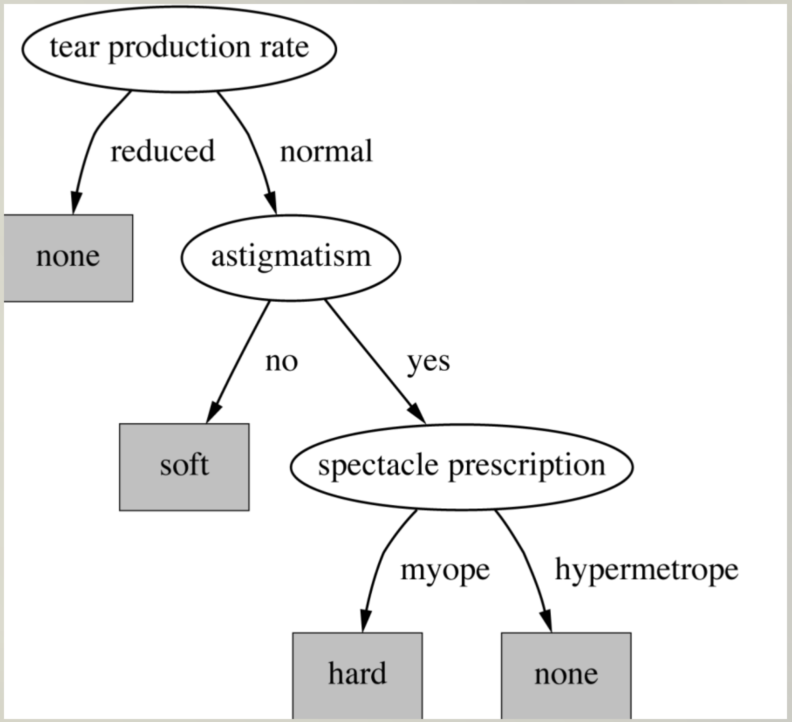
CSCI 446: Artificial Intelligence

Decision Trees

Trees

- “Divide-and-conquer” approach produces tree
- Nodes involve testing a particular attribute
- Usually, attribute value is compared to constant
- Other possibilities:
  - Comparing values of two attributes
  - Using a function of one or more attributes
- Leaves assign classification, set of classifications, or probability distribution to instances
- Unknown instance is routed down the tree

Decision Tree



Nominal and Numeric Attributes

- Nominal:
  - Number of children usually equal to number values
  - Attribute won’t get tested more than once
  - Other possibility: division into two subsets
- Numeric:
  - Test whether value is greater or less than constant
  - Attribute may get tested several times
  - Other possibility: three-way split (or multi-way split)
  - Integer: less than, equal to, greater than
  - Real: below, within, above

Missing Values

- Does absence of value have some significance?
  - Yes: “missing” is a separate value
  - No: “missing” must be treated in a special way
    - Solution A: Assign instance to most popular branch
    - Solution B: Split instance into pieces
      - Pieces receive weight according to fraction of training instances that go down each branch
      - Classifications from leave nodes are combined using the weights that have percolated to them

Constructing Decision Trees

- Strategy: Top Down
  Recursive *divide-and-conquer* fashion

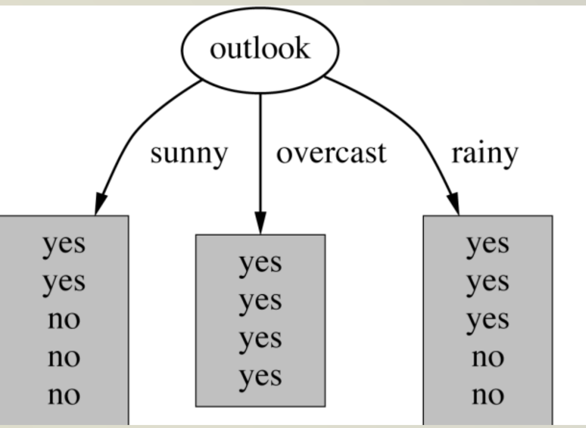
  - First: Select attribute for root node
    Create branch for each possible attribute value
  - Then: Split instances into subsets
    One for each branch extending from the node
  - Finally: Repeat recursively for each branch, using only instances that reach the branch

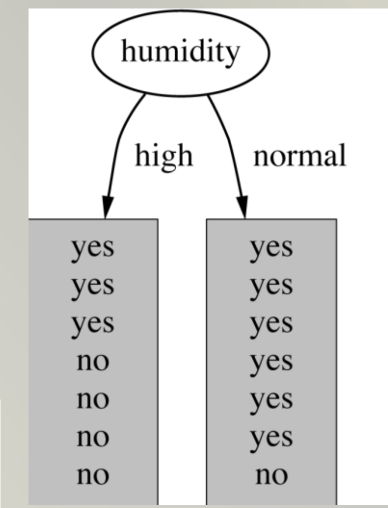
- Stop if all instances have the same class or there are no more attributes to split on

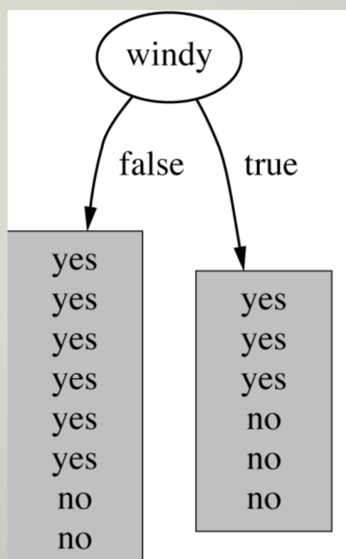
Weather Data with *ID Code*

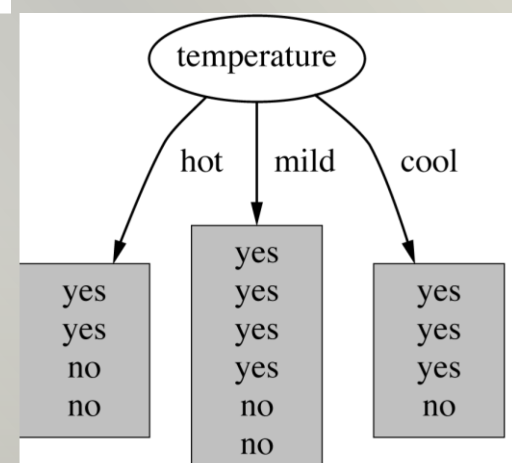
ID code	Outlook	Temp.	Humidity	Wind	Play
A	Sunny	Hot	High	False	No
B	Sunny	Hot	High	True	No
C	Overcast	Hot	High	False	Yes
D	Rainy	Mild	High	False	Yes
E	Rainy	Cool	Normal	False	Yes
F	Rainy	Cool	Normal	True	No
G	Overcast	Cool	Normal	True	Yes
H	Sunny	Mild	High	False	No
I	Sunny	Cool	Normal	False	Yes
J	Rainy	Mild	Normal	False	Yes
K	Sunny	Mild	Normal	True	Yes
L	Overcast	Mild	High	True	Yes
M	Overcast	Hot	Normal	False	Yes
N	Rainy	Mild	High	True	No

Which Attribute to Select?

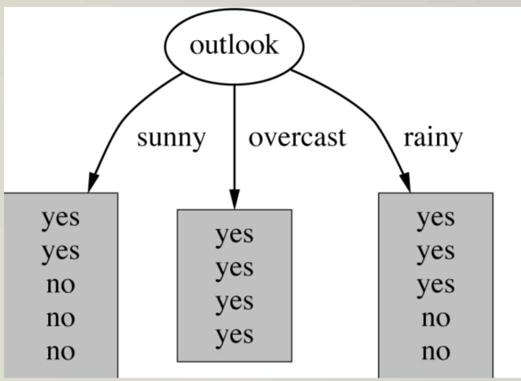


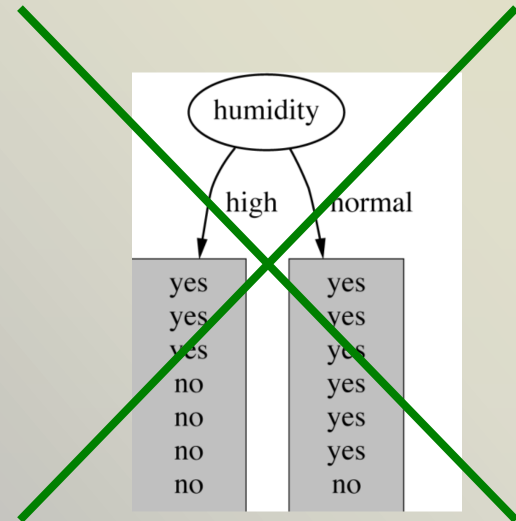


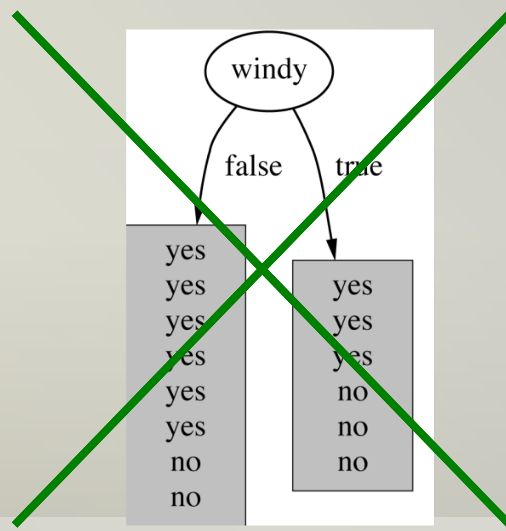


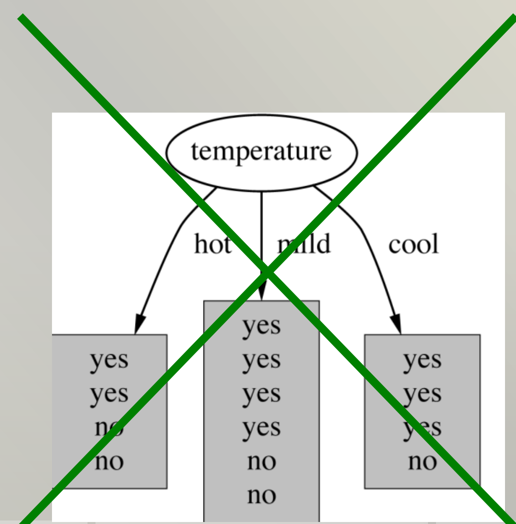


Which Attribute to Select?









Criterion for Attribute Selection

- Which is the best attribute?
  - Want to get the smallest tree
  - Heuristic: choose the attribute that produces the "purest" nodes
- Popular impurity criterion: *information gain*
  - Information gain increases with the average purity of the subsets
- Strategy: Choose attribute that gives greatest information gain

Computing Information

- Measure information in *bits*
  - Given a probability distribution, the info required to predict an event is the distribution's *entropy*
  - Entropy gives the information required in bits (can involve fractions of bits)
    - Because were dealing with bits, the log is calculated in base 2
- Formula for computing the entropy:

$$\text{entropy}(p_{1,}p_{2,}\ldots,p_n) = -p_1 \log p_1 - p_2 \log p_2 \ldots -p_n \log p_n$$

An Algebraic Aside…

- Logarithms:
  - $b^y = x$
  - $y = \log_b x$
  - e.g. $2^4 = 16$, $4 = \log_2 16$
- To change to a different base:
  - $\log_b x = \log_{10} x / \log_{10} b$
  - e.g.

    $\log_2 2 = \log_{10} 2 / \log_{10} 2 = 0.301 / 0.301 = 1$

    $\log_2 4 = \log_{10} 4 / \log_{10} 2 = 0.602 / 0.301 = 2$

    $\log_2 8 = \log_{10} 8 / \log_{10} 2 = 0.9031 / 0.301 = 3$

Example: Attribute *Outlook*

- *Outlook* = *Sunny* :

$$\text{info}([2,3]) = entropy(2/5,3/5) = -2/5\log(2/5) - 3/5\log(3/5) = 0.971\,bits$$

- *Outlook* = *Overcast* :

$$\text{info}([4,0]) = entropy(1,0) = -1\log(1) - 0\log(0) = 0\,bits$$

Note: this is normally undefined.

- *Outlook* = *Rainy* :

$$\text{info}([2,3]) = entropy(3/5,2/5) = -3/5\log(3/5) - 2/5\log(2/5) = 0.971\,bits$$

- Expected information for attribute:

$$\text{info}([3,2],[4,0],[3,2]) = (5/14)\times 0.971 + (4/14)\times 0 + (5/14)\times 0.971 = 0.693\,bits$$

Computing Information Gain

- Information gain: information before splitting – information after splitting

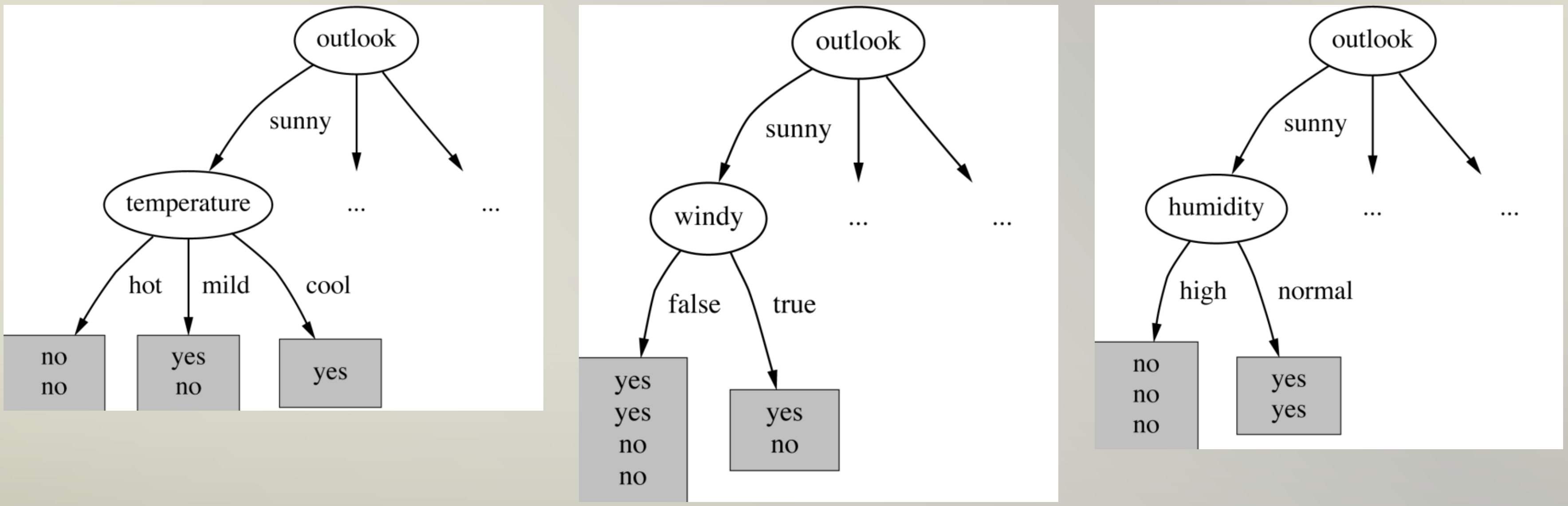
$$\text{gain}(\textit{Outlook}) = \text{info}([9,5]) - \text{info}([2,3],[4,0],[3,2])$$
$$= 0.940 - 0.693$$
$$= 0.247 \text{ bits}$$

- Information gain for attributes from weather data:

gain(*Outlook*)	= 0.247 bits
gain(*Temperature*)	= 0.029 bits
gain(*Humidity*)	= 0.152 bits
gain(*Windy*)	= 0.048 bits

Continuing to Split

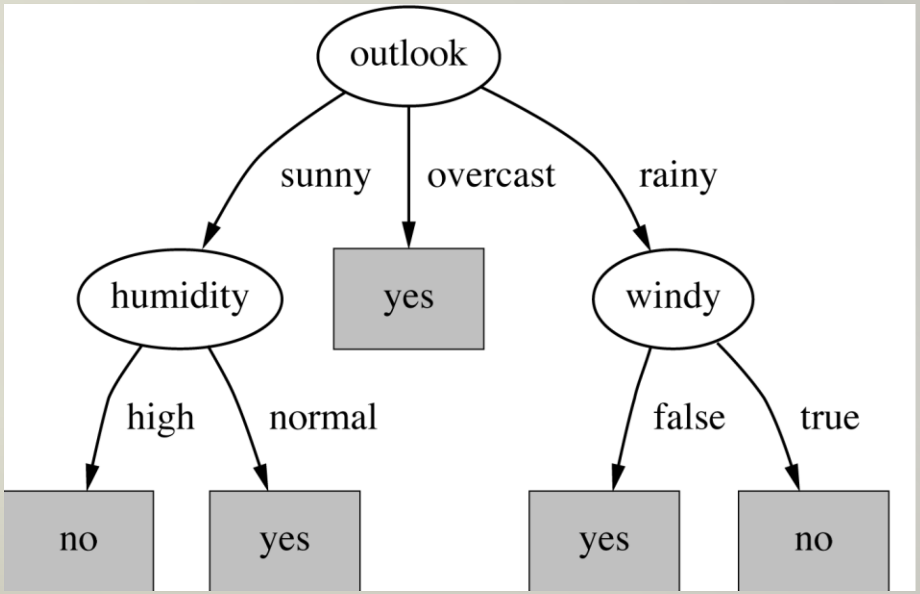
gain(*Temperature*) = 0.571 bits
gain(*Humidity*) = 0.971 bits
gain(*Windy*) = 0.020 bits



Final Decision Tree

- Note: not all leaves need to be pure; sometimes identical instances have different classes

  ⇒ Splitting stops when data can't be split any further



Wishlist for a Purity Measure

- Properties we require from a purity measure:
  - When node is pure, measure should be zero
  - When impurity is maximal (i.e. all classes equally likely), measure should be maximal
  - Measure should obey *multistage property* (i.e. decisions can be made in several stages):

$$\text{measure}([2,3,4]) = measure([2,7]) + (7/9) \times measure([3,4])$$

- Entropy is the only function that satisfies all three properties!

Highly-Branching Attributes

- Problematic - attributes with a large number of values
- Subsets are more likely to be pure if there is a large number of values
  - ⇒Information gain is biased towards choosing attributes with a large number of values
  - ⇒This may result in *overfitting* (selection of an attribute that is non-optimal for prediction)

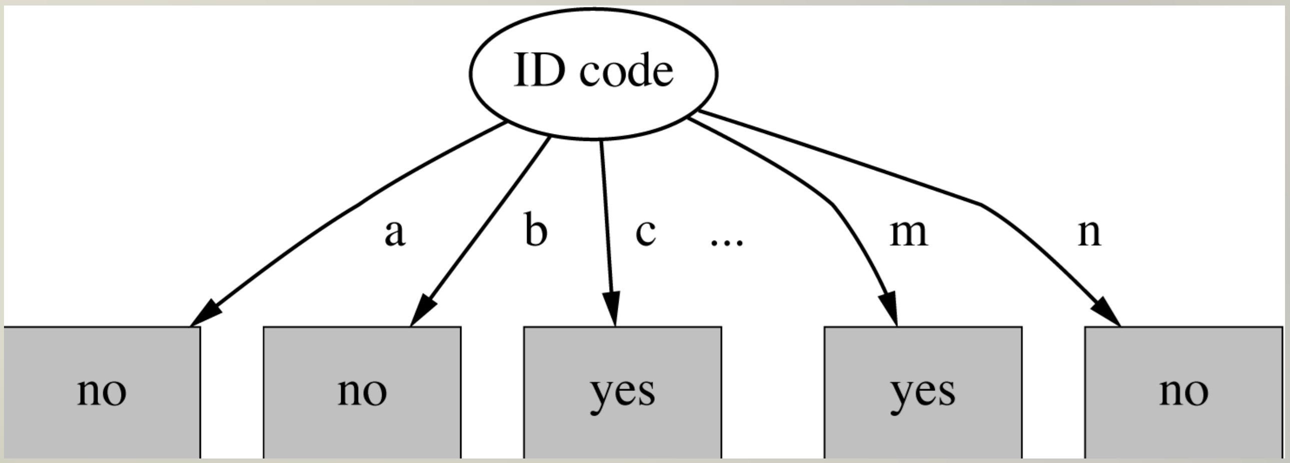
Weather Data with *ID Code*

ID code	Outlook	Temp.	Humidity	Wind	Play
A	Sunny	Hot	High	False	No
B	Sunny	Hot	High	True	No
C	Overcast	Hot	High	False	Yes
D	Rainy	Mild	High	False	Yes
E	Rainy	Cool	Normal	False	Yes
F	Rainy	Cool	Normal	True	No
G	Overcast	Cool	Normal	True	Yes
H	Sunny	Mild	High	False	No
I	Sunny	Cool	Normal	False	Yes
J	Rainy	Mild	Normal	False	Yes
K	Sunny	Mild	Normal	True	Yes
L	Overcast	Mild	High	True	Yes
M	Overcast	Hot	Normal	False	Yes
N	Rainy	Mild	High	True	No

Tree Stump for *ID Code* Attribute

- Entropy of split:

$\Rightarrow$ Information gain is maximal for ID code (namely 0.940 bits)



$$\text{info}(\textit{ID code}) = info([0,1]) + info([0,1]) + \cdots + info([0,1]) = 0 bits$$

Gain Ratio

- *Gain ratio*: a modification of the information gain that reduces its bias
- Gain ratio takes number and size of branches into account when choosing an attribute
  - It corrects the information gain by taking the *intrinsic information* of a split into account
- Intrinsic information:
  - Entropy of distribution of instances into branches (i.e. how much info do we need to tell which branch an instance belongs to)

Computing the Gain Ratio

- Example: intrinsic information for ID code

$$\text{info}([1,1,\ldots,1]) = 14 \times (-1/14 \times \log(1/14)) = 3.807\,bits$$

- Value of attribute decreases as intrinsic information gets larger
- Definition of gain ratio:

$$gain\_ratio(attribute) \quad = \frac{gain(attribute)}{intrinsic\_info(attribute)}$$

- Example:

$$\text{gain\_}ratio(ID\ code) = \frac{0.940\,bits}{3.807\,bits} = 0.246$$

Gain Ratios for Weather Data

Outlook		Temperature	
Info:	0.693	Info:	0.911
Gain: 0.940-0.693	0.247	Gain: 0.940-0.911	0.029
Split info: info([5,4,5])	1.577	Split info: info([4,6,4])	1.557
Gain ratio: 0.247/1.577	0.157	Gain ratio: 0.029/1.557	0.019

Humidity		Windy	
Info:	0.788	Info:	0.892
Gain: 0.940-0.788	0.152	Gain: 0.940-0.892	0.048
Split info: info([7,7])	1.000	Split info: info([8,6])	0.985
Gain ratio: 0.152/1	0.152	Gain ratio: 0.048/0.985	0.049

More on the Gain Ratio

- *Outlook* still comes out top
- However *ID code* still has greater gain ratio
  - Standard fix: *ad hoc* test to prevent splitting on that type of attribute
- Problem with gain ratio: it may overcompensate
  - May choose an attribute just because its intrinsic information is very low
  - Standard fix: only consider attributes with greater than average information gain

Walking Through the Weather Example…

1. Calculate the information value of the problem as a whole.

2. For each attribute:

   A. Calculate the information in each of its potential values.

   B. Calculate the average information value of that attribute.

   C. Calculate the gain by subtracting its value from the information value of the problem as a whole.

3. Calculate the intrinsic information value of the split.

4. Calculate the ratio by dividing the attribute gain by the intrinsic information value.

Walking Through the Example…

1. Calculate the information value of the problem as a whole.

info([9,5]) = entropy(9/14, 5/14)

$= -9/14(\log_2 9/14) – 5/14(\log_2 5/14)$

$= -9/14((\log_{10} 9/14)/(\log_{10} 2)) – 5/14((\log_{10} 5/14)/(\log_{10} 2))$

= 0.940 bits

Walking Through the Example…

2. For each attribute:

A. Calculate the information in each of its potential values.

Outlook = Sunny

info([2,3]) = entropy(2/5, 3/5)

$= -2/5(\log_2 2/5) – 3/5(\log_2 3/5)$

$= -2/5((\log_{10} 2/5)/(\log_{10} 2)) – 3/5((\log_{10} 3/5)/(\log_{10} 2))$

= 0.971 bits

Outlook = Overcast

info([4,0]) = entropy(4/4, 0/4) = entropy(1, 0)

$= -1(\log_2 1) – 0(\log_2 0)$

$= -1((\log_{10} 1)/(\log_{10} 2)) – 0$

= 0 bits

Outlook = Rainy

info([2,3]) = entropy(2/5, 3/5)

$= -2/5(\log_2 2/5) – 3/5(\log_2 3/5)$

$= -2/5((\log_{10} 2/5)/(\log_{10} 2)) – 3/5((\log_{10} 3/5)/(\log_{10} 2))$

= 0.971 bits

Walking Through the Example…

2. For each attribute:

B. Calculate the average information value of that attribute.

info([3,2], [4,0], [3,2])

= 5/14 * 0.971 + 4/14 * 0 + 5/14 * 0.971

= 0.693 bits

Walking Through the Example…

2. For each attribute:

C. Calculate the gain by subtracting its value from the information value of the problem as a whole.

info([9,5]) - info([2,3],[4,0], [2,3])

= 0.940 – 0.693

= 0.247

Walking Through the Example…

3. Calculate the intrinsic information value of the split.

info([5, 4, 5]) = entropy(5/14, 4/14, 5/14)

$= -5/14(\log_2 5/14) – 4/14(\log_2 4/14) – 5/14(\log_2 5/14)$

$= -5/14((\log_{10} 5/14)/(\log_{10} 2)) – 4/14((\log_{10} 4/14)/(\log_{10} 2)) - 5/14((\log_{10} 5/14)/(\log_{10} 2))$

= 1.577 bits

Walking Through the Example…

4. Calculate the ratio by dividing the attribute gain by the intrinsic information value.

Gain Ratio = Gain from Attribute / Intrinsic Value of Split

= 0.247 / 1.577

= 0.157

Walking Through the Example…

- Now you try the math for an attribute, (Temperature, Humidity, or Windy) and see if your numbers come out the same as those listed on slide 19.

Numeric Attributes

- Standard method: binary splits
  - E.g. temp < 45
- Unlike nominal attributes, every attribute has many possible split points
- Solution is straightforward extension:
  - Evaluate info gain (or other measure) for every possible split point of attribute
  - Choose “best” split point
  - Info gain for best split point is info gain for attribute
- Computationally more demanding

Weather Data (Again!)

Outlook	Temperature	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
…	…	…	…	…

```
If outlook = sunny and humidity = high then play = no
If outlook = rainy and windy = true then play = no
If outlook = overcast then play = yes
If humidity = normal then play = yes
If none of the above then play = yes
```

Weather Data (Again!)

Outlook	Temperature	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot			
Overcast	Hot			
Rainy	Mild			
Rainy	Cool			
Rainy	Cool			
…	…			

Outlook	Temperature	Humidity	Windy	Play
Sunny	85	85	False	No
Sunny	80	90	True	No
Overcast	83	86	False	Yes
Rainy	70	96	False	Yes
Rainy	68	80	False	Yes
Rainy	65	70	True	No
…	…	…	…	…

```
If outlook = sunny an
If outlook = rainy and windy = true then play = no
If outlook = overcast then play = yes
If humidity = normal then play = yes
If none of the above th
```

```
If outlook = sunny and humidity > 83 then play = no
If outlook = rainy and windy = true then play = no
If outlook = overcast then play = yes
If humidity < 85 then play = no
If none of the above then play = yes
```

Example

- Split on temperature attribute:

64	65	68	69	70	71	72	72	75	75	80	81	83	85
Yes	**No**	**Yes**	**Yes**	**Yes**	**No**	**No**	**Yes**	**Yes**	**Yes**	**No**	**Yes**	**Yes**	**No**

  - E.g. temperature $<$ 71.5: yes/4, no/2
    temperature $\geq$ 71.5: yes/5, no/3

  - Info([4,2],[5,3])
    = 6/14 info([4,2]) + 8/14 info([5,3])
    = 0.939 bits

- Place split points halfway between values
- Can evaluate all split points in one pass!

Can Avoid Repeated Sorting

- Sort instances by the values of the numeric attribute
  - Time complexity for sorting: $O(n \log n)$
- Does this have to be repeated at each node of the tree?
- No! Sort order for children can be derived from sort order for parent
  - Time complexity of derivation: $O(n)$
  - Drawback: need to create and store an array of sorted indices for each numeric attribute

Binary *vs* Multiway Splits

- Splitting (multi-way) on a nominal attribute exhausts all information in that attribute
  - Nominal attribute is tested (at most) once on any path in the tree
- Not so for binary splits on numeric attributes!
  - Numeric attribute may be tested several times along a path in the tree
- Disadvantage: tree is hard to read
- Remedy:
  - Pre-discretize numeric attributes, *or*
  - Use multi-way splits instead of binary ones

Missing Values

- Split instances with missing values into pieces
  - A piece going down a branch receives a weight proportional to the popularity of the branch
  - Weights sum to 1
- Info gain works with fractional instances
  - Use sums of weights instead of counts
- During classification, split the instance into pieces in the same way
  - Merge probability distribution using weights

Pruning

- Prevent overfitting to noise in the data
- “Prune” the decision tree
- Two strategies:
  - *Postpruning*
    Take a fully-grown decision tree and discard unreliable parts
  - *Prepruning*
    Stop growing a branch when information becomes unreliable
- Postpruning preferred in practice—prepruning can “stop early”

Prepruning

- Based on statistical significance test
  - Stop growing the tree when there is no *statistically significant* association between any attribute and the class at a particular node
- ID3 used chi-squared test in addition to information gain
  - Only statistically significant attributes were allowed to be selected by information gain procedure

Early Stopping

- Pre-pruning may stop the growth process prematurely: *early stopping*
- Classic example: XOR/Parity-problem
  - No *individual* attribute exhibits any significant association to the class
  - Structure is only visible in fully expanded tree
  - Prepruning won't expand the root node
- But: XOR-type problems rare in practice
- And: prepruning faster than postpruning

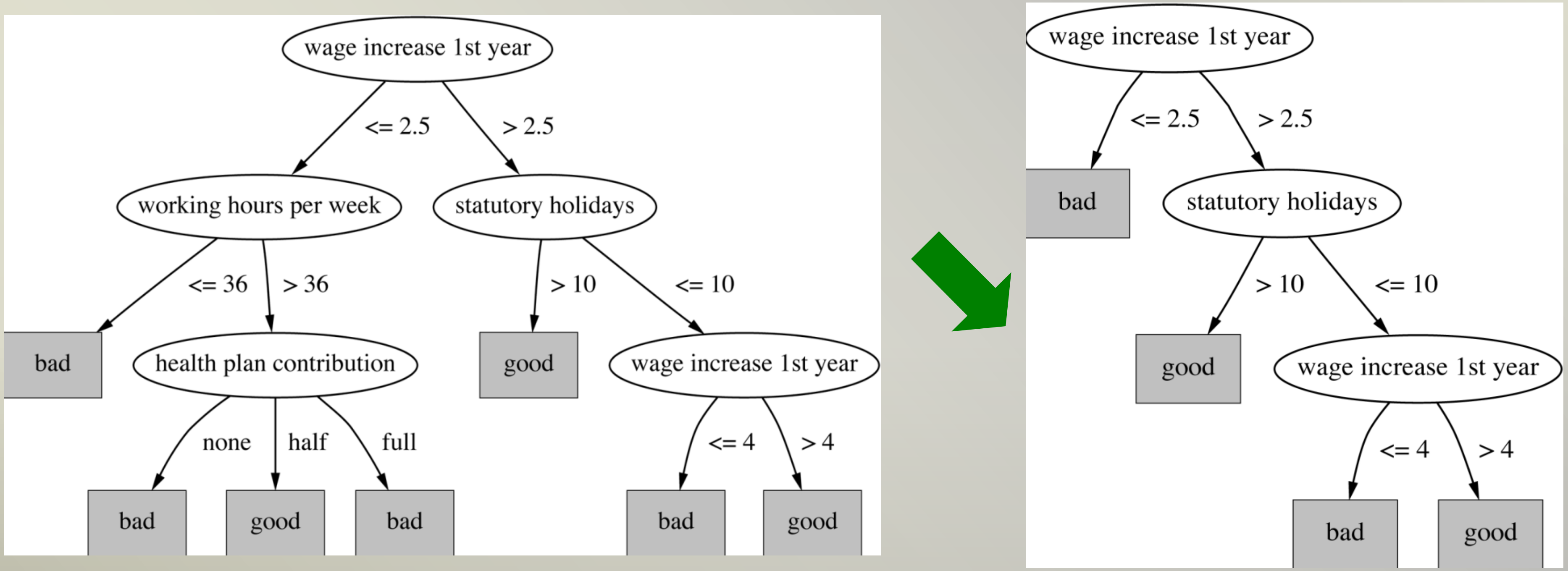
	a	b	class
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0

Postpruning

- First, build full tree
- Then, prune it
  - Fully-grown tree shows all attribute interactions
- Two pruning operations:
  - *Subtree replacement*
  - *Subtree raising*
- Possible strategies:
  - Error estimation
  - Significance testing
  - MDL principle

Subtree Replacement

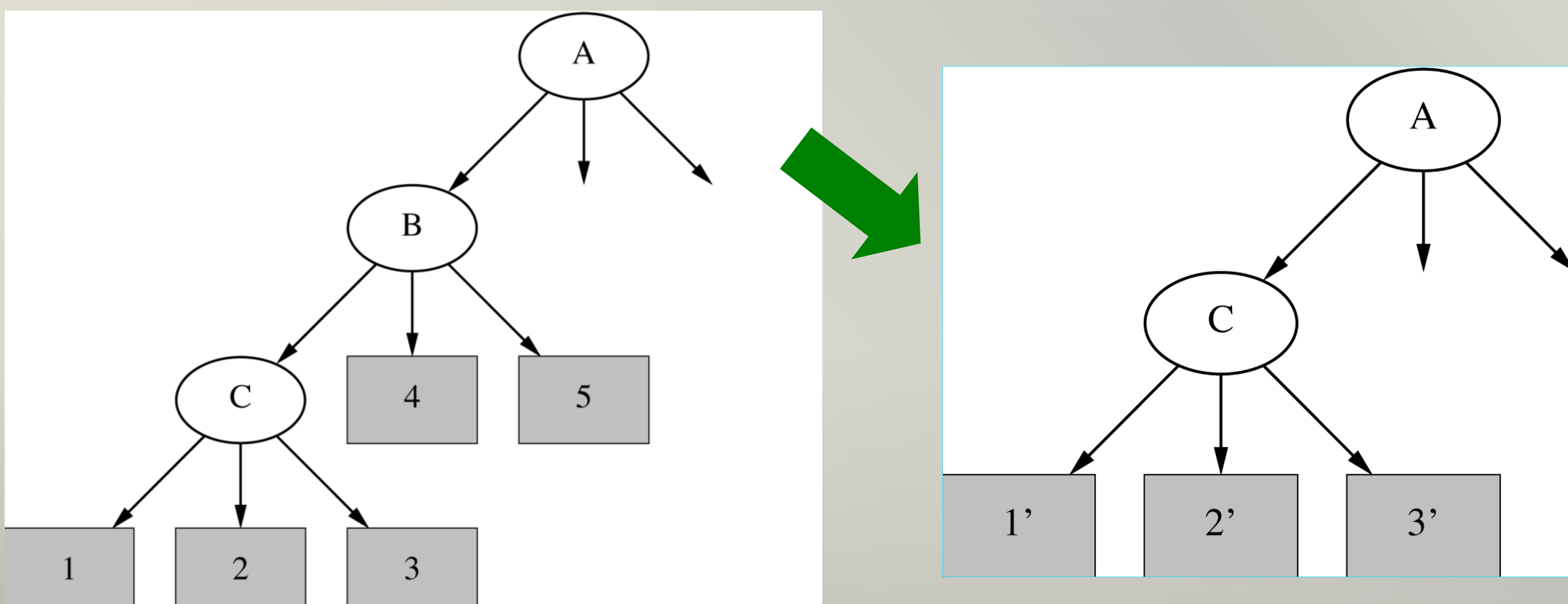
- *Bottom-up*
- Consider replacing a tree only after considering all its subtrees



Subtree Raising

- Delete node
- Redistribute instances
- Slower than subtree replacement

  (Worthwhile?)



Estimating Error Rates

- Prune only if it does not increase the estimated error
- Error on the training data is NOT a useful estimator *(would result in almost no pruning)*
- Use hold-out set for pruning (“reduced-error pruning”)
- C4.5’s method
  - Derive confidence interval from training data
  - Use a heuristic limit, derived from this, for pruning
  - Standard Bernoulli-process-based method
  - Shaky statistical assumptions (based on training data)

Complexity of Tree Induction

- Assume
  - m attributes
  - n training instances
  - tree depth $O(\log n)$
- Building a tree $O(m\, n \log n)$
- Subtree replacement $O(n)$
- Subtree raising $O(n (\log n)^2)$
  - Every instance may have to be redistributed at every node between its leaf and the root
  - Cost for redistribution (on average): $O(\log n)$
  - Total cost: $O(m\, n \log n) + O(n (\log n)^2)$

Discussion

- The most extensively studied method of machine learning
- Different criteria for attribute/test selection rarely make a large difference
- Different pruning methods mainly change the size of the resulting pruned tree