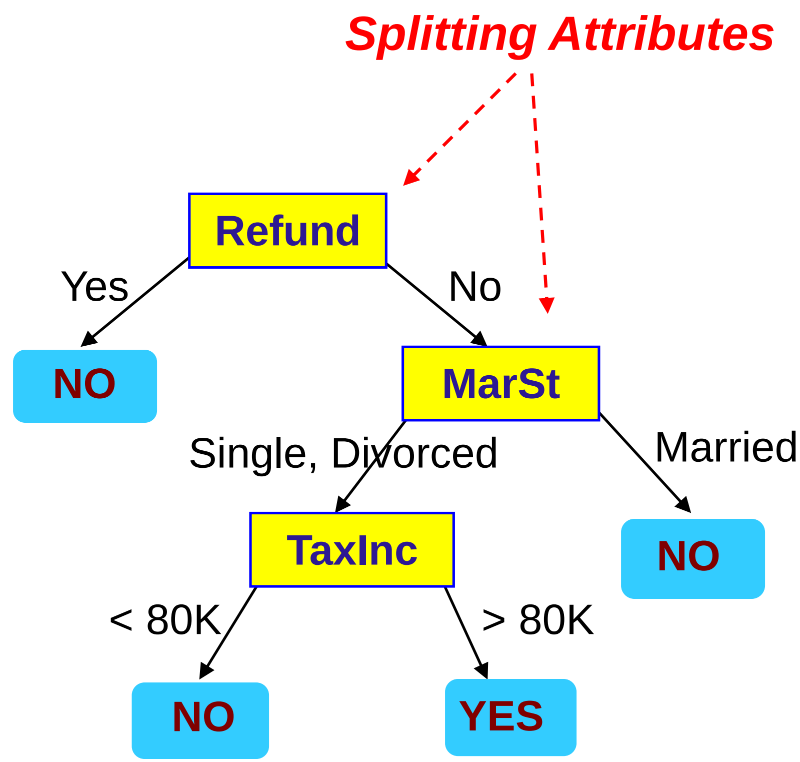
Decision Trees

Example of a 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

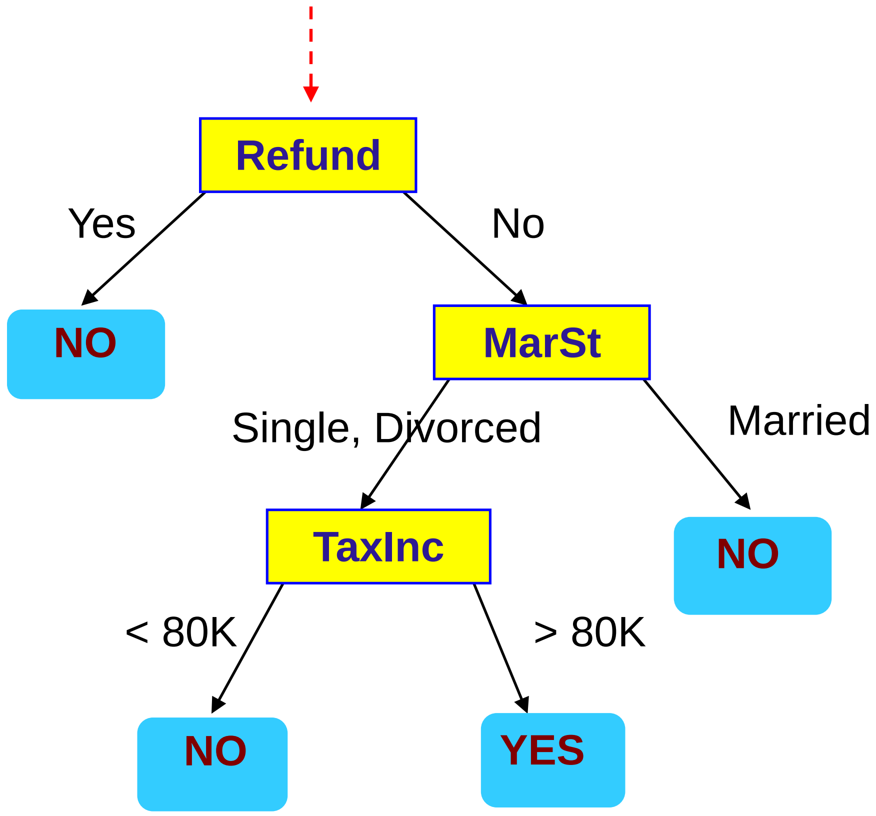
Training Data



Model: Decision Tree

Apply Model to Test Data

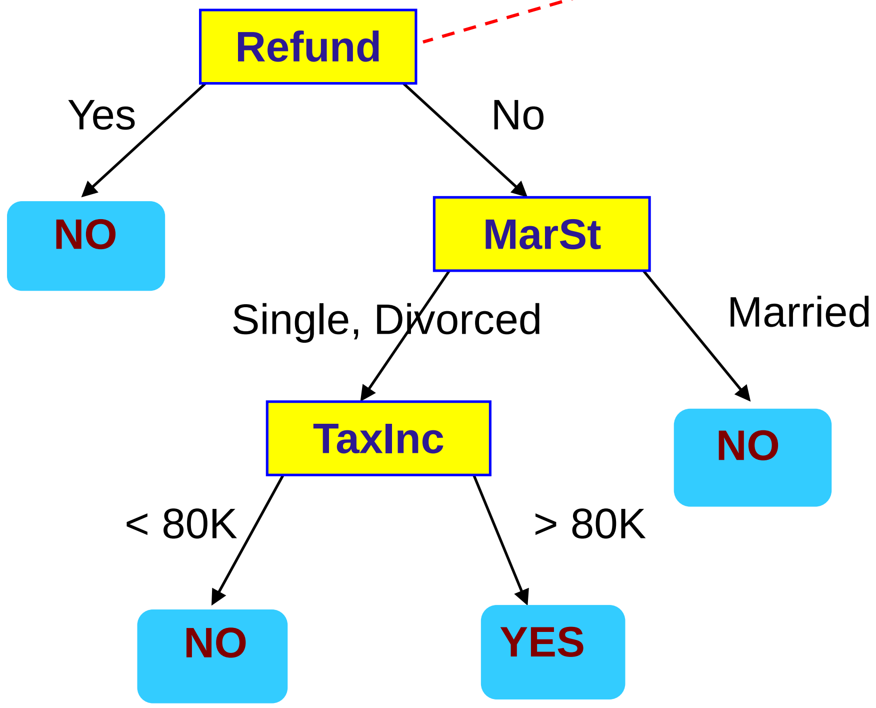
Start from the root of tree.



Test Data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

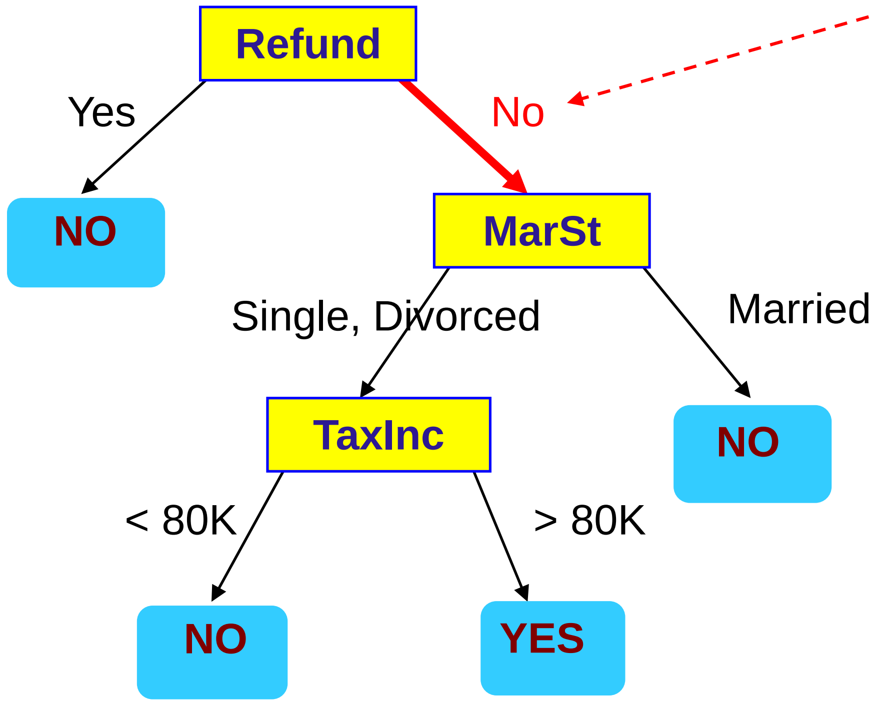
Apply Model to Test Data



Test Data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

Apply Model to Test Data



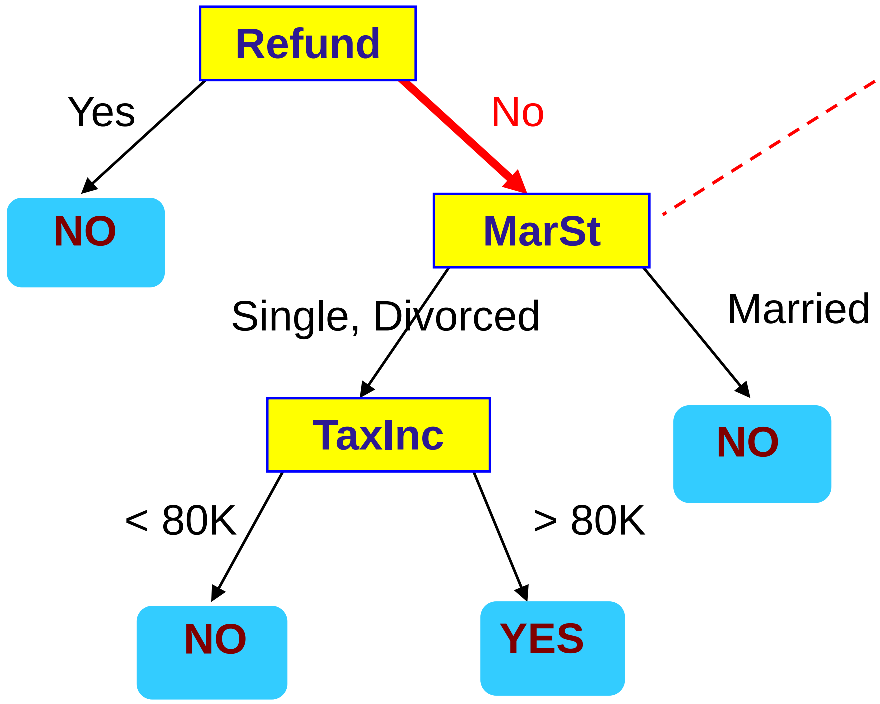
Test Data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

Apply Model to Test Data

Test Data

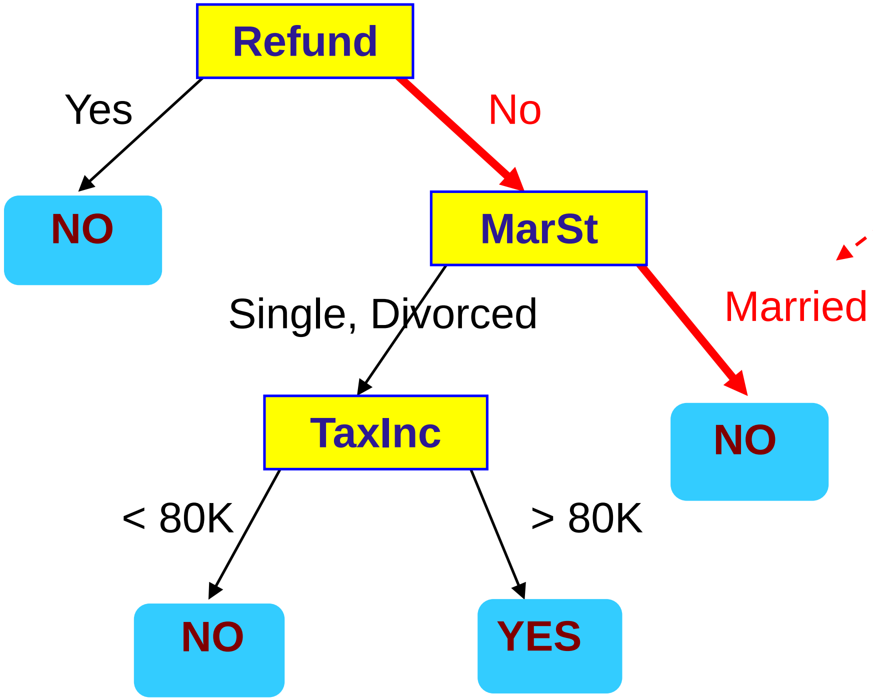
Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	**?**



Apply Model to Test Data

Test Data

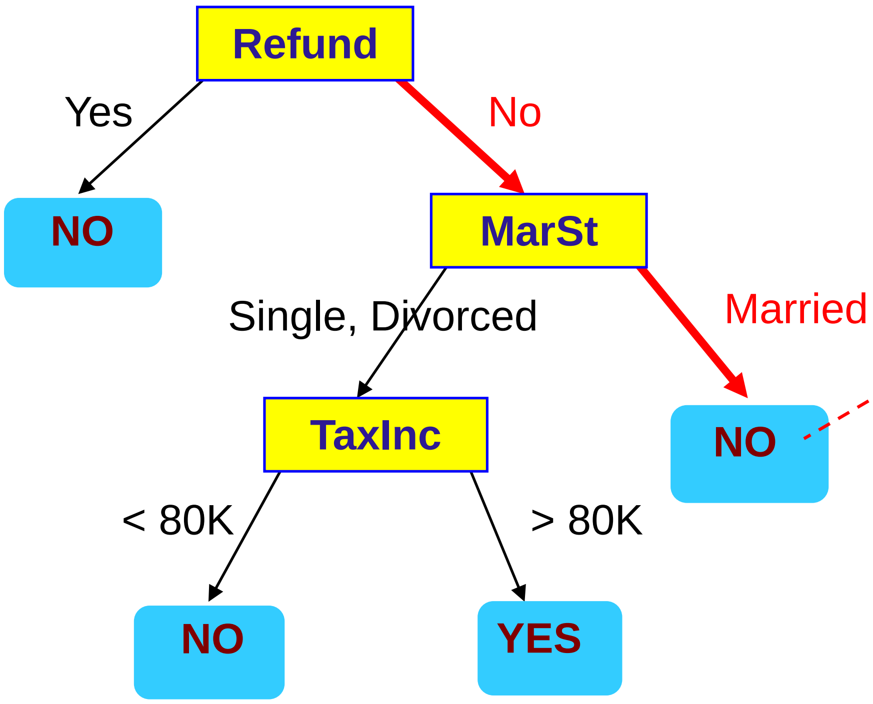
Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?



Apply Model to Test Data

Test Data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	**?**



Assign Cheat to “No”

Digression: Entropy

Bits

- We are watching a set of independent random samples of X
- We see that X has four possible values

P(X=A) = 1/4	P(X=B) = 1/4	P(X=C) = 1/4	P(X=D) = 1/4

- So we might see: BAACBADCDADDDA…
- We can encode each symbol with two bits (e.g. A=00, B=01, C=10, D = 11)

0100001001001110110011111100…

Fewer Bits

- Someone tells us that the probabilities are not equal

$P(X=A) = 1/2$	$P(X=B) = 1/4$	$P(X=C) = 1/8$	$P(X=D) = 1/8$

- It is possible…

…to invent a coding for your transmission that only uses 1.75 bits on average per symbol. Here is one.

A	0
B	10
C	110
D	111

Bound

- Suppose X can have one of *m* values…

$P(X=V_1) = p_1$	$P(X=V_2) = p_2$	….	$P(X=V_m) = p_m$

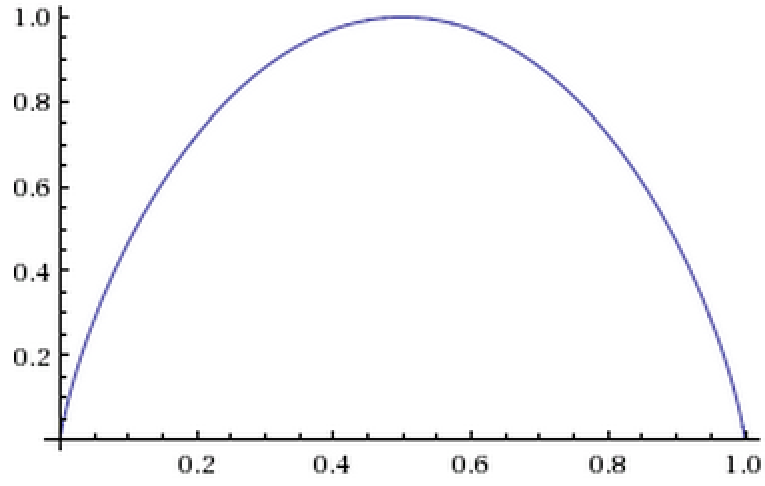
- What's the **smallest possible number of bits**, on average, per symbol, needed to code a stream of symbols drawn from X's distribution?
- Shannon (1948) showed that this number is:

$$entropy(p_1,\dots,p_m) = -p_1 \log_2 p_1 - \dots - p_m \log_2 p_m$$

For the previous example:
-(1/2)log(1/2)-(1/4)log(1/4)-(1/8)log(1/8)-(1/8)log(1/8) = 1.75

Entropy chart for two values

Entropy =
-p1*log(p1)-p2*log(p2)=
-p1*log(p1)-(1-p1)*log(1-p1)



p1

p1+p2=1

For two values, the closest p1 and p2 are to each other, the bigger the entropy. The farther p1 and p2 are from each other, the smaller the entropy, e.g. if there are mostly records having the first value (data is very homogenous, or “dull”), then the entropy is small.

Entropy is also known as a **measure of information**. The greater the entropy, the more information there is. The smaller the entropy, the less information there is (we say the dataset is “dull”).

Back to Decision Trees

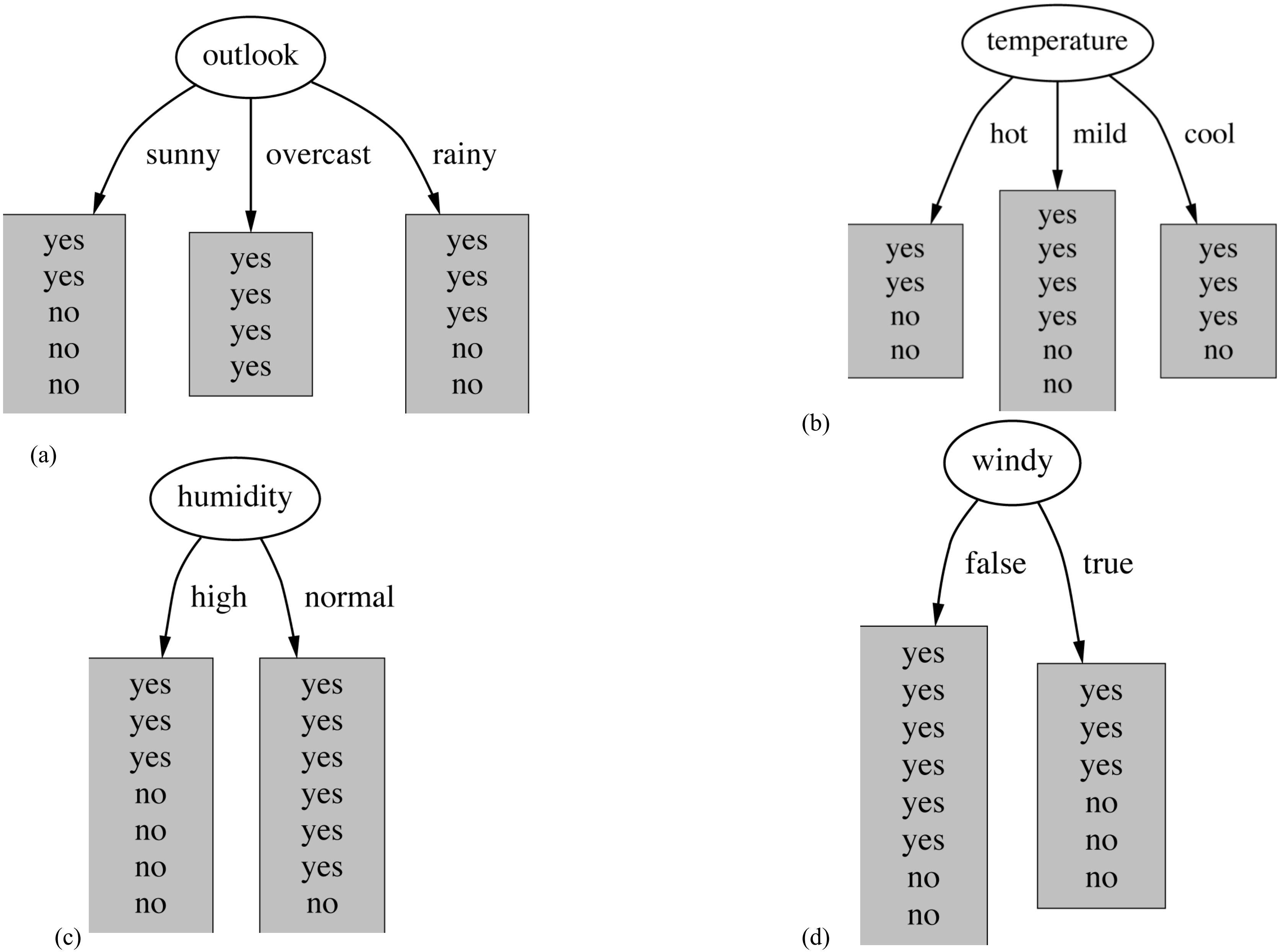
Constructing decision trees (ID3)

- Normal procedure: top down in a recursive **divide-and-conquer** fashion
  - First: an attribute is selected for the root node and a branch is created for each possible attribute value
  - Then: the instances are split into subsets (one for each branch extending from the node)
  - Finally: the same procedure is repeated recursively for each branch, using only instances that reach the branch
- Process stops if all instances have the same class

Weather data

Outlook	**Temp**	**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
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

Which attribute to select?



A criterion for attribute selection

- Which is the best attribute?

- The one which will result in the smallest tree
  - Heuristic: choose the attribute that produces the "purest" or "dullest" nodes

- Popular impurity criterion: entropy of nodes
  - **Lower the entropy, purer the node**.

- Strategy: choose attribute that results in lowest entropy of the children nodes.

Attribute “Outlook”

outlook=sunny

info([2,3]) = entropy(2/5,3/5) = -2/5*log(2/5) -3/5*log(3/5) = .971

outlook=overcast

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

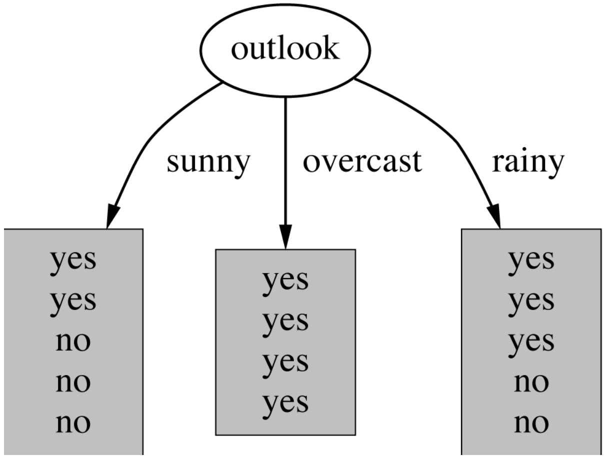
0*log(0) is normally not defined.

outlook=rainy

info([3,2]) = entropy(3/5,2/5) = -3/5*log(3/5)-2/5*log(2/5) = .971

Expected info:

info([2,3],[4,0],[3,2]) = .971*(5/14) + 0*(4/14) + .971*(5/14) = **.693**



Attribute “Temperature”

temperature=hot

info([2,2]) = entropy(2/4,2/4) = -2/4*log(2/4) -2/4*log(2/4) = 1

temperature=mild

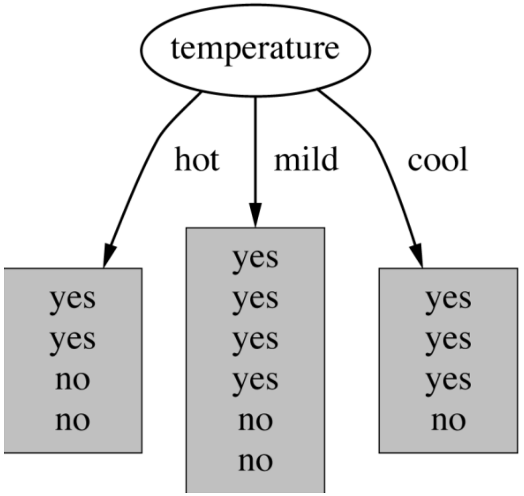
info([4,2]) = entropy(4/6,2/6) = -4/6*log(1) -2/6*log(2/6) = .528

temperature=cool

info([3,1]) = entropy(3/4,1/4) = -3/4*log(3/4)-1/4*log(1/4) = .811

Expected info:

info([2,2],[4,2],[3,1]) = 1*(4/14) + .528*(6/14) + .811*(4/14) = **.744**



Attribute "Humidity"

humidity=high

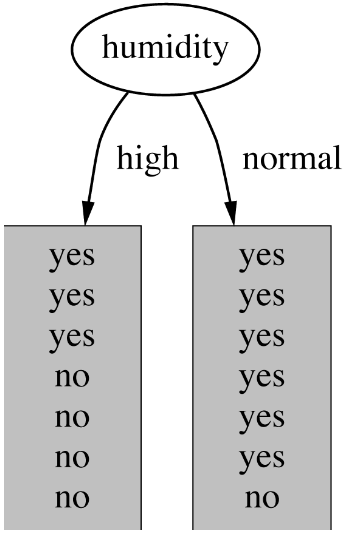
info([3,4]) = entropy(3/7,4/7) = -3/7*log(3/7) -4/7*log(4/7) = .985

humidity=normal

info([6,1]) = entropy(6/7,1/7) = -6/7*log(6/7) -1/7*log(1/7) = .592

Expected info:

info([3,4],[6,1]) = .985*(7/14) + .592*(7/14) = **.788**



Attribute “Windy”

windy=false

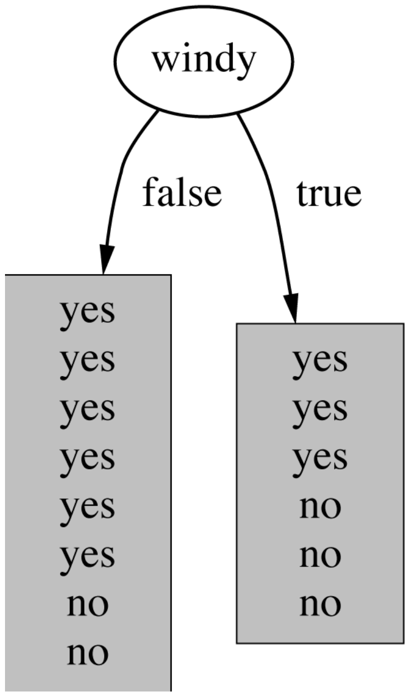
info([6,2]) = entropy(6/8,2/8) = -6/8*log(6/8) -2/8*log(2/8) = .811

humidity=true

info([3,3]) = entropy(3/6,3/6) = -3/6*log(3/6) -3/6*log(3/6) = 1

Expected info:

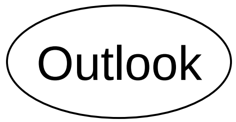
info([6,2],[3,3]) = .811*(8/14) + 1*(6/14) = **.892**



And the winner is...

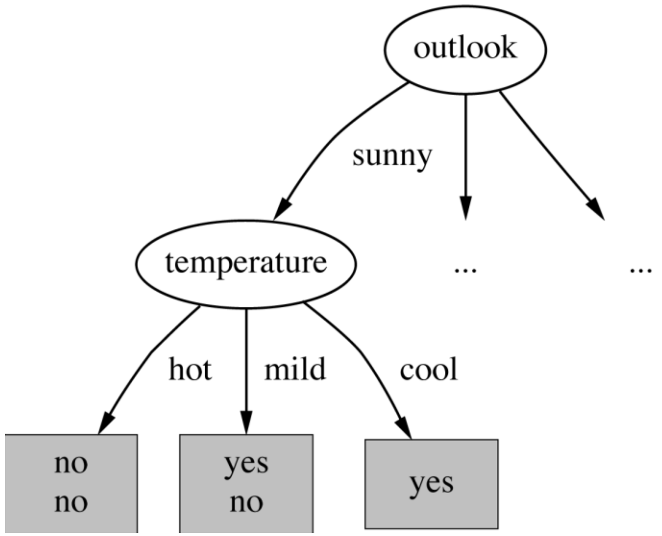
"Outlook"

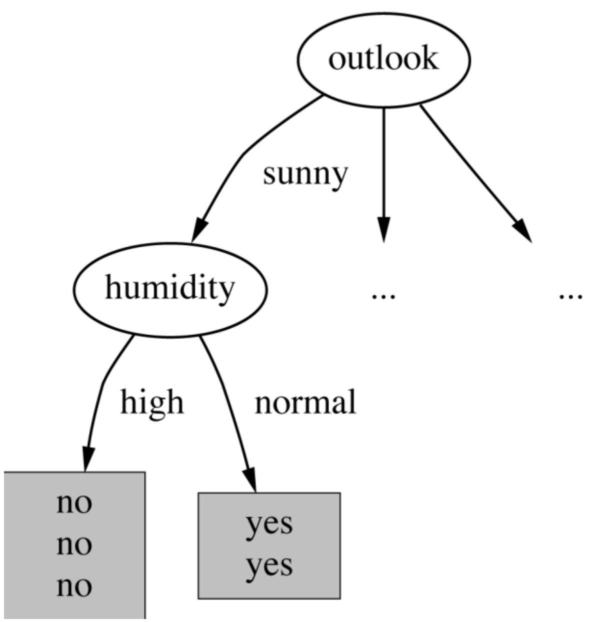
...So, the root will be "Outlook"



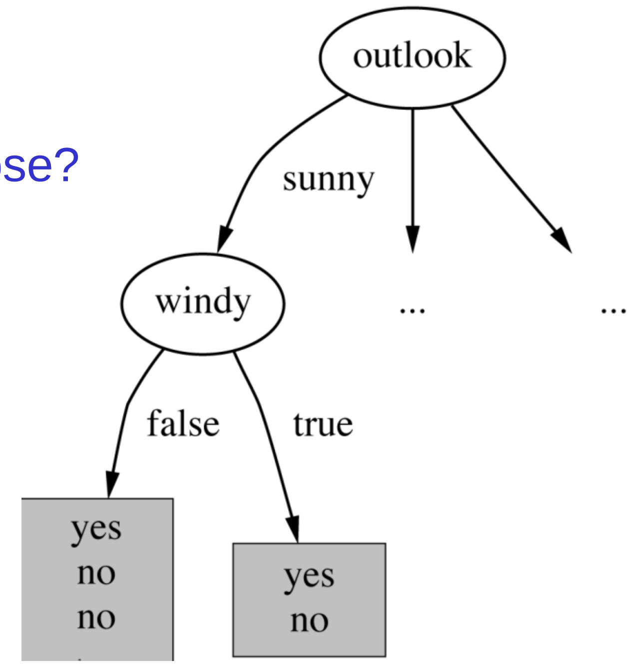
Continuing to split (for Outlook="Sunny")

Outlook	**Temp**	**Humidity**	**Windy**	**Play**
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Sunny	Mild	Normal	True	Yes





Which one to choose?



Continuing to split (for Outlook="Sunny")

temperature=hot: info([2,0]) = entropy(2/2,0/2) = 0

temperature=mild: info([1,1]) = entropy(1/2,1/2) = 1

temperature=cool: info([1,0]) = entropy(1/1,0/1) = 0

Expected info: 0*(2/5) + 1*(2/5) + 0*(1/5) = **.4**

humidity=high: info([3,0]) = 0

humidity=normal: info([2,0]) = 0

Expected info: **0**

windy=false: info([1,2]) = entropy(1/3,2/3) =

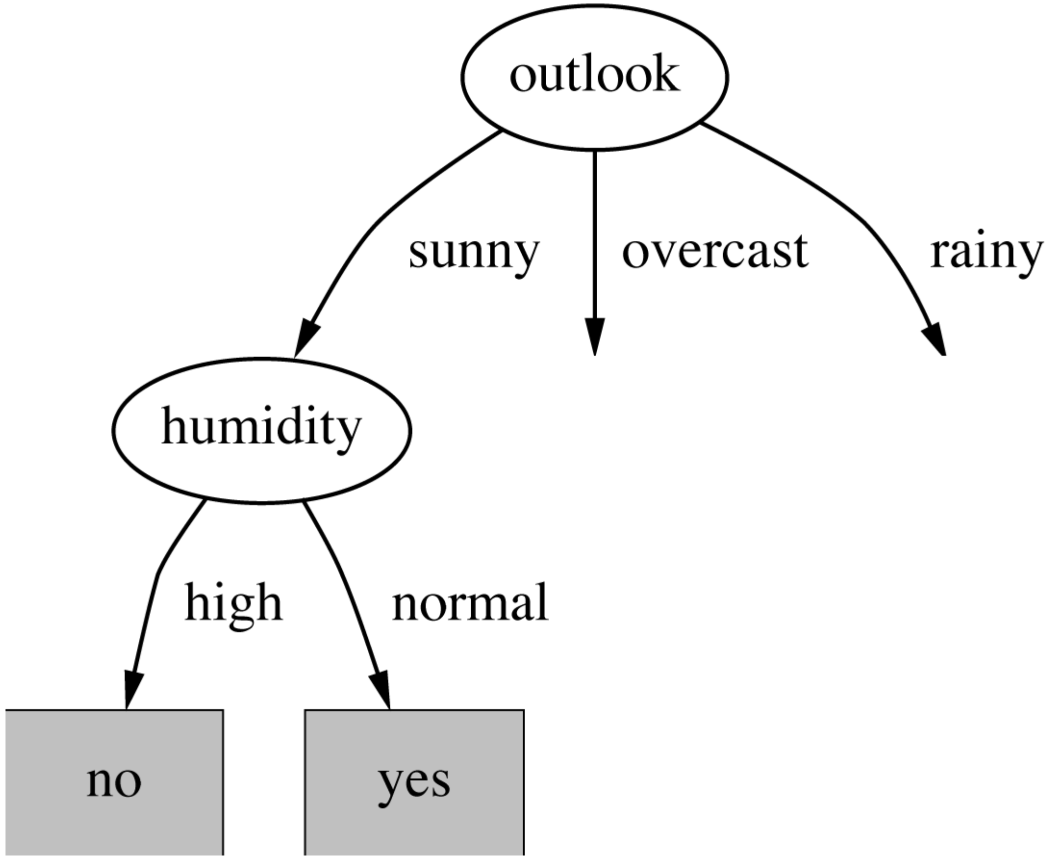
-1/3*log(1/3) -2/3*log(2/3) = .918

windy=true: info([1,1]) = entropy(1/2,1/2) = 1

Expected info: .918*(3/5) + 1*(2/5) = **.951**

Winner is "humidity"

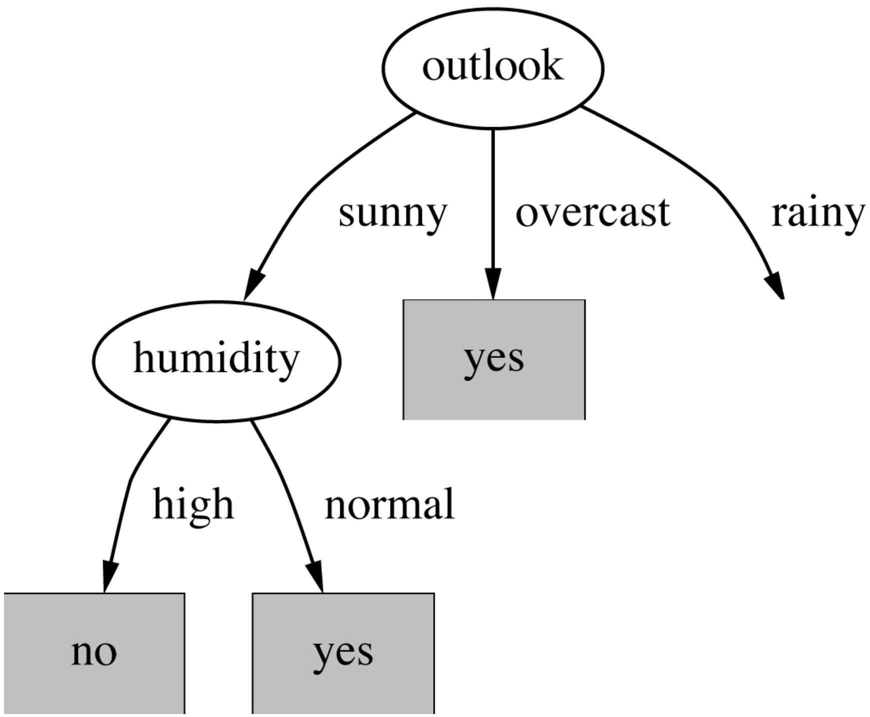
Tree so far



Continuing to split (for Outlook="Overcast")

Outlook	Temp	Humidity	Windy	Play
Overcast	Hot	High	False	Yes
Overcast	Cool	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes

- Nothing to split here, "play" is always "yes".



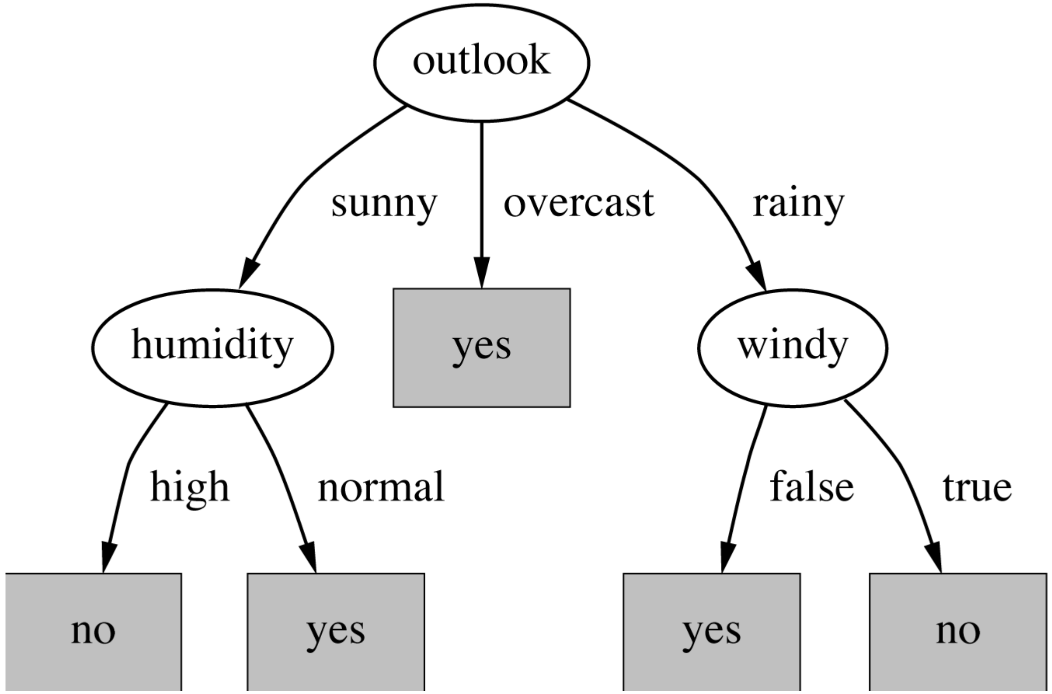
Tree so far

Continuing to split (for Outlook="Rainy")

Outlook	Temp	Humidity	Windy	Play
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Rainy	Mild	Normal	False	Yes
Rainy	Mild	High	True	No

- We can easily see that "Windy" is the one to choose. (Why?)

The 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

Information gain

- Sometimes, people don't use directly the entropy of a node. Rather they talk about the "information gain".
  - The result though will be exactly the same.

- Info-gain: *information before splitting – information after splitting*.

```
gain(Outlook)  = info([9,5])-info([2,3],[4,0],[3,2]) = .940-.693 =.247 bits
gain(Temp)     = info([9,5])-info([2,2],[4,2],[3,1]) = .940-.744 =.196 bits
gain(Humidity) = info([9,5])-info([3,4],[6,1])       = .940-.788 =.152 bits
gain(Windy)    = info([9,5])-info([6,2],[3,3])       = .940-.892 =.048 bits
```

- Clearly, **the greater the info-gain the better the purity** of a node.
  - So, we choose "**Outlook**" for the root.

Discussion

- Algorithm for top-down induction of decision trees (“ID3” - **Iterative Dichotomiser**) was developed by **Ross Quinlan**
  - University of Sydney Australia

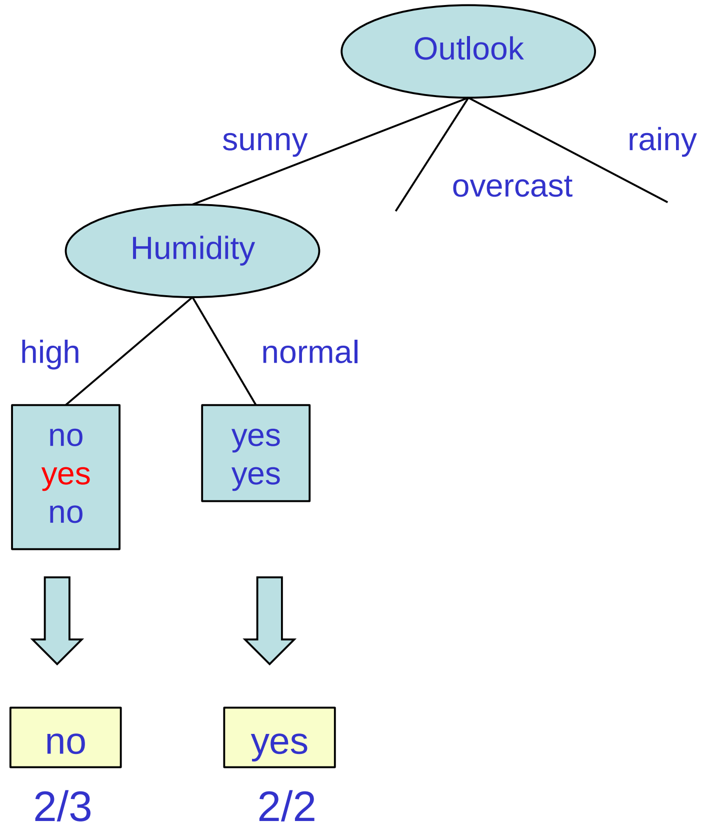
- Led to development of C4.5, which can deal with
    - numeric attributes
    - missing values
    - noisy data

Noisy data

- Not all leaves need to be pure; sometimes identical tuples have different class values
  - Splitting stops when data can't be split any further

ID	Outlook	Temp	Humidity	Windy	Play
1	sunny	hot	high	false	no
2	sunny	hot	high	false	yes
8	sunny	mild	high	false	no
9	sunny	cool	normal	false	yes
11	sunny	mild	normal	true	yes

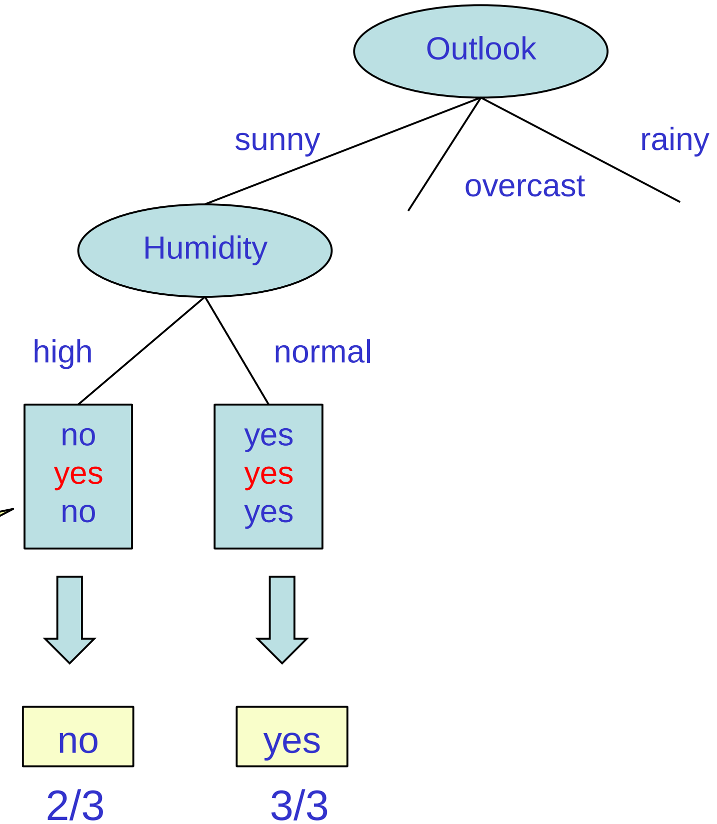
No chance to split and achieve perfect purity. All attributes (except ID and Play) have the same values for tuple 1 and 2.



Missing data

- Sometimes, some attributes of some tuples have missing values

ID	Outlook	Temp	Humidity	Windy	Play
1	sunny	hot	high	false	no
2	sunny	hot	?	false	yes
8	sunny	mild	high	false	no
9	sunny	cool	normal	false	yes
11	sunny	mild	normal	true	yes

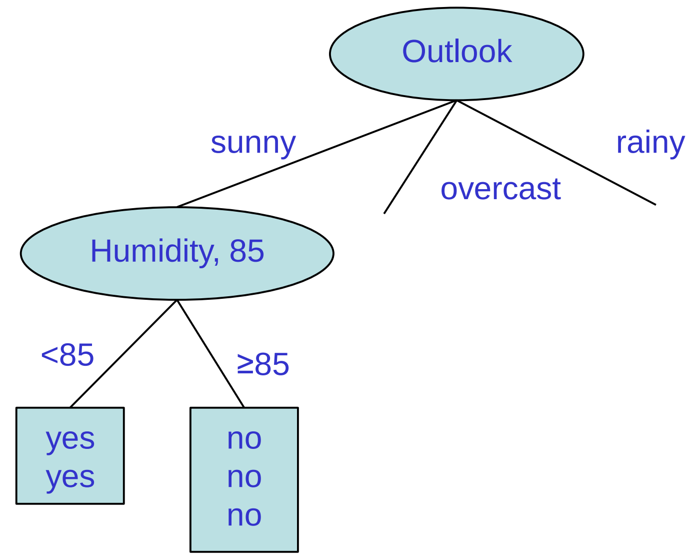


Tuple 2 is sent both branches of Humidity.
This is because we don't know its Humidity value.

Numeric attributes

- Some attributes can be numeric.
- No problem, we can have binary splits (≥v, <v), still use Entropy

ID	Outlook	Temp	Humidity	Windy	Play
1	sunny	85	85	false	no
2	sunny	80	90	true	no
3	overcast	83	86	false	yes
4	rainy	70	96	false	yes
5	rainy	68	80	false	yes
6	rainy	65	70	true	no
7	overcast	64	65	true	yes
8	sunny	72	95	false	no
9	sunny	69	70	false	yes
10	rainy	75	80	false	yes
11	sunny	75	70	true	yes
12	overcast	72	90	true	yes
13	overcast	81	75	false	yes
14	rainy	71	91	true	no



ID	Outlook	Temp	Humidity	Windy	Play
1	sunny	69	70	false	no
2	sunny	75	70	true	no
8	sunny	85	85	false	no
9	sunny	80	90	false	yes
11	sunny	72	95	true	yes

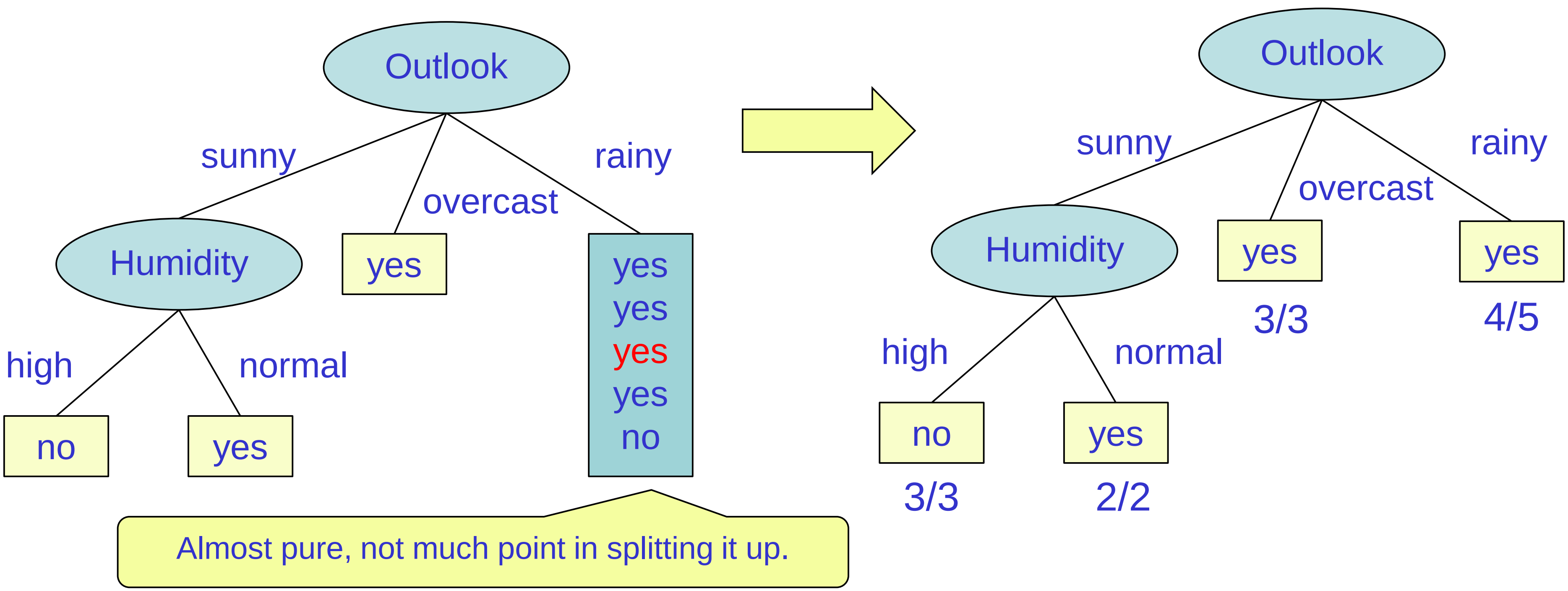
Pruning the tree

- Not always a good idea to grow the tree exhaustively
  - Saying goes:
    - “tree will over fit the training data”
    - “tree will not “abstract well to classify new data”
- Solutions
  - **Pre-pruning**
  - **Post-pruning**

Pre-pruning

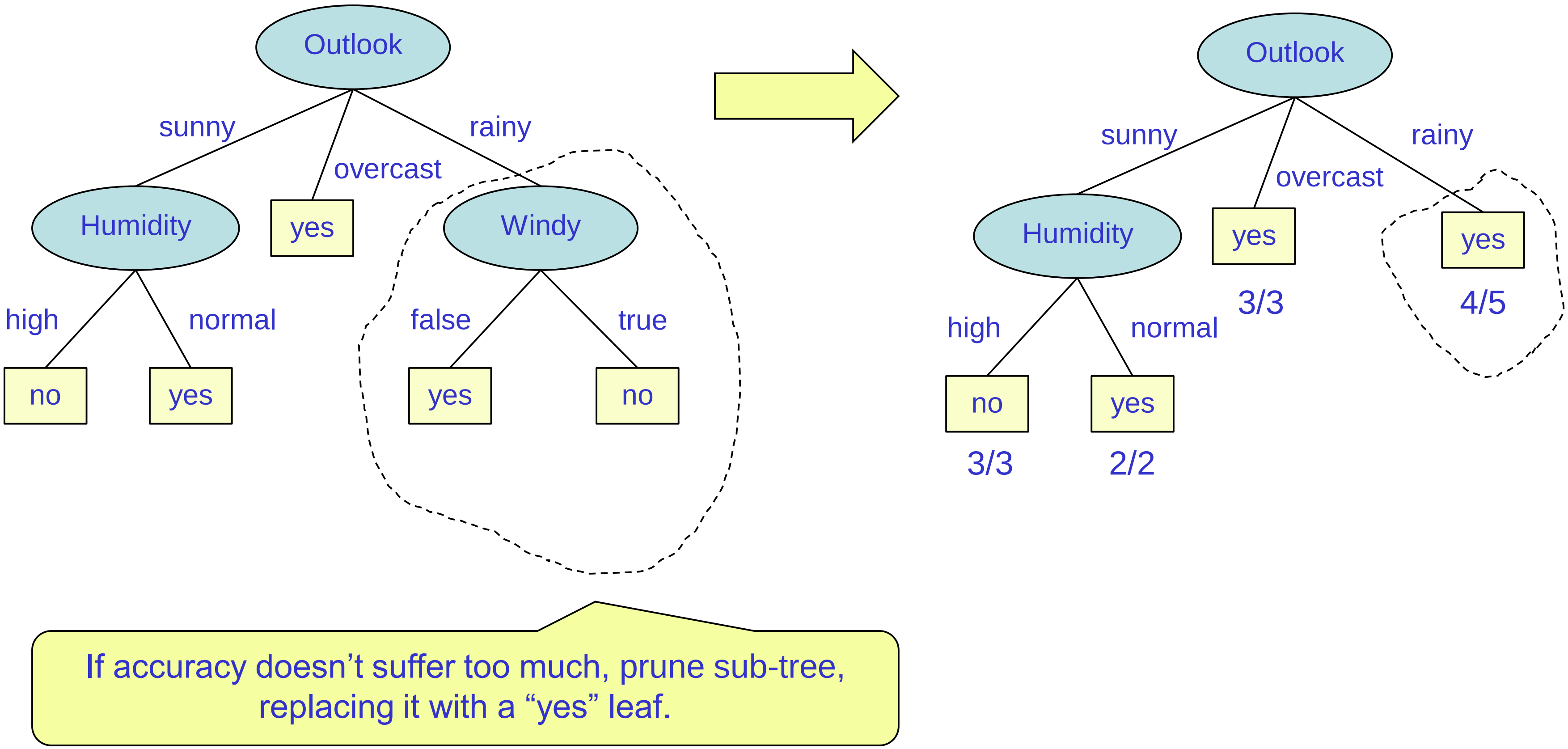
- Don’t split beyond a certain point

ID	Outlook	Temp	Humidity	Windy	Play
4	rainy	mild	high	false	yes
5	rainy	cool	normal	false	yes
6	rainy	cool	normal	true	yes
10	rainy	mild	normal	false	yes
14	rainy	mild	high	true	no



Post-pruning

- Grow first tree exhaustively, then remove those sub-trees that don't cause significant decrease in accuracy.



Random Forest Construction

Each tree is constructed using the following algorithm:

Input

- *N* training cases with *M* attributes each.
- Number *m* (<M) of attributes to be used to determine the decision at a node of the tree
- Number *n* (<N) of training cases to be used for one tree.

Algorithm

- Choose a training set for this tree by choosing *n* times with replacement from all *N* available training cases.
- **For each node of the tree**, randomly choose *m* attributes on which to base the decision at that node. Calculate the best split based on these *m* attributes.
- Fully grow the tree.

Random Forest Prediction

- The new sample is pushed down a tree.
- It is assigned the label of the terminal node it ends up in.
- This procedure is iterated over all trees in the ensemble (forest), and the majority vote of all trees is reported as random forest prediction.