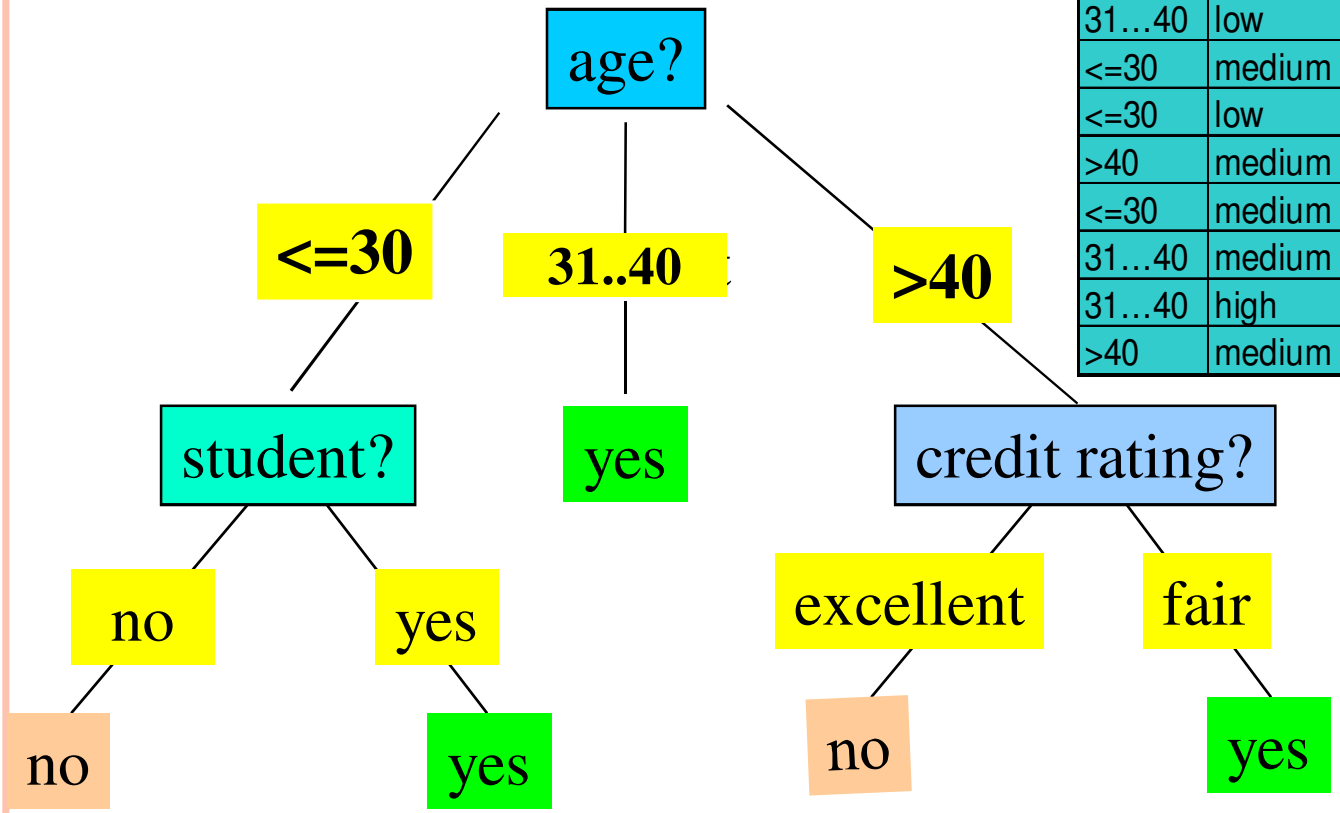
ISSUES IN DECISION TREE LEARNING

- Handling Continuous Attributes
- Other attribute selection measures
- Over fitting - Pruning
- Handling of missing values
- Incremental Induction of Decision Tree

DECISION TREE INDUCTION: AN EXAMPLE

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no



Continuous value attributes

- Two Methods for Handling Continuous Attributes
  - Discretization
    - Break real-valued attributes into ranges *in advance*
    - e.g., {*high* ≡ *Temp* > 35° C, *med* ≡ 10° C < *Temp* ≤ 35° C, *low* ≡ *Temp* ≤ 10° C}
  - Using thresholds for splitting nodes
    - e.g., $A \leq a$ produces subsets $A \leq a$ and $A > a$
    - *Information gain is calculated the same way* as for discrete splits

Computing Information-Gain for Continuous-Valued Attributes

- Let attribute A be a continuous-valued attribute
- Must determine the *best split point* for A
  - Sort the value A in increasing order
  - Typically, the midpoint between each pair of adjacent values is considered as a possible *split point*

    $(a_i+a_{i+1})/2$ is the midpoint between the values of a_i and a_{i+1}
  - The point with the *minimum expected information requirement* for A is selected as the split-point for A
- Split:
  - D1 is the set of tuples in D satisfying A ≤ split-point, and D2 is the set of tuples in D satisfying A > split-point

- **Example**

-	*A* ≡ *Length*:	10	15	21	28	32	40	50
-	Class:	-	+	+	-	+	+	-

- Check thresholds: *Length* ≤ 12.5? ≤ 24.5? ≤ 30? ≤ 45?

Gain Ratio for Attribute Selection

- Information gain measure is biased towards attributes with a large number of values
- C4.5 (a successor of ID3) uses gain ratio to overcome the problem (normalization to information gain)

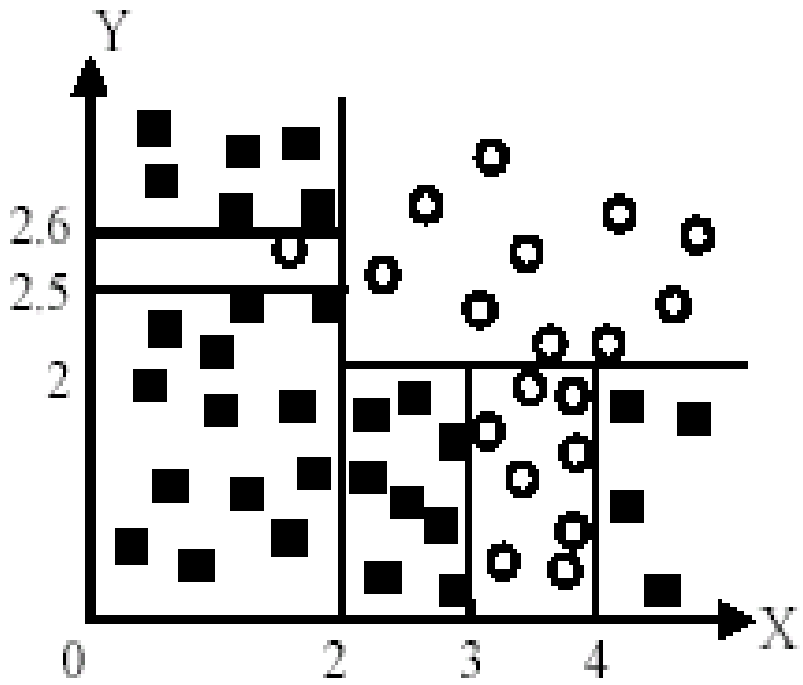
$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2\left(\frac{|D_j|}{|D|}\right)$$

  - GainRatio(A) = Gain(A)/SplitInfo(A)
- Ex.

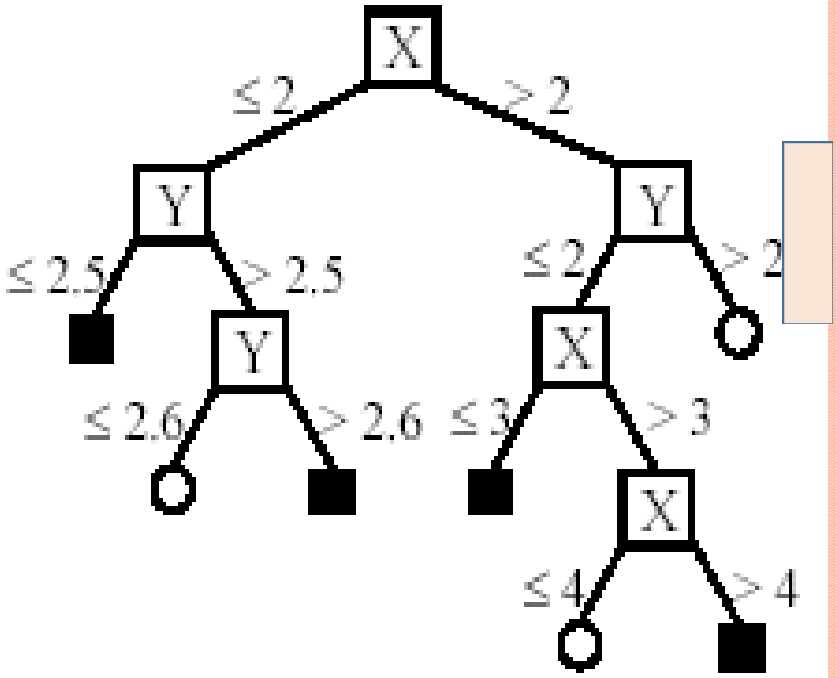
$$SplitInfo_{income}(D) = -\frac{4}{14} \times \log_2\left(\frac{4}{14}\right) - \frac{6}{14} \times \log_2\left(\frac{6}{14}\right) - \frac{4}{14} \times \log_2\left(\frac{4}{14}\right) = 1.557$$

  - gain_ratio(income) = 0.029/1.557 = 0.019
- The attribute with the maximum gain ratio is selected as the splitting attribute

An example in a continuous space



(A) A partition of the data space



(B). The decision tree

GINI INDEX (CART, IBM INTELLIGENTMINER)

- If a data set D contains examples from n classes, gini index, $gini(D)$ is defined as

$$gini(D) = 1 - \sum_{j=1}^{n} p_j^2$$

where p_j is the relative frequency of class j in D

- If a data set D is split on A into two subsets D_1 and D_2, the *gini* index $gini(D)$ is defined as
- Reduction in Impurity:

$$gini_A(D) = \frac{|D_1|}{|D|} gini(D_1) + \frac{|D_2|}{|D|} gini(D_2)$$

$$\Delta gini(A) = gini(D) - gini_A(D)$$

- The attribute provides the smallest $gini_{split}(D)$ (or the largest reduction in impurity) is chosen to split the node (*need to enumerate all the possible splitting points for each attribute*)

COMPUTATION OF GINI INDEX

- Ex. D has 9 tuples in buys_computer = "yes" and 5 in "no"

$$gini(D) = 1 - \left(\frac{9}{14}\right)^2 - \left(\frac{5}{14}\right)^2 = 0.459$$

- Suppose the attribute income partitions D into 10 in D_1: {low, medium} and 4 in D_2

$$gini_{income \in \{low,medium\}}(D) = \left(\frac{10}{14}\right) Gini(D_1) + \left(\frac{4}{14}\right) Gini(D_1)$$

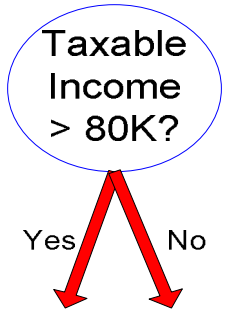
$$= \frac{10}{14}\left(1 - \left(\frac{7}{10}\right)^2 - \left(\frac{3}{10}\right)^2\right) + \frac{4}{14}\left(1 - \left(\frac{2}{4}\right)^2 - \left(\frac{2}{4}\right)^2\right)$$
$$= 0.443$$
$$= Gini_{income \in \{high\}}(D).$$

$Gini_{\{low,high\}}$ is 0.458; $Gini_{\{medium,high\}}$ is 0.450. Thus, split on the {low,medium} (and {high}) since it has the lowest Gini index

Continuous Attributes: Computing Gini Index

- Use Binary Decisions based on one value
- Several Choices for the splitting value
  - Number of possible splitting values = Number of distinct values
- Each splitting value has a count matrix associated with it
  - Class counts in each of the partitions, $A < v$ and $A \geq v$
- Simple method to choose best v
  - For each v, scan the database to gather count matrix and compute its Gini index
  - Computationally Inefficient! Repetition of work.

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



CONTINUOUS ATTRIBUTES: COMPUTING GINI INDEX...

- For efficient computation: for each attribute,
  - Sort the attribute on values
  - Linearly scan these values, each time updating the count matrix and computing gini index
  - Choose the split position that has the least gini index

Sorted Values →

Split Positions →

Cheat	No	No	No	Yes	Yes	Yes	No	No	No	No
Taxable Income										
	60	70	75	85	90	95	100	120	125	220
	55	65	72	80	87	92	97	110	122	172	230
	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >
Yes	0 3	0 3	0 3	0 3	1 2	2 1	3 0	3 0	3 0	3 0	3 0
No	0 7	1 6	2 5	3 4	3 4	3 4	3 4	4 3	5 2	6 1	7 0
Gini	0.420	0.400	0.375	0.343	0.417	0.400	***0.300***	0.343	0.375	0.400	0.420

Comparing Attribute Selection Measures

- The three measures, in general, return good results but
  - **Information gain**:
    - biased towards multivalued attributes
  - **Gain ratio**:
    - tends to prefer unbalanced splits in which one partition is much smaller than the others
  - **Gini index**:
    - biased to multivalued attributes
    - has difficulty when # of classes is large
    - tends to favor tests that result in equal-sized partitions and purity in both partitions

Other Attribute Selection Measures

- CHAID: a popular decision tree algorithm, measure based on χ^2 test for independence
- C-SEP: performs better than info. gain and gini index in certain cases
- G-statistic: has a close approximation to χ^2 distribution
- MDL (Minimal Description Length) principle (i.e., the simplest solution is preferred):
  - The best tree as the one that requires the fewest # of bits to both (1) encode the tree, and (2) encode the exceptions to the tree
- Multivariate splits (partition based on multiple variable combinations)
  - CART: finds multivariate splits based on a linear comb. of attrs.
- Which attribute selection measure is the best?
  - Most give good results, none is significantly superior than others

OVERFITTING

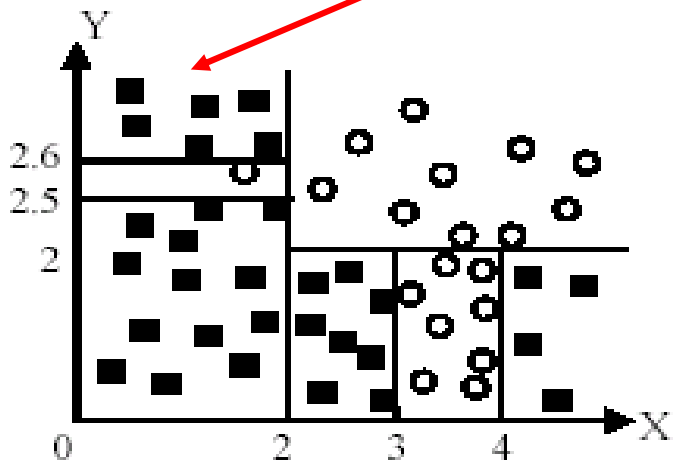
- Hypothesis h overfits training data set D if $\exists$ an alternative hypothesis h' such that $error_D(h) < error_D(h')$ but $error_{test}(h) > error_{test}(h')$
- Causes: sample too small (decisions based on too little data); noise; coincidence

Overfitting and Tree Pruning

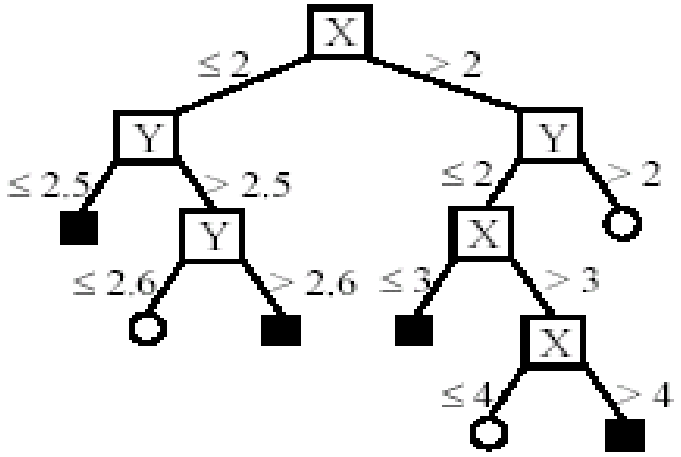
- Overfitting: An induced tree may overfit the training data
  - Too many branches, some may reflect anomalies due to noise or outliers
  - Poor accuracy for unseen samples
- Two approaches to avoid overfitting
  - Prepruning: *Halt tree construction early* - do not split a node if this would result in the goodness measure falling below a threshold
    - Difficult to choose an appropriate threshold
  - Postpruning: *Remove branches* from a "fully grown" tree—get a sequence of progressively pruned trees
    - Use a set of data different from the training data to decide which is the "best pruned tree"

Likely to overfit the data

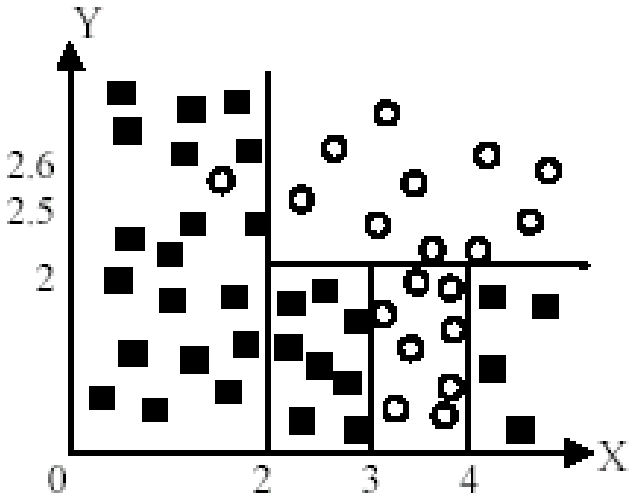
AN EXAMPLE

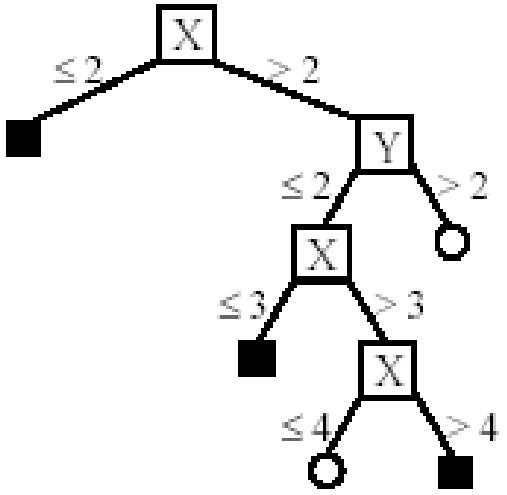


(A) A partition of the data space



(B). The decision tree





TYPES OF PRUNING

- Expected error pruning
- Reduced error pruning
- Rule post pruning

EXPECTED ERROR PRUNING

Algorithm:

- Approximate expected error assuming that we prune at a particular node.
- Approximate backed-up error from children assuming we did not prune.
- If expected error is less than backed-up error, prune.

(Static) Expected Error:

- If we prune a node, it becomes a leaf labelled, C.
- What will be the expected classification error at this leaf?

- $E(S) = (N - n + k - 1)/(N + k)$

- (This is called the *Laplace* error estimate - it is based on the assumption that the distribution of probabilities that examples will belong to different classes is uniform.)

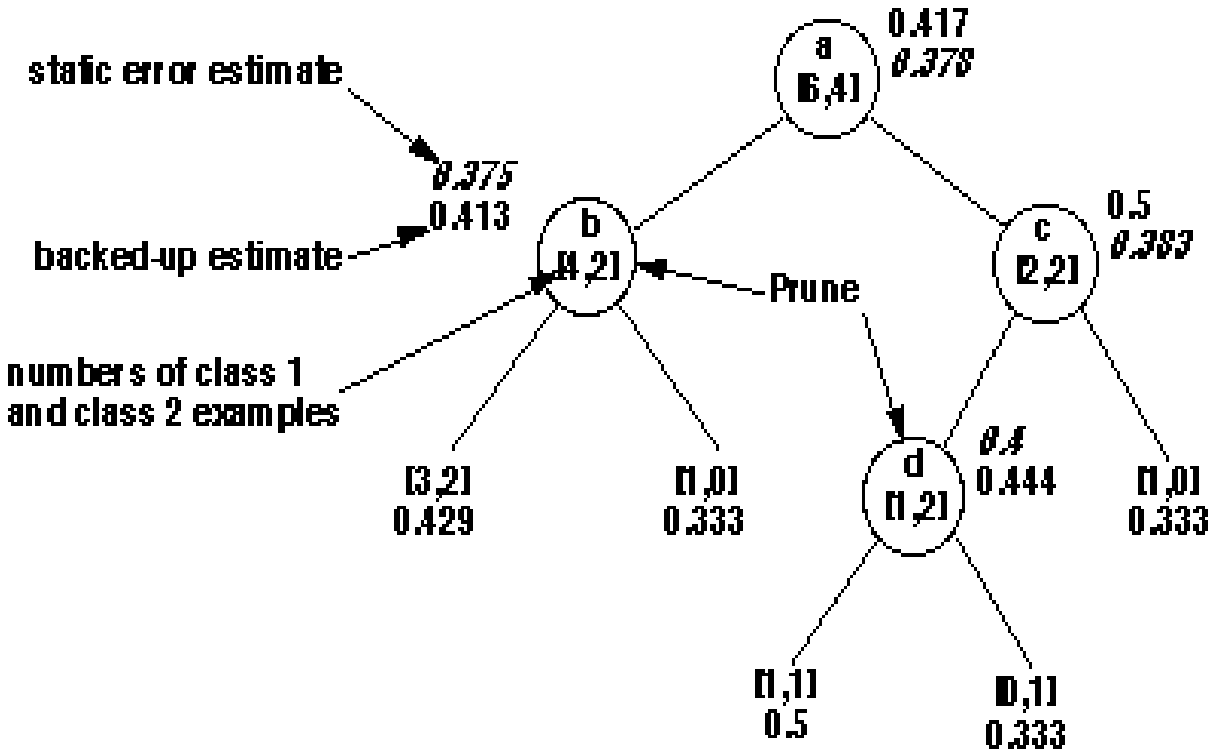
- S - is the set of examples in a node
- k - is the number of classes
- N - examples in S
- C - is the majority class in S
- n - out of N examples in S belong to C

Backed-up Error:

- For a non-leaf node
- Let children of *Node* be *Node1*, *Node2*, etc
- $BackedUpError(Node) = \Sigma_i P_i \times Error(Node_i)$
- Probabilities can be estimated by relative frequencies of attribute values in sets of examples that fall into child nodes.

- $Error(Node) = \min(E(Node), BackedUpError(Node))$

EXPECTED ERROR PRUNING - EXAMPLE



- Error Calculation for Pruning Example
- Left child of *b* has class frequencies [3, 2]
- $E = (N - n + k - 1)/(N + k)$

  $= (5 - 3 + 2 - 1)/(5 + 2) = 0.429$

- Right child has error of 0.333, calculated in the same way

- Static error estimate $E(b)$ is 0.375, again calculated using the Laplace error estimate formula, with N=6, n=4, and k=2.

- Backed-up error is:
- $BackedUpError(b) = (5/6)\times 0.429 + (1/6)\times 0.333 = 0.413$
- (5/6 and 1/6 because there are 4+2=6 examples handled by node *b*, of which 3+2=5 go to the left subtree and 1 to the right subtree.

- Since backed-up estimate of 0.413 is greater than static estimate of 0.375, we prune the tree and use the static error of 0.375

Reduced Error Pruning

- **Algorithm *Reduced-Error-Pruning* (*D*)**
  - **Partition *D* into D_{train} (training / "growing"), $D_{validation}$ (validation / "pruning")**
  - **Build complete tree *T* using *ID3* on D_{train}**
  - **UNTIL accuracy on $D_{validation}$ decreases DO**

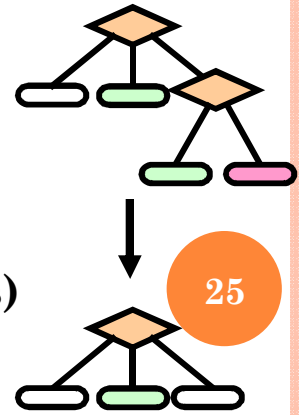
    FOR each non-leaf node *candidate* in *T*

    $Temp[candidate] \leftarrow Prune(T, candidate)$

    $Accuracy[candidate] \leftarrow Test(Temp[candidate], D_{validation})$

    $T \leftarrow T' \in Temp$ with best value of *Accuracy* (best increase; *greedy*)
  - **RETURN (pruned) *T***

- **Function *Prune*(*T, node*)**
  - **Remove the subtree rooted at *node***
  - **Make *node* a leaf (with majority label of associated examples)**



REDUCED-ERROR PRUNING

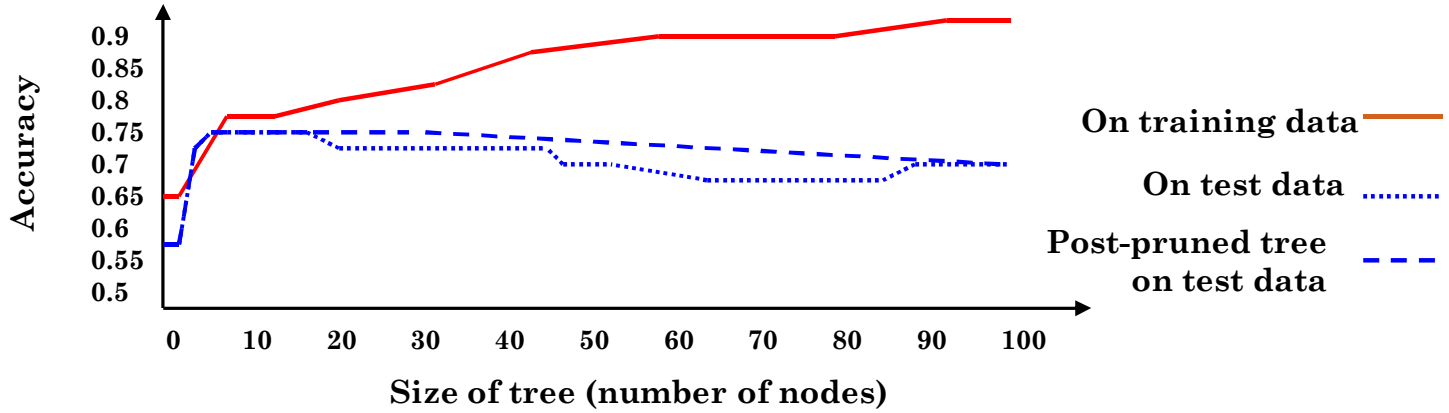
- Split data into training and validation set
- Do until further pruning is harmful:

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

  2. Greedily remove the one that most improves validation set accuracy

- It produces smallest version of most accurate subtree

- **Reduction of Test Error by Reduced-Error Pruning**



RULE POST PRUNNING

- **Infer *T* from *D* (using *ID3*) - grow until *D* is fit as well as possible (allow overfitting)**
- **Convert *T* into equivalent set of rules (one for each root-to-leaf path)**
- **Prune (generalize) each rule *independently* by deleting any preconditions whose deletion improves its estimated accuracy**
- **Sort the pruned rules**
  - **Sort by their estimated accuracy**
  - **Apply them in sequence on D_{test}**

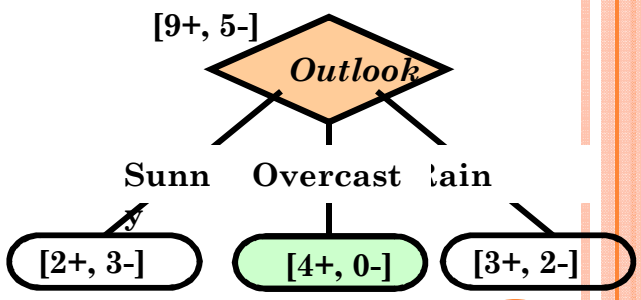
MISSING ATTRIBUTE VALUES

- Assign the most common value of the attribute
- Assign probability to each of the possible values

Missing values

- **Problem: What If Some Examples Missing Values of *A*?**
  - Often, values not available for all attributes during training or testing
  - Example: medical diagnosis
    - *<Fever = true, Blood-Pressure = normal, …, Blood-Test = ?, …>*
    - Sometimes values truly unknown, sometimes low priority (or cost too high)
  - Missing values in learning versus classification
    - Training: evaluate *Gain* (*D*, *A*) where for some $x \in D$, a value for *A* is not given
    - Testing: classify a new example *x* without knowing the value of *A*
- Solutions: Incorporating a *Guess* into Calculation of *Gain*(*D*, *A*)

Day	Outlook	Temperature	Humidity	Wind	*PlayTennis*?
1	Sunny	Hot	High	Light	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Light	Yes
4	Rain	Mild	High	Light	Yes
5	Rain	Cool	Normal	Light	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	???	Light	No
9	Sunny	Cool	Normal	Light	Yes
10	Rain	Mild	Normal	Light	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Light	Yes
14	Rain	Mild	High	Strong	No



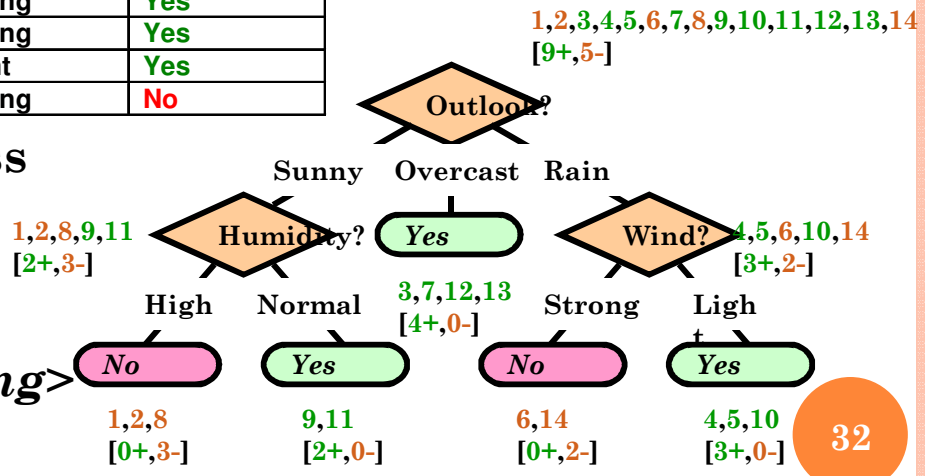
- **Use Training Example Anyway, Sort Through Tree**
  - **For each attribute being considered, guess its value in examples where unknown**
  - **Base the guess upon examples *at current node* where value is known**
- **Guess the Most Likely Value of *x.A***
  - **Variation 1: if node *n* tests *A*, assign most common value of *A* among other examples routed to node *n***
  - **Variation 2 : if node *n* tests *A*, assign most common value of *A* among other examples routed to node *n that have the same class label* as *x***
- **Distribute the Guess Proportionately**
  - **Hedge the bet: distribute the guess according to distribution of values**
  - **Assign probability p_i to each possible value v_i of *x.A***
    - **Assign fraction p_i of *x* to each descendant in the tree**
    - **Use this in calculating *Gain* (*D*, *A*) or *Cost-Normalized-Gain* (*D*, *A*)**
- **In All Approaches, Classify New Examples in Same Fashion**

Example

- **Guess the Most Likely Value of *x.A***
  - **Variation 1: *Humidity* = *High* or *Normal* (*High*: *Gain* = 0.97, *Normal*: < 0.97)**
  - **Variation 2: *Humidity* = *High* (all *No* cases are *High*)**

Day	Outlook	Temperature	Humidity	Wind	*PlayTennis*?
1	Sunny	Hot	High	Light	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Light	Yes
4	Rain	Mild	High	Light	Yes
5	Rain	Cool	Normal	Light	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	???	Light	No
9	Sunny	Cool	Normal	Light	Yes
10	Rain	Mild	Normal	Light	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Light	Yes
14	Rain	Mild	High	Strong	No

- **Probabilistically Weighted Guess**
  - **Guess *0.5 High, 0.5 Normal***
  - ***Gain* < 0.97**
- **Test Case: <*?*, *Hot*, *Normal*, *Strong*>**
  - **1/3 *Yes* + 1/3 *Yes* + 1/3 *No* = *Yes***



HANDLING MISSING ATTRIBUTE VALUES

- Missing values affect decision tree construction in three different ways:
  - Affects how impurity measures are computed
  - Affects how to distribute instance with missing value to child nodes
  - Affects how a test instance with missing value is classified

Computing Impurity Measure

Tid	Refund	Marital Status	Taxable Income	Class
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	?	Single	90K	Yes

Missing value

Before Splitting:

Entropy(Parent)
= -0.3 log(0.3)-(0.7)log(0.7) = 0.8813

	Class = Yes	Class = No
Refund=Yes	0	3
Refund=No	2	4
Refund=?	1	0

Split on Refund:

Entropy(Refund=Yes) = 0

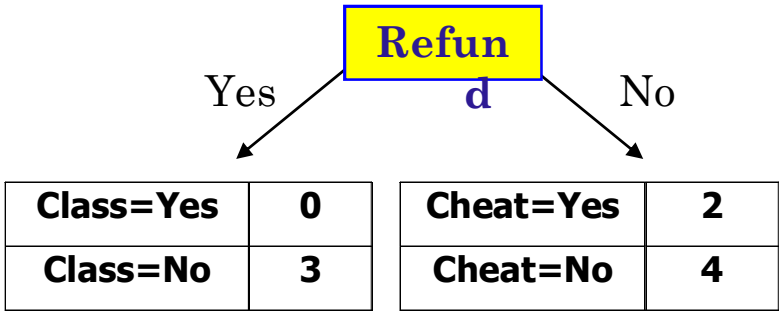
Entropy(Refund=No)
= -(2/6)log(2/6) – (4/6)log(4/6) = 0.9183

Entropy(Children)
= 0.3 (0) + 0.6 (0.9183) = 0.551

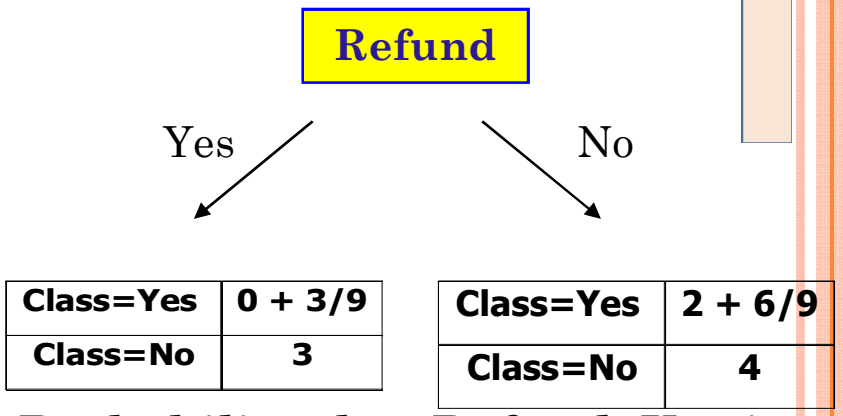
Gain = 0.9 × (0.8813 – 0.551) = 0.3303

DISTRIBUTE INSTANCES

Tid	Refund	Marital Status	Taxable Income	Class
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



Tid	Refund	Marital Status	Taxable Income	Class
10	?	Single	90K	Yes



Probability that Refund=Yes is 3/9

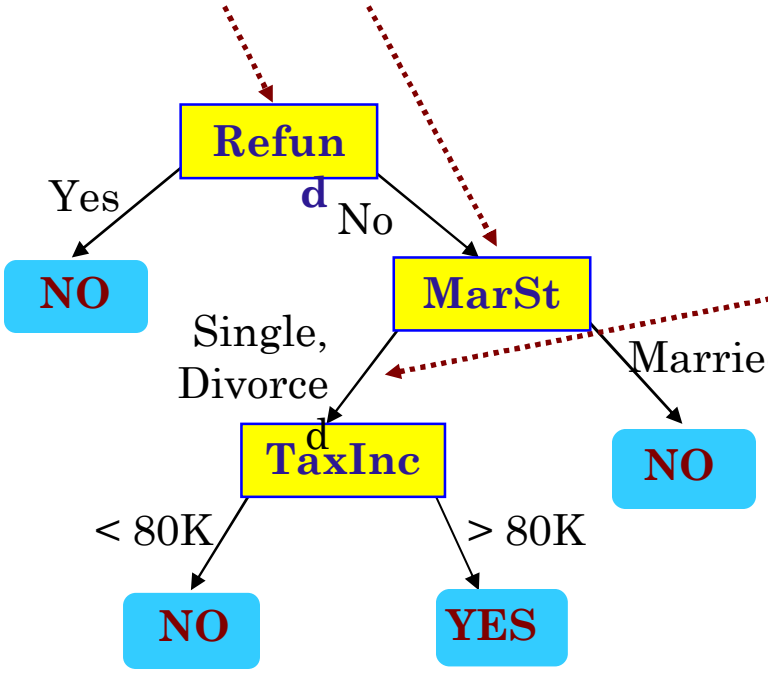
Probability that Refund=No is 6/9

Assign record to the left child with weight = 3/9 and to the right child with weight = 6/9

CLASSIFY INSTANCES

New record:

Tid	Refund	Marital Status	Taxable Income	Class
11	No	?	85K	?



	Married	Single	Divorced	Total
Class=No	3	1	0	4
Class=Yes	6/9	1	1	2.67
Total	3.67	2	1	6.67

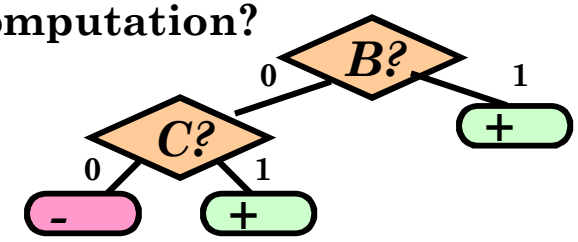
Probability that Marital Status
= Married is 3.67/6.67

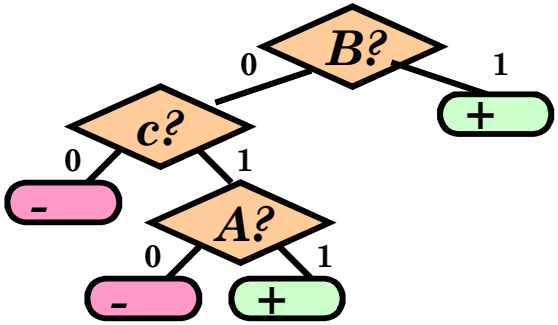
Probability that Marital Status ={Single,Divorced} is 3/6.67

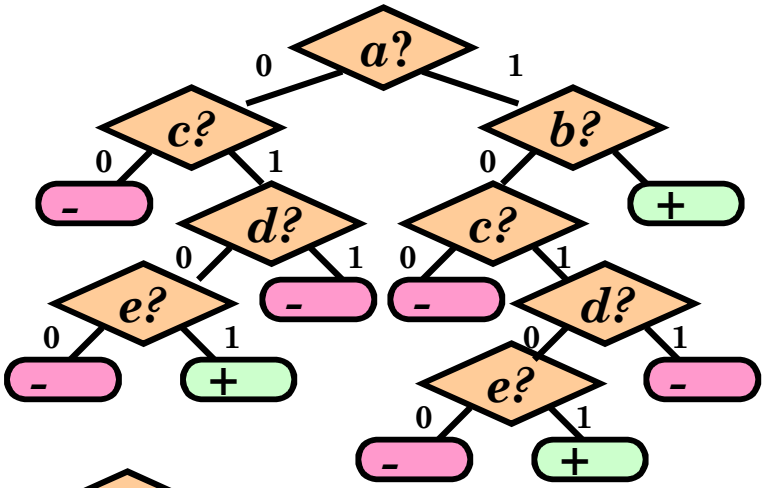
INCREMENTAL INDUCTION

- **Update an existing decision tree to account for new examples incrementally**
- **Consistency issues**
- **Minimality issues**

- **Synthesizing New Attributes**
  - Synthesize (create) a new attribute from the conjunction of the last two attributes before a + node
  - *aka* feature construction
- **Example**
  - $(a \wedge b) \vee (c \wedge \neg d \wedge e)$
  - $A = \neg d \wedge e$
  - $B = a \wedge b$
- **Repeated application**
  - $C = A \wedge c$
  - Correctness?
  - Computation?



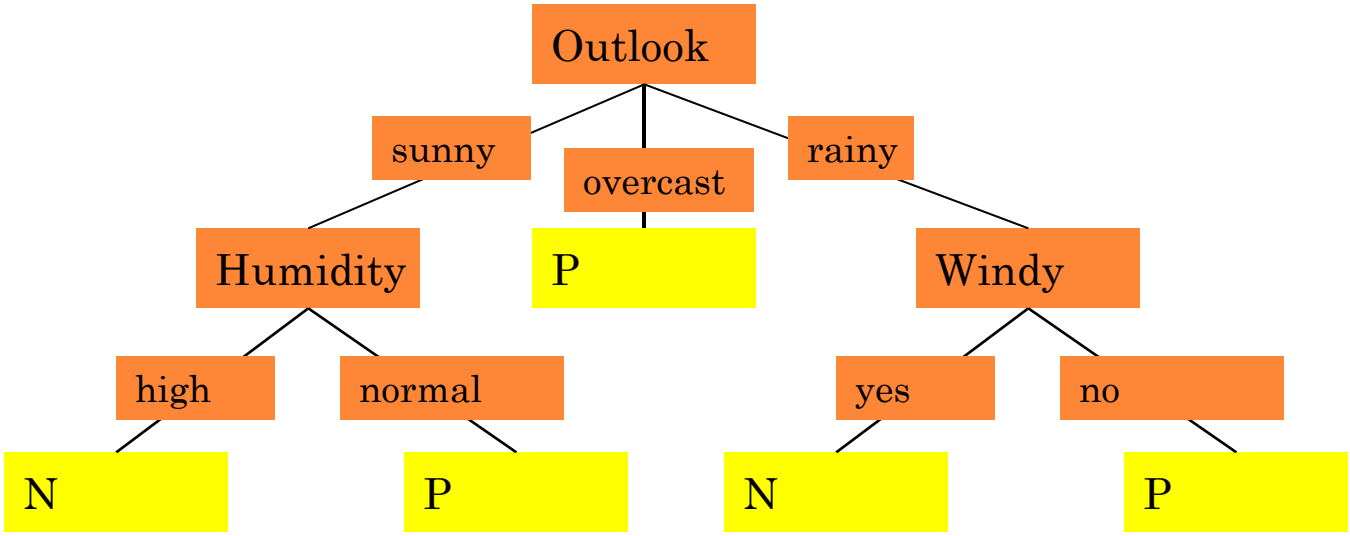




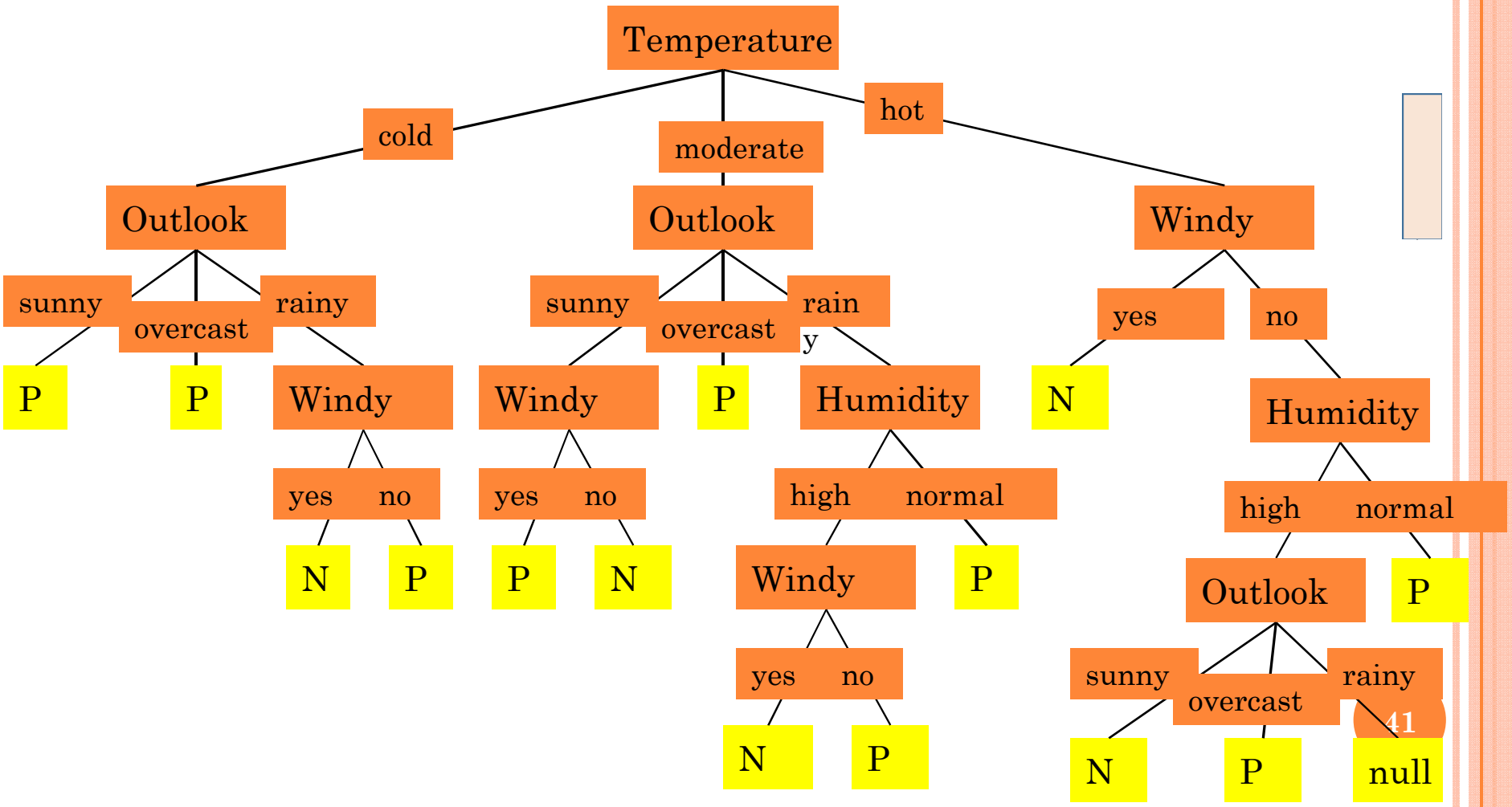
DECISION TREE LEARNING

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

SIMPLE TREE


Outlook
sunny
overcast
rainy
Humidity
P
Windy
high
normal
yes
no
N
P
N
P

COMPLICATED TREE



Decision Tree Learning: A Simple Example

- Let's start off by calculating the Entropy of the Training Set.
- E(S) = E([9+,5-]) = (-9/14 $\log_2$ 9/14) + (-5/14 $\log_2$ 5/14)
- = 0.94

Decision Tree Learning: A Simple Example

- Next we will need to calculate the information gain G(S,A) for each attribute A where A is taken from the set {Outlook, Temperature, Humidity, Wind}.

Decision Tree Learning: A Simple Example

- The information gain for Outlook is:
  - G(S,Outlook) = E(S) – [5/14 * E(Outlook=sunny) + 4/14 * E(Outlook = overcast) + 5/14 * E(Outlook=rain)]
  - G(S,Outlook) = E([9+,5-]) – [5/14*E(2+,3-) + 4/14*E([4+,0-]) + 5/14*E([3+,2-])]
  - G(S,Outlook) = 0.94 – [5/14*0.971 + 4/14*0.0 + 5/14*0.971]
  - **G(S,Outlook) = 0.246**

Decision Tree Learning: A Simple Example

- G(S,Temperature) = 0.94 – [4/14*E(Temperature=hot) + 6/14*E(Temperature=mild) + 4/14*E(Temperature=cool)]
- G(S,Temperature) = 0.94 – [4/14*E([2+,2-]) + 6/14*E([4+,2-]) + 4/14*E([3+,1-])]
- G(S,Temperature) = 0.94 – [4/14 + 6/14*0.918 + 4/14*0.811]
- **G(S,Temperature) = 0.029**

Decision Tree Learning: A Simple Example

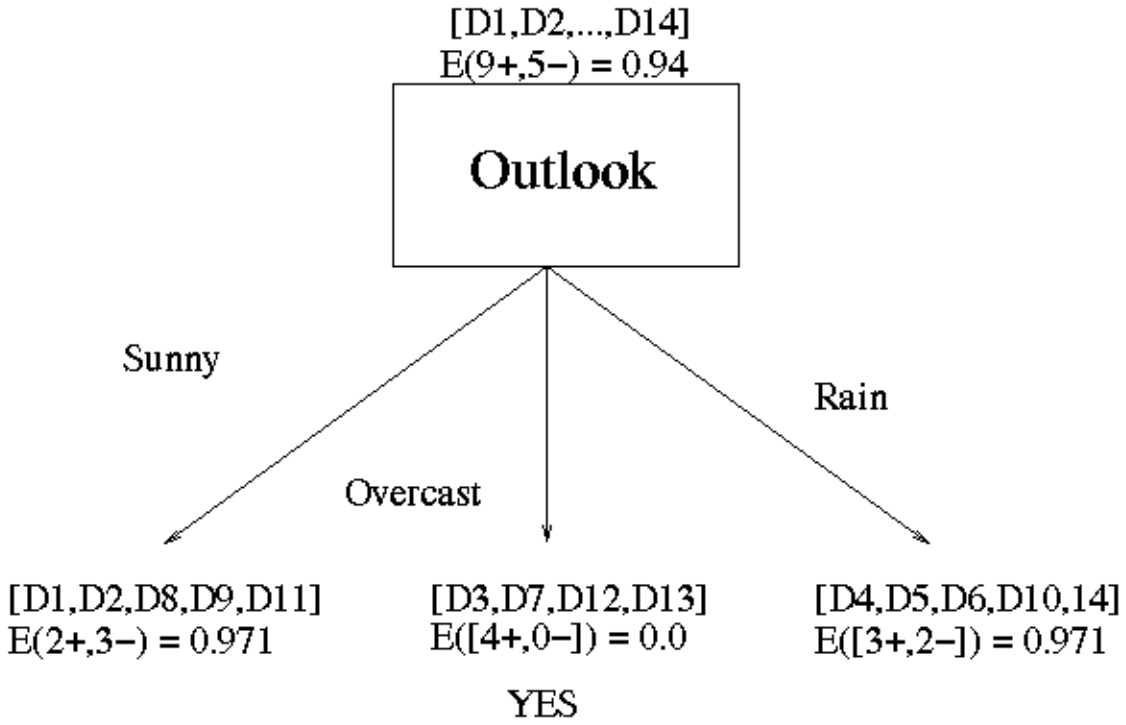
- G(S,Humidity) = 0.94 – [7/14*E(Humidity=high) + 7/14*E(Humidity=normal)]
- G(S,Humidity = 0.94 – [7/14*E([3+,4-]) + 7/14*E([6+,1-])]
- G(S,Humidity = 0.94 – [7/14*0.985 + 7/14*0.592]
- **G(S,Humidity) = 0.1515**

Decision Tree Learning: A Simple Example

- G(S,Wind) = 0.94 – [8/14*0.811 + 6/14*1.00]
- **G(S,Wind) = 0.048**

Decision Tree Learning: A Simple Example

- Outlook is our winner!



Decision Tree Learning: A Simple Example

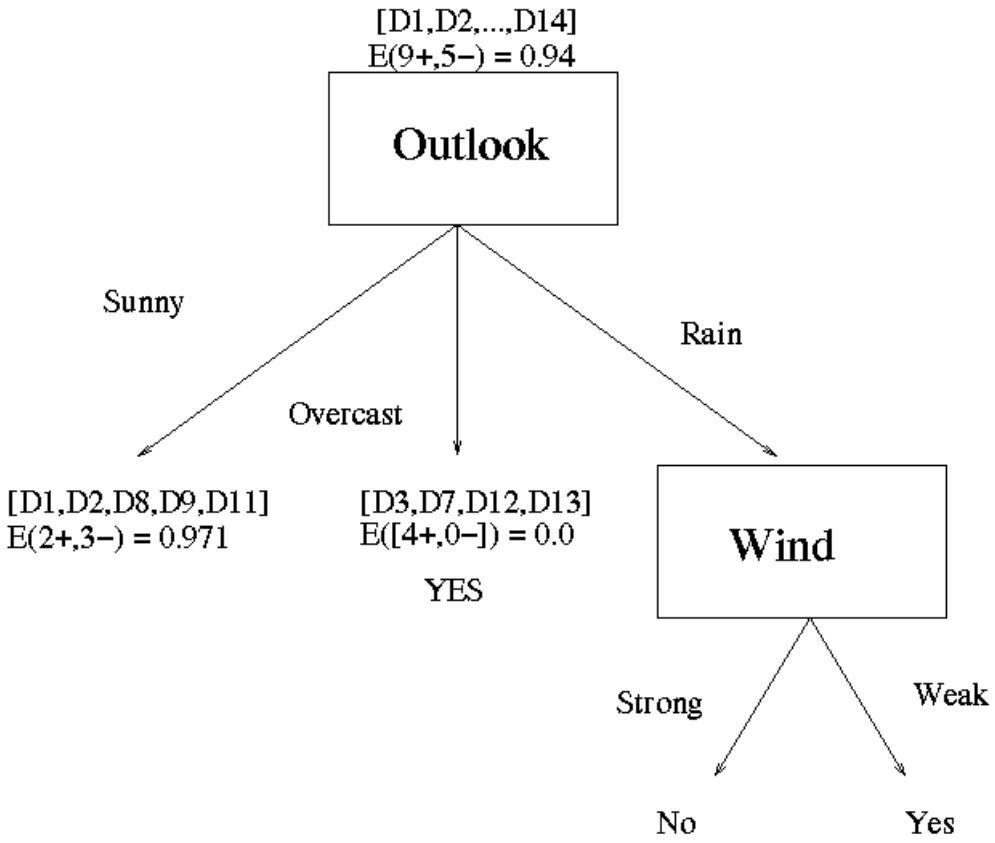
- Now that we have discovered the root of our decision tree we must now recursively find the nodes that should go below Sunny, Overcast, and Rain.

Decision Tree Learning: A Simple Example

- G(Outlook=Rain, Humidity) = 0.971 – [2/5*E(Outlook=Rain ^ Humidity=high) + 3/5*E(Outlook=Rain ^Humidity=normal]
- **G(Outlook=Rain, Humidity) = 0.02**
- G(Outlook=Rain,Wind) = 0.971- [3/5*0 + 2/5*0]
- **G(Outlook=Rain,Wind) = 0.971**

Decision Tree Learning: A Simple Example

- Now our decision tree looks like:



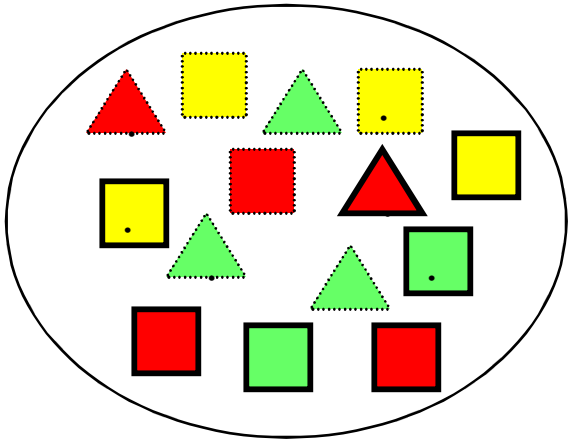
Triangles and Squares

#	Attribute			Shape
	Color	Outline	Dot	
1	green	dashed	no	triange
2	green	dashed	yes	triange
3	yellow	dashed	no	square
4	red	dashed	no	square
5	red	solid	no	square
6	red	solid	yes	triange
7	green	solid	no	square
8	green	dashed	no	triange
9	yellow	solid	yes	square
10	red	solid	no	square
11	green	solid	yes	square
12	yellow	dashed	yes	square
13	yellow	solid	no	square
14	red	dashed	yes	triange

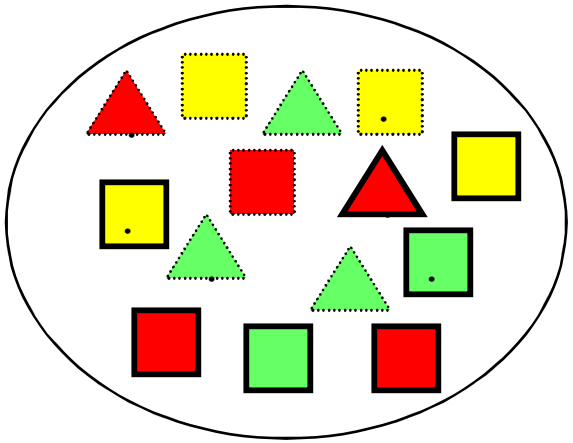
Triangles and Squares

#	Attribute			Shape
	Color	Outline	Dot	
1	green	dashed	no	triange
2	green	dashed	yes	triange
3	yellow	dashed	no	square
4	red	dashed	no	square
5	red	solid	no	square
6	red	solid	yes	triange
7	green	solid	no	square
8	green	dashed	no	triange
9	yellow	solid	yes	square
10	red	solid	no	square
11	green	solid	yes	square
12	yellow	dashed	yes	square
13	yellow	solid	no	square
14	red	dashed	yes	triange

Data Set:
A set of classified objects



ENTROPY



- 5 triangles
- 9 squares
- class probabilities

$$p(\square) = \frac{9}{14}$$

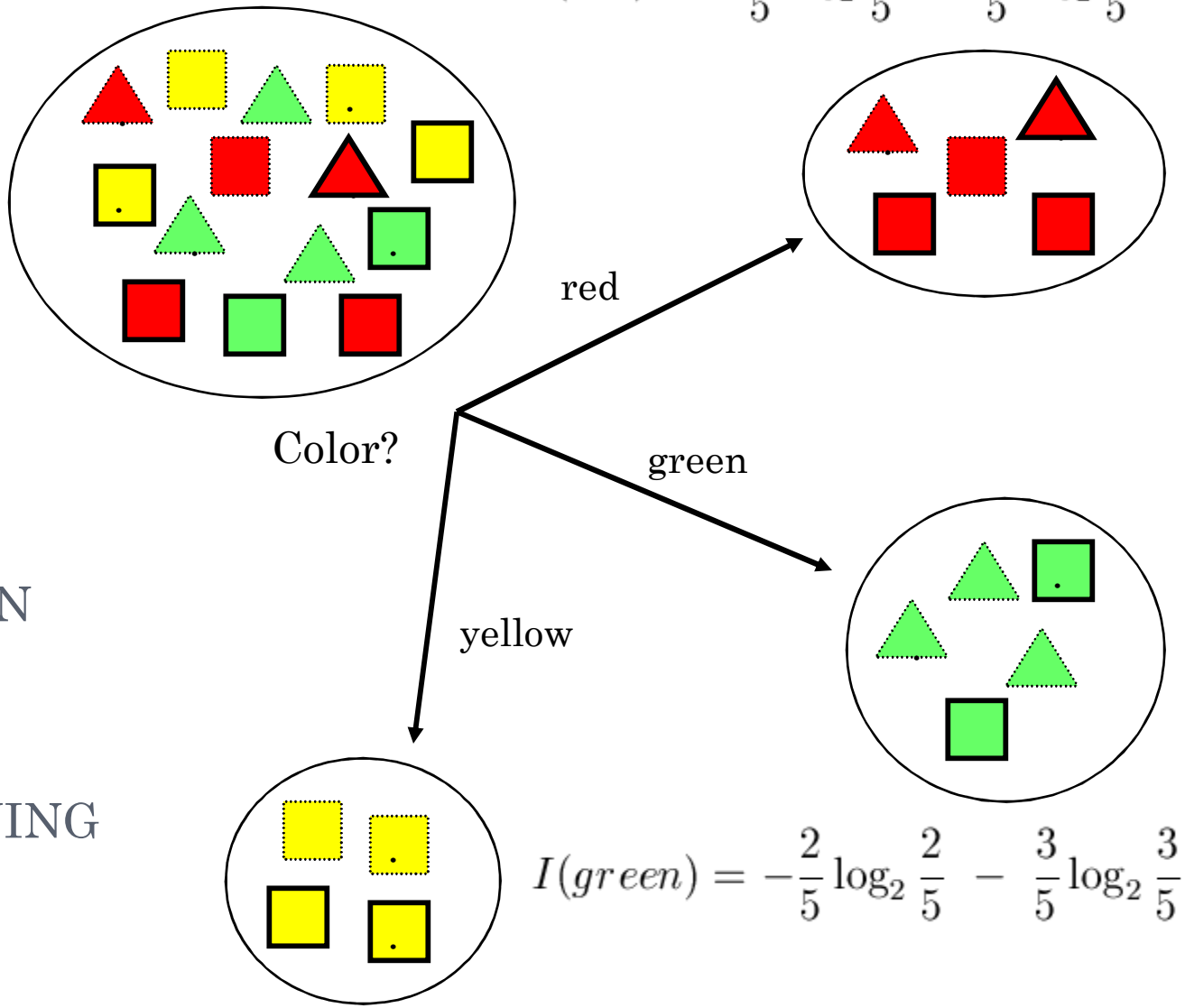
$$p(\triangle) = \frac{5}{14}$$

- entropy

$$I = -\frac{9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14} = 0.940 \text{ bits}$$

ENTROPY REDUCTION BY DATA SET PARTITIONING

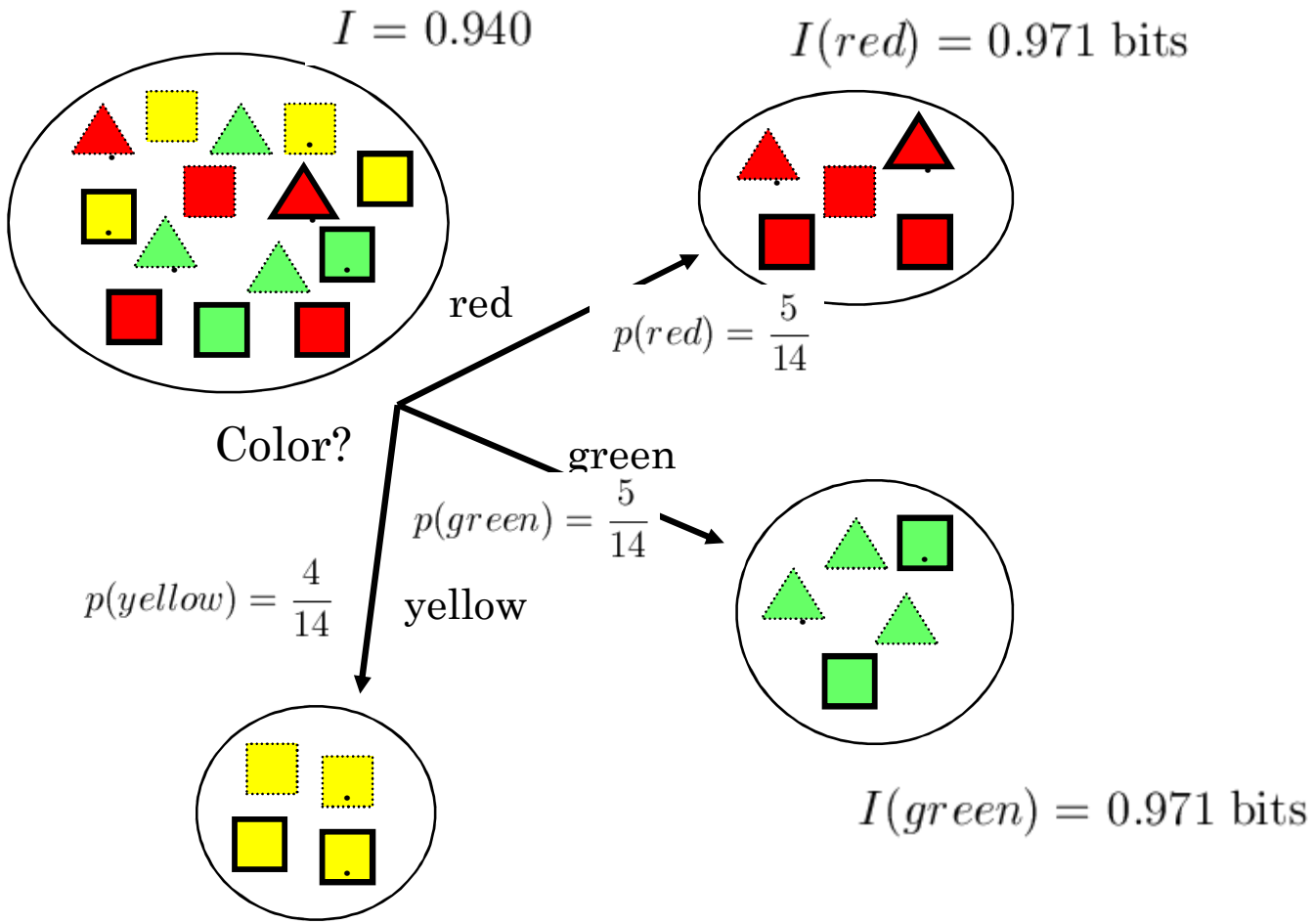
$$I(red) = -\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5} = 0.971 \text{ bits}$$



$$I(green) = -\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5} = 0.971 \text{ bits}$$

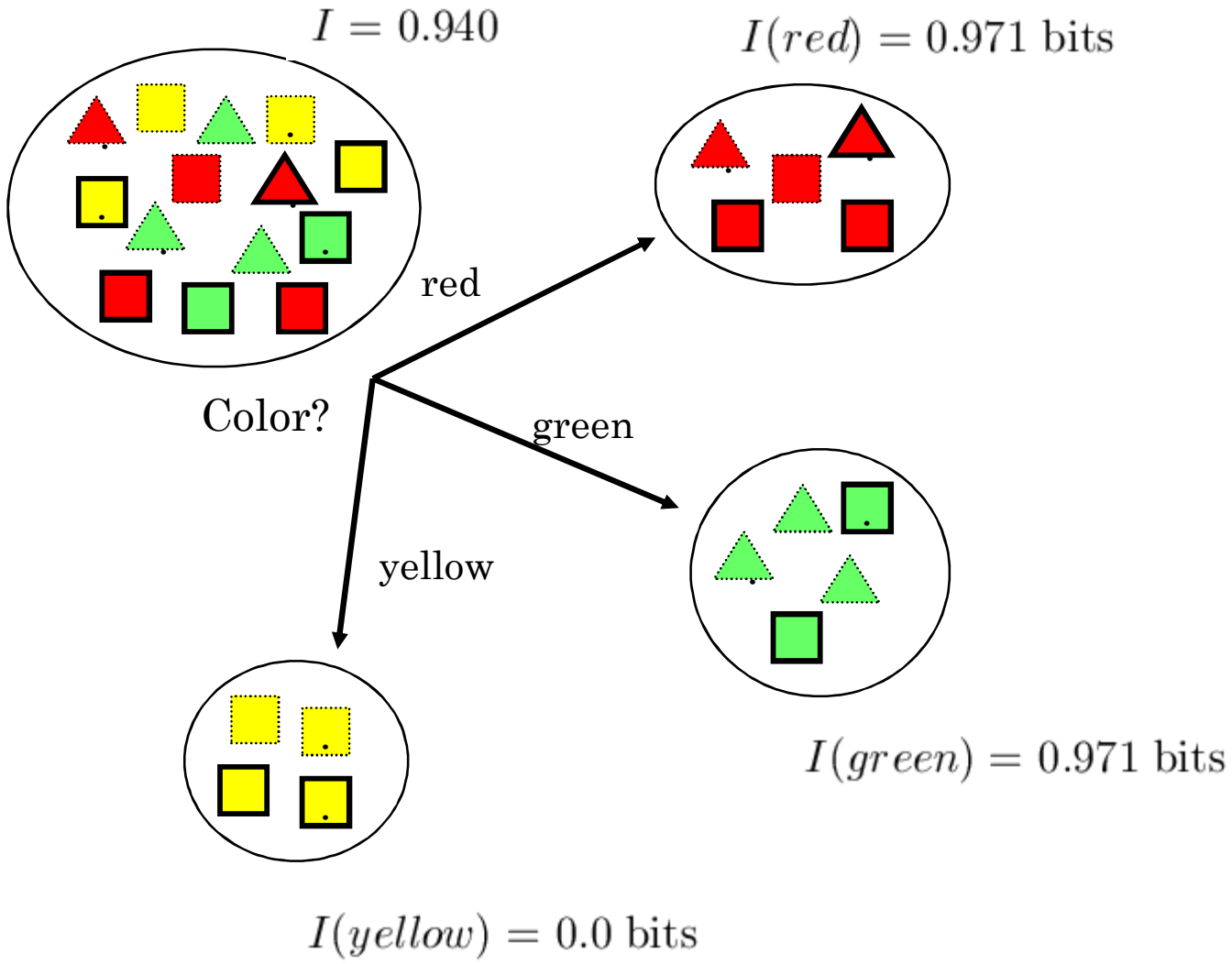
$$I(yellow) = -\frac{4}{4}\log_2\frac{4}{4} - \frac{0}{4}\log_2\frac{0}{4} = 0.0 \text{ bits}$$

ENTROPIJA VREDNOSTI ATRIBUTA



$$I_{res}(\text{Color}) = \sum p(v)I(v) = \frac{5}{14}0.971 + \frac{5}{14}0.971 + \frac{4}{14}0.0 = 0.694 \; bits$$

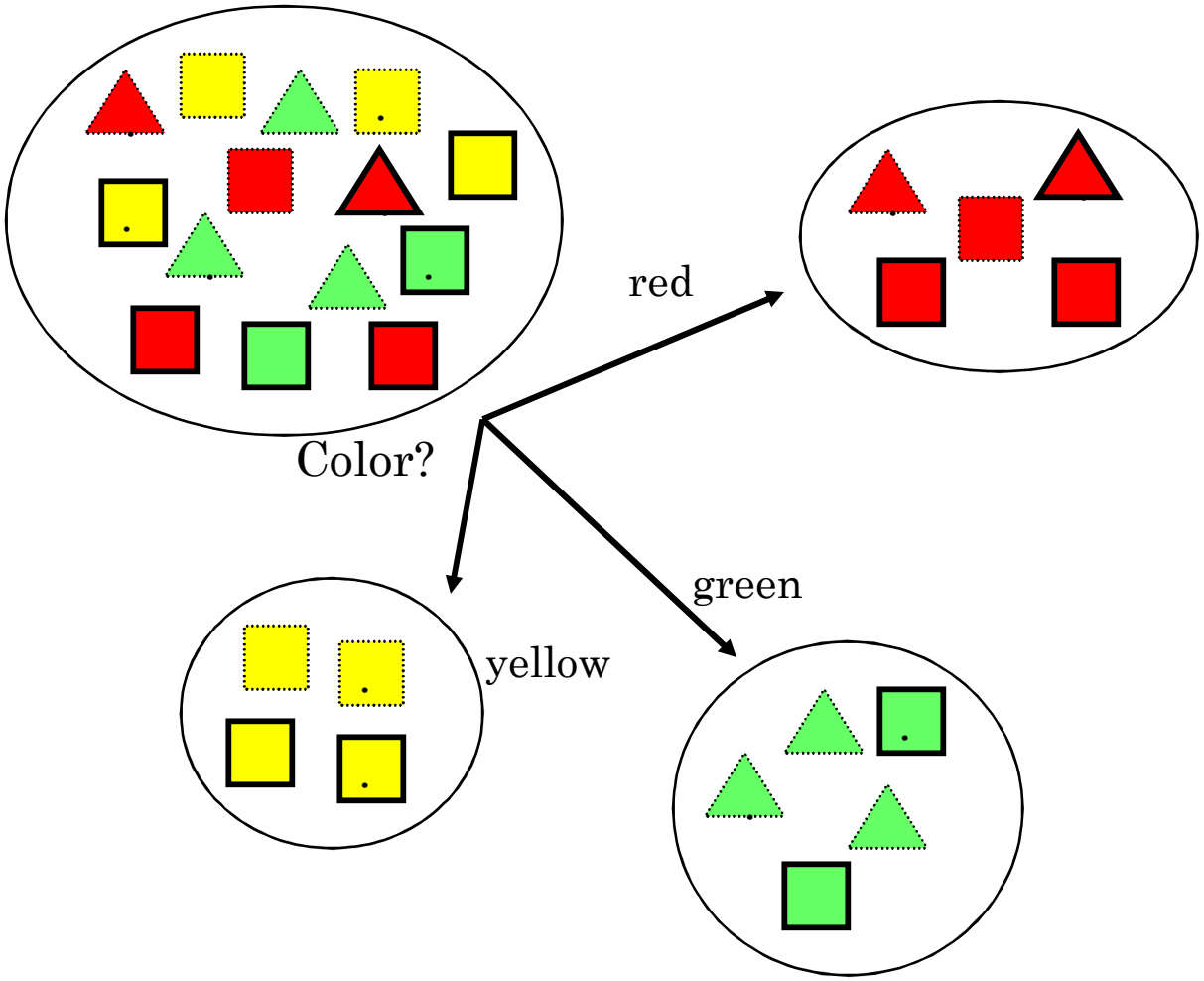
Information Gain



$$Gain(\text{Color}) = I - I_{res}(\text{Color}) = 0.940 - 0.694 = 0.246\ bits$$

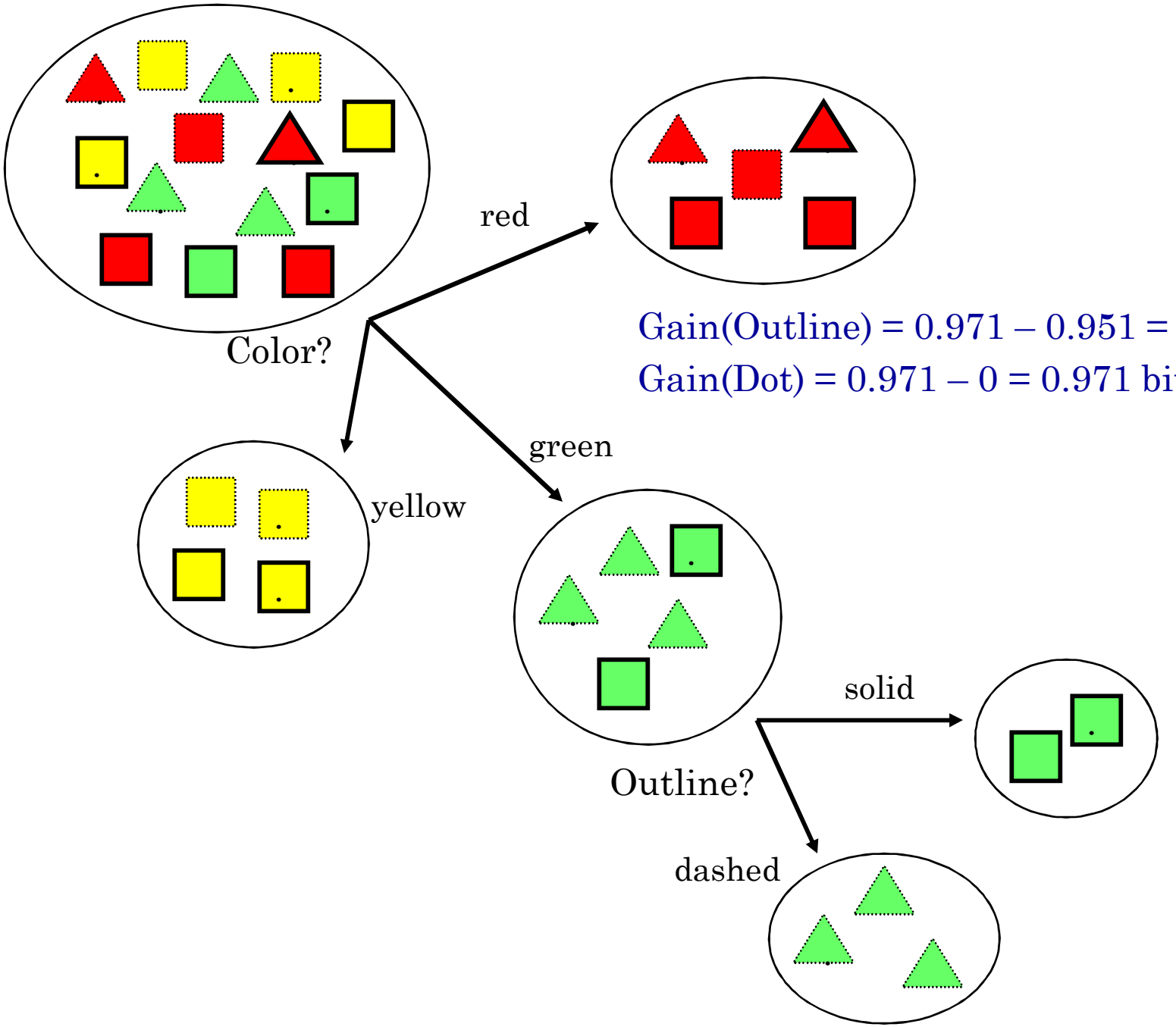
Information Gain of The Attribute

- Attributes
  - Gain(Color) = 0.246
  - Gain(Outline) = 0.151
  - Gain(Dot) = 0.048
- Heuristics: attribute with the highest gain is chosen
- This heuristics is local (local minimization of impurity)



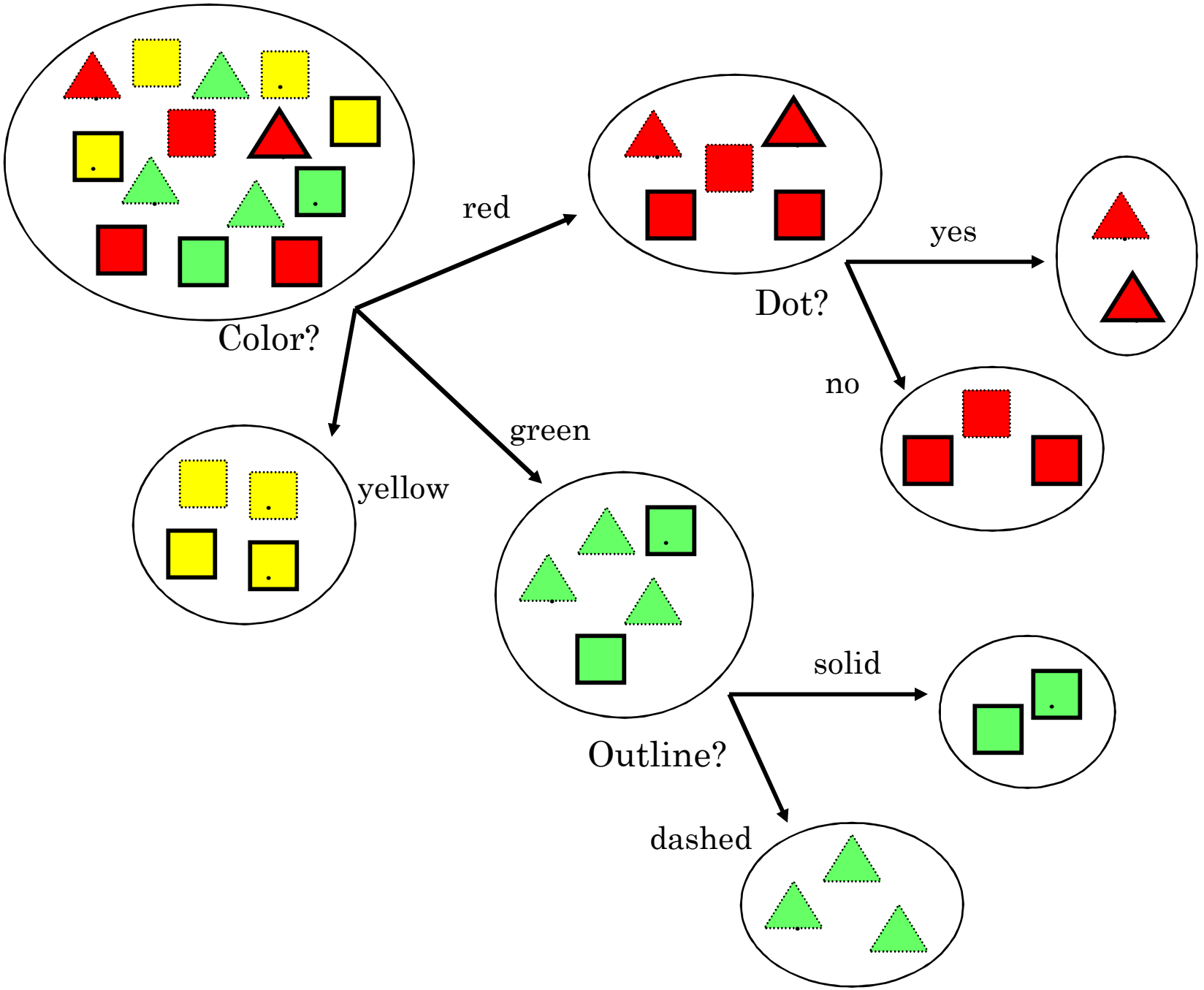
Gain(Outline) = 0.971 – 0 = 0.971 bits

Gain(Dot) = 0.971 – 0.951 = 0.020 bits



Gain(Outline) = 0.971 – 0.951 = 0.020 bits

Gain(Dot) = 0.971 – 0 = 0.971 bits


Color?
red
yellow
green
Dot?
yes
no
Outline?
solid
dashed

Decision Tree