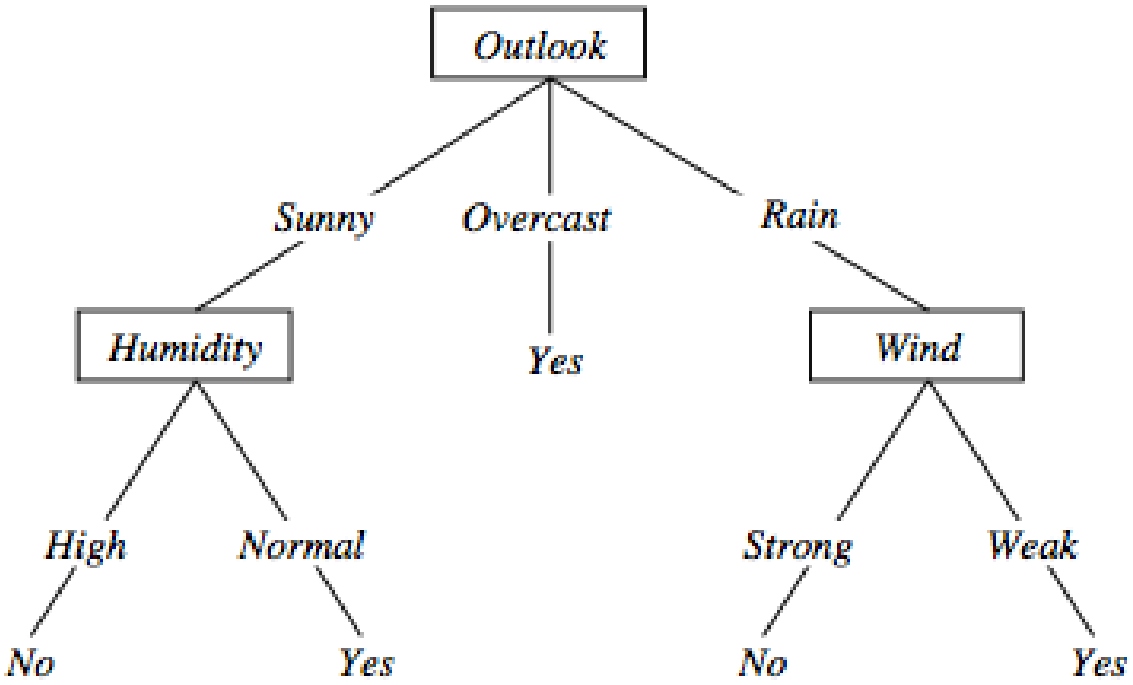
Effect of reduced error pruning



Rule post-pruning

1. Create the decision tree from the training set
2. Convert the tree into an equivalent set of rules
   - Each path corresponds to a rule
   - Each node along a path corresponds to a pre-condition
   - Each leaf classification to the post-condition
3. Prune (generalize) each rule by removing those preconditions whose removal improves accuracy …
   - … over validation set
   - … over training with a pessimistic, statistically inspired, measure
4. Sort the rules in estimated order of accuracy, and consider them in sequence when classifying new instances

Converting to rules



$$(Outlook{=}Sunny) \wedge (Humidity{=}High) \Rightarrow (PlayTennis{=}No)$$

Why converting to rules?

- Each distinct path produces a different rule: a condition removal may be based on a local (contextual) criterion. Node pruning is global and affects all the rules
- In rule form, tests are not ordered and there is no book-keeping involved when conditions (nodes) are removed
- Converting to rules improves readability for humans

Dealing with continuous-valued attributes

- So far discrete values for attributes and for outcome.
- Given a continuous-valued attribute A, dynamically create a new attribute A_c

  A_c = True *if* $A < c$, False *otherwise*
- How to determine threshold value *c* ?
- Example. *Temperature* in the *PlayTennis* example
  - Sort the examples according to *Temperature*

Temperature	40	48	|	60	72	80	|	90
PlayTennis	No	No	*54*	Yes	Yes	Yes	*85*	No

  - Determine candidate thresholds by averaging consecutive values where there is a change in classification: (48+60)/2=54 and (80+90)/2=85
  - Evaluate candidate thresholds (attributes) according to information gain. The best is $Temperature_{>54}$.The new attribute competes with the other ones

Problems with *information gain*

- Natural bias of information gain: it favours attributes with many possible values.
- Consider the attribute *Date* in the *PlayTennis* example.
  - *Date* would have the highest information gain since it perfectly separates the training data.
  - It would be selected at the root resulting in a very broad tree
  - Very good on the training, this tree would perform poorly in predicting unknown instances. Overfitting.
- The problem is that the partition is too specific, too many small classes are generated.
- We need to look at alternative measures …

An alternative measure: *gain ratio*

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

- S_i are the sets obtained by partitioning on value i of A
- *SplitInformation* measures the entropy of *S* with respect to the values of *A*. The more uniformly dispersed the data the higher it is.

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

- *GainRatio* penalizes attributes that split examples in many small classes such as *Date*. Let $|S|=n$, *Date* splits examples in n classes
  - $SplitInformation(S, Date) = -[(1/n \log_2 1/n) + \ldots + (1/n \log_2 1/n)] = -\log_2 1/n = \log_2 n$
- Compare with A, which splits data in two even classes:
  - $SplitInformation(S, A) = -[(1/2 \log_2 1/2) + (1/2 \log_2 1/2)] = -[-1/2 - 1/2] = 1$