Decision Tree

Decision-Tree

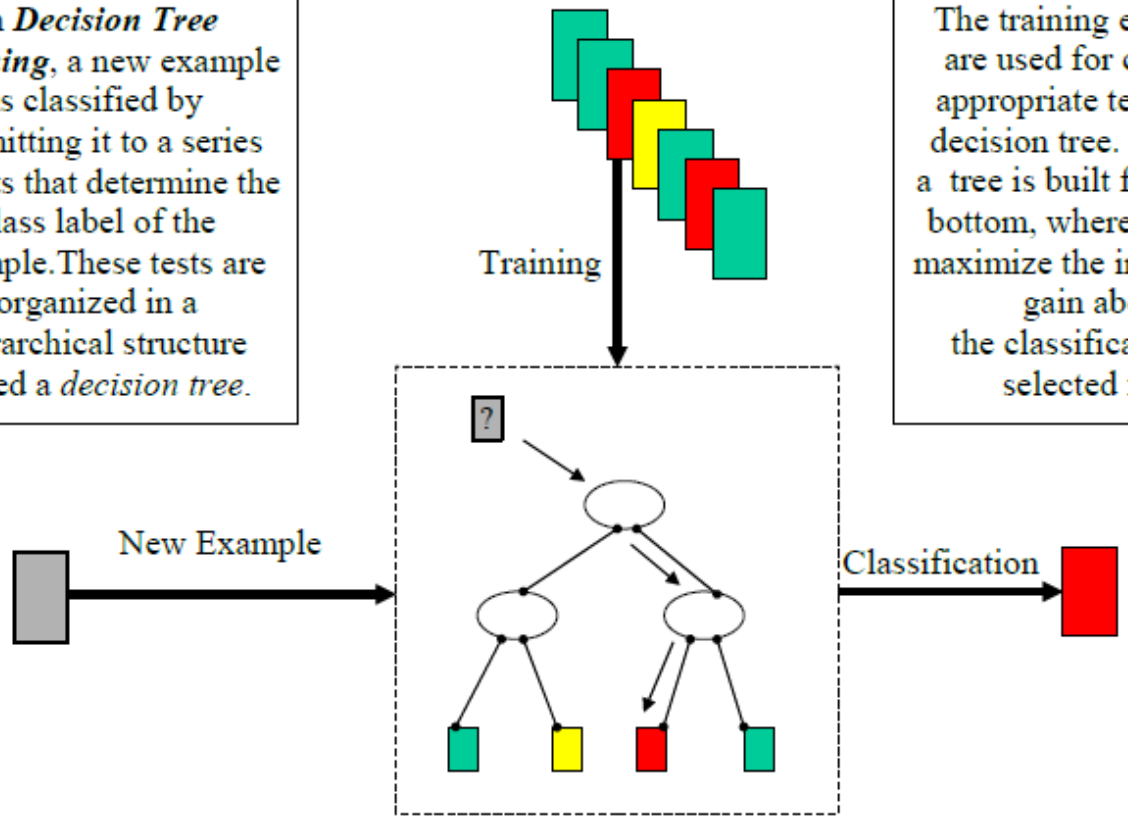
- a decision tree consists of
  - **Nodes:**
    - test for the value of a certain attribute
  - **Edges:**
    - correspond to the outcome of a test
    - connect to the next node or leaf
  - **Leaves:**
    - terminal nodes that predict the outcome

to classifiy an example:

1. start at the root
2. perform the test
3. follow the edge corresponding to outcome
4. goto 2. unless leaf
5. predict that outcome associated with the leaf

Decision-Tree Learning

In ***Decision Tree Learning***, a new example is classified by submitting it to a series of tests that determine the class label of the example. These tests are organized in a hierarchical structure called a *decision tree*.

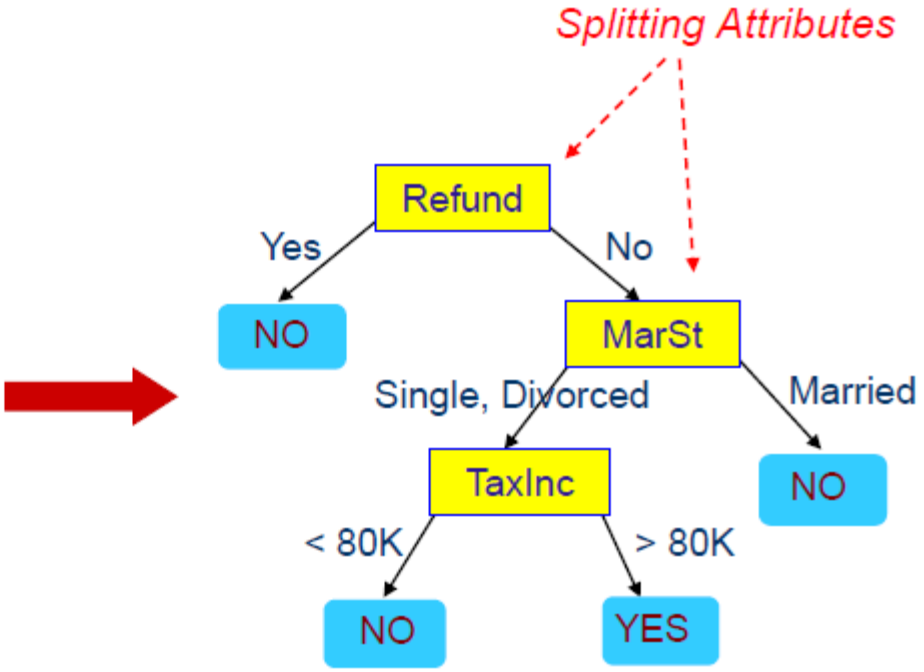


The training examples are used for choosing appropriate tests in the decision tree. Typically, a tree is built from top to bottom, where tests that maximize the information gain about the classification are selected first.

Example of a Decision Tree

Tid	Refund	Marital Status	Taxable Income	Cheat
	categorical	categorical	continuous	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	No	Single	90K	Yes

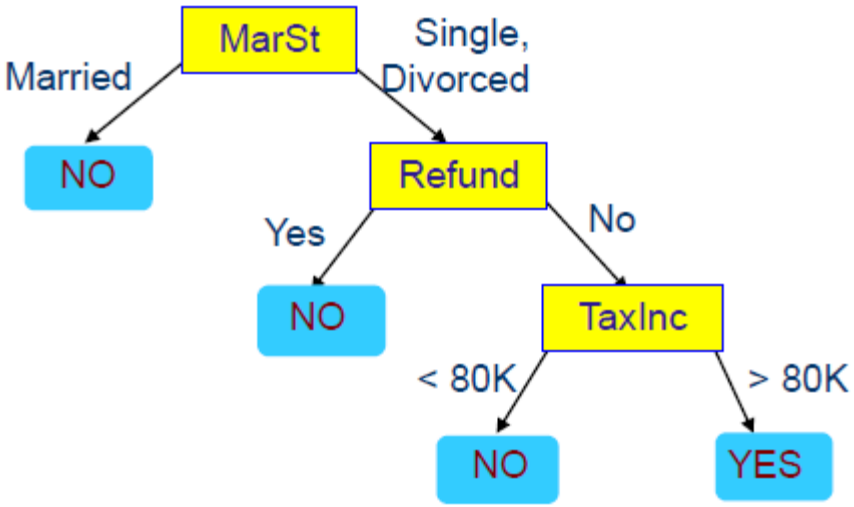
Training Data



Model: Decision Tree

Another Example of Decision Tree

Tid	Refund	Marital Status	Taxable Income	Cheat
	categorical	categorical	continuous	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	No	Single	90K	Yes



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

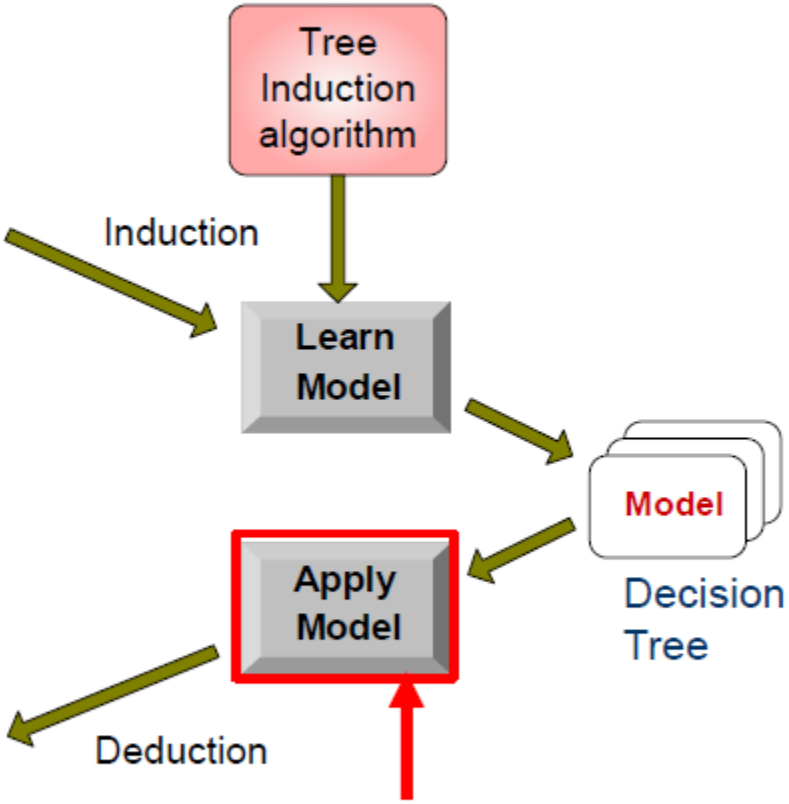
Decision Tree Classification Task

Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Training Set

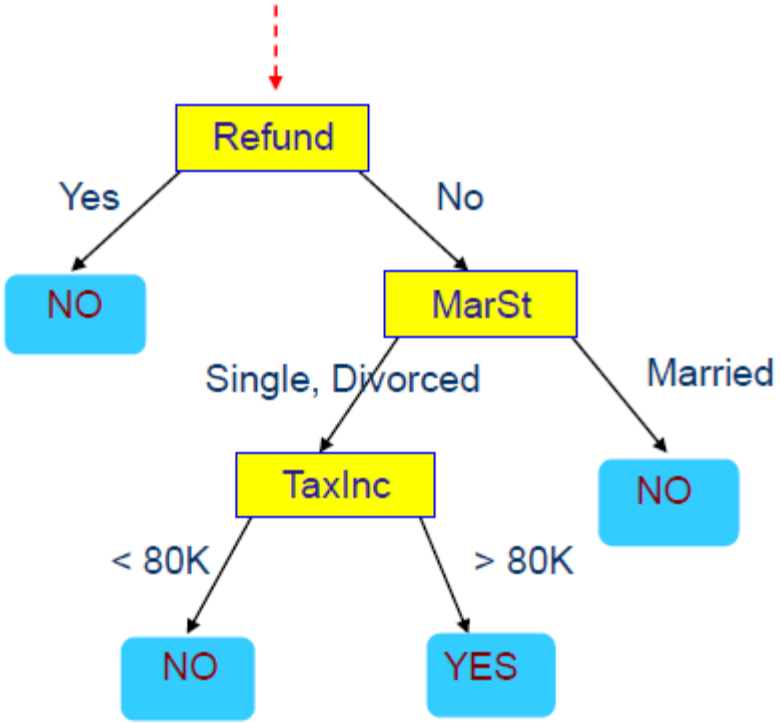
Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Test Set



Apply Model to Test Data

Start from the root of tree.



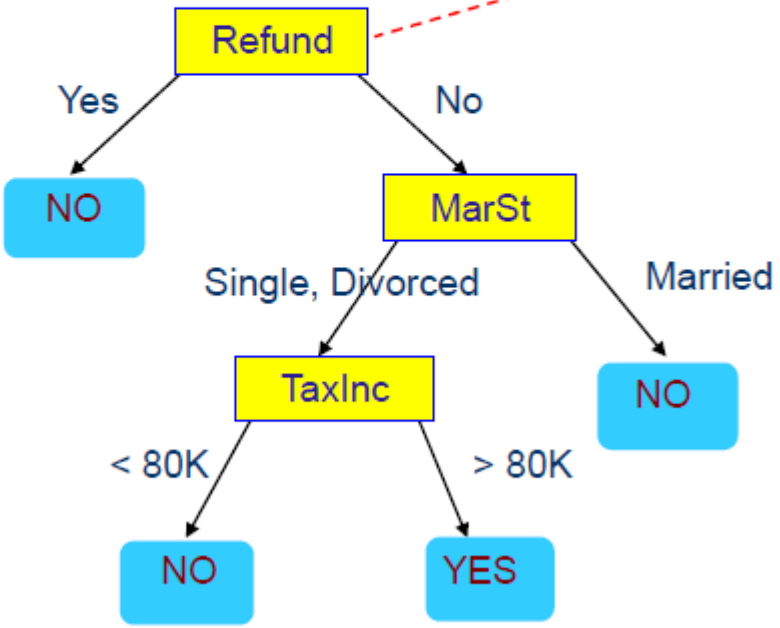
Test Data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

Apply Model to Test Data

Test Data

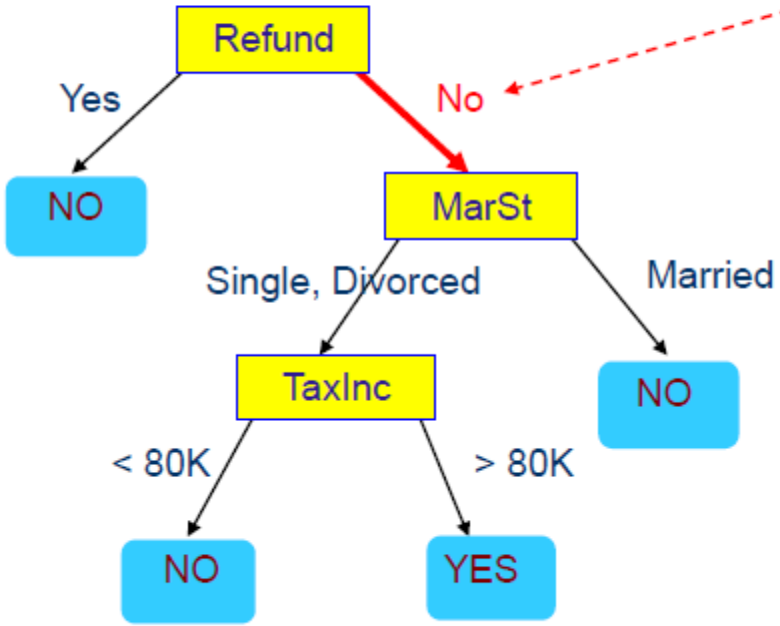
Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?



Apply Model to Test Data

Test Data

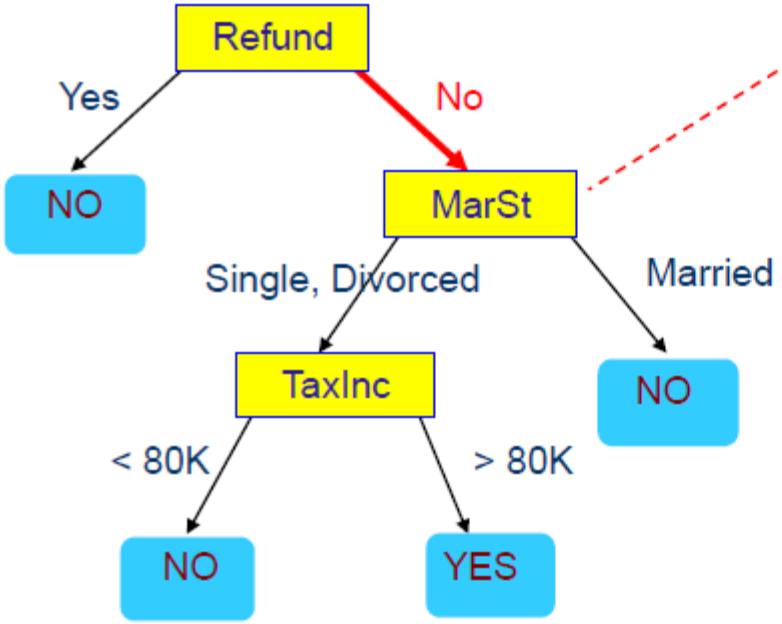
Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?



Apply Model to Test Data

Test Data

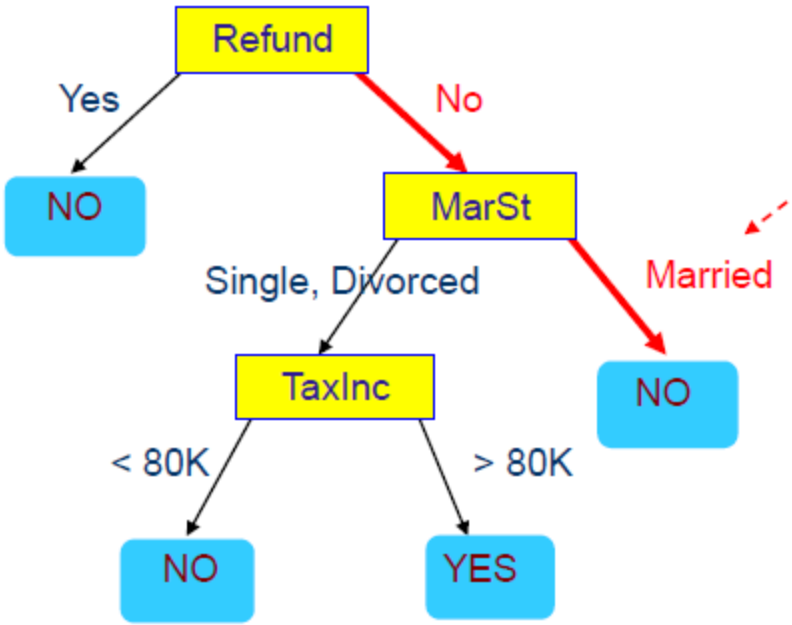
Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?



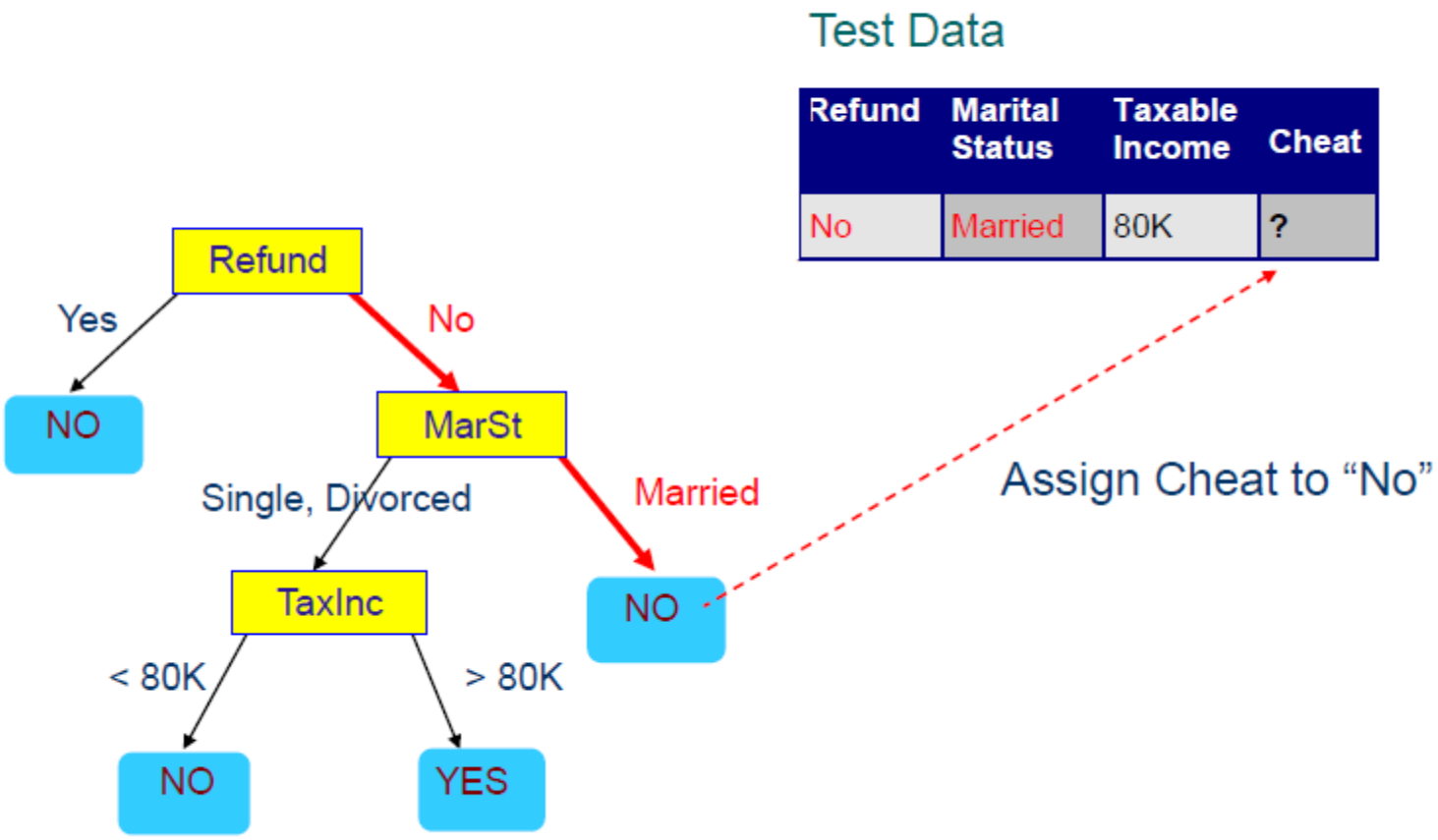
Apply Model to Test Data

Test Data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?



Apply Model to Test Data



Decision Trees Functions:

- Data compression
- Prediction

DECISION TREE

- An internal node is a test on an attribute
- A branch represents an outcome of the test, e.g., COLOR = Red
- A leaf node represents a class label or class label distribution
- At each node, one attribute is chosen to split training examples into distinct classes as much as possible
- A new case is classified by following a matching path to a leaf node
- The no. of leaf nodes represent the no of branches.
- Each branch corresponds to a *classification rule.*

Decision Tree Based Classification

- Advantages:
  - –Inexpensive to construct
  - –Extremely fast at classifying unknown records
  - –Easy to interpret for small-sized trees
  - –Decision trees provide a clear indication of which features are most important for classification.
  - –Decision trees perform classification without requiring much computation.
  - –Accuracy is comparable to other classification techniques for many simple data sets

The TDIDT Algorithm (*Top-Down Induction of Decision Trees*)

- Many Algorithms:
  - –Hunt's Algorithm (one of the earliest)
  - –CART
  - –ID3 (Iterative Dichotomiser 3), C4.5, C5.0
  - –SLIQ,SPRINT

Divide-And-Conquer Algorithms

- Family of decision tree learning algorithms
  - TDIDT: Top-Down Induction of Decision Trees
- Learn trees in a Top-Down fashion:
  - divide the problem in subproblems
  - solve each problem

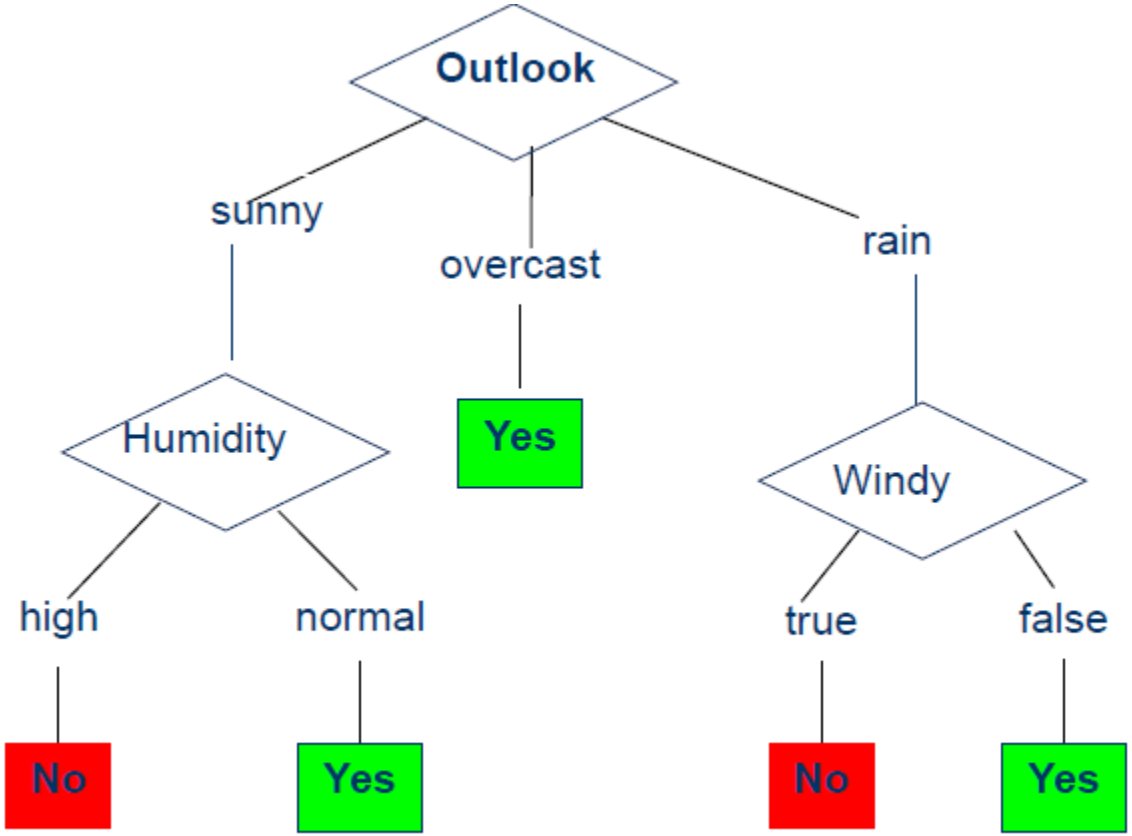
Basic Divide-And-Conquer Algorithm:

1. select a test for root node
   Create branch for each possible outcome of the test
2. split instances into subsets
   One for each branch extending from the node
3. repeat recursively for each branch, using only instances that reach the branch
4. stop recursion for a branch if all its instances have the same class

Weather Data: Play or not Play?Outlook

Outlook	Temperature	Humidity	Windy	Play?
sunny	hot	high	false	No
sunny	hot	high	true	No
overcast	hot	high	false	Yes
rain	mild	high	false	Yes
rain	cool	normal	false	Yes
rain	cool	normal	true	No
overcast	cool	normal	true	Yes
sunny	mild	high	false	No
sunny	cool	normal	false	Yes
rain	mild	normal	false	Yes
sunny	mild	normal	true	Yes
overcast	mild	high	true	Yes
overcast	hot	normal	false	Yes
rain	mild	high	true	No

Example Tree for “Play?”



Building Decision Tree

- **Top-down tree construction**

  –At start, all training examples are at the root

  –Partition the examples recursively by choosing one attribute each time

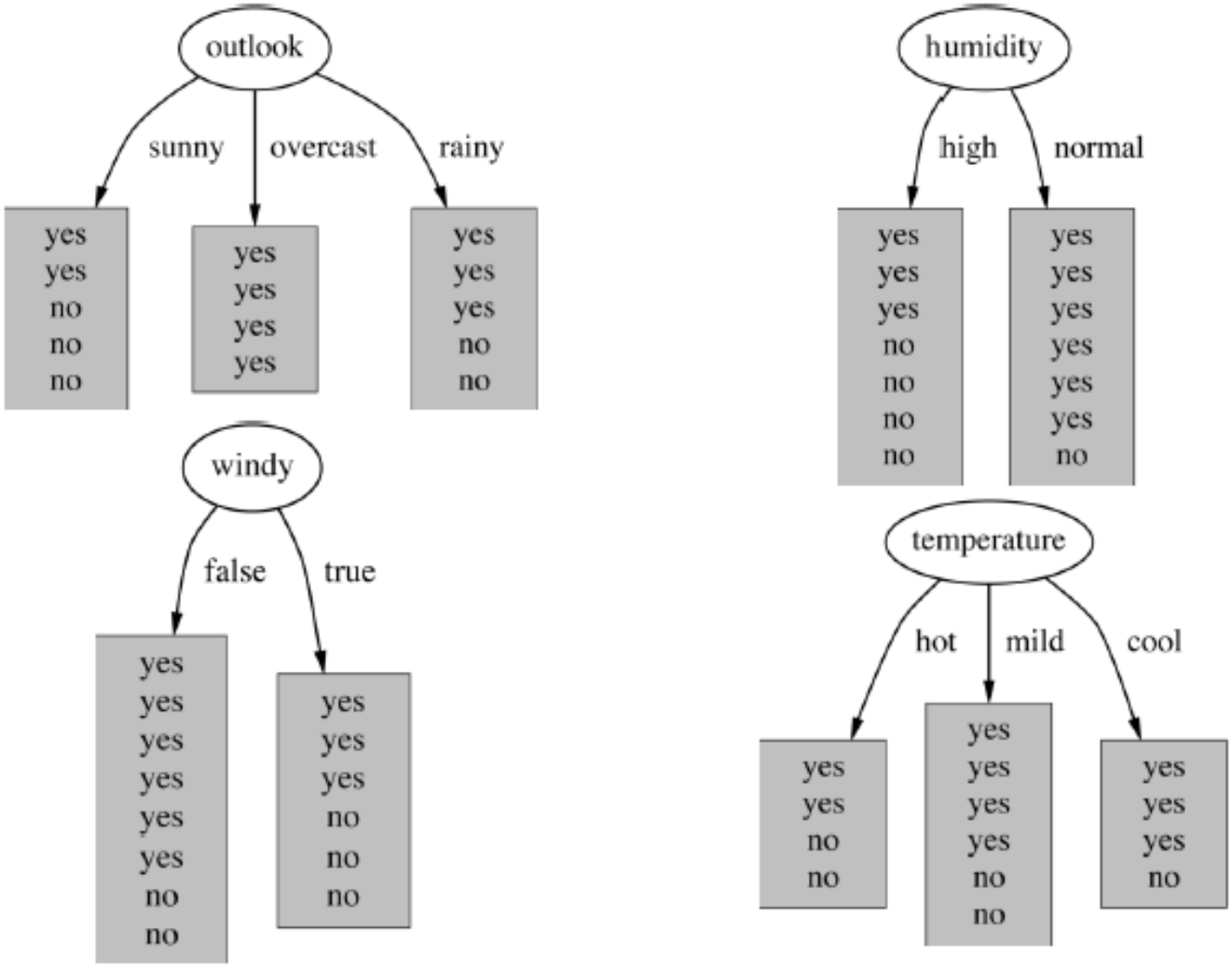
- **Bottom-up tree pruning**

  –Remove subtrees or branches, in a bottom-up manner, to improve the estimated accuracy on new

Choosing the Splitting Attribute

- At each node, available attributes are evaluated on the basis of separating the classes of the training examples. A Goodness function is used for this purpose
- **Typical goodness functions:**
  - –information gain (ID3/C4.5)
  - –accuracy
  - –giniindex
  - –others (information gain ratio)

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” nodes
- **Popular *impurity criterion: information gain***
  - –Information gain increases with the average purity of the subsets that an attribute produces
- **Strategy:** choose attribute that results in greatest information gain

Computing information

- Information is measured 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 (this can involve fractions of bits!)
- Formula for computing the entropy:

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

Example: attribute “Outlook”

- “Outlook” = “Sunny”:

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

- “Outlook” = “Overcast”:

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

*Note: log(0) is not defined, but we evaluate 0*log(0) as zero*

- “Outlook” = “Rainy”:

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

- Expected information for attribute:

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

$$= 0.693\,\text{bits}$$

Computing the information gain

- Information gain:

(information before split) – (information after split)

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

- Information gain for attributes from weather data:

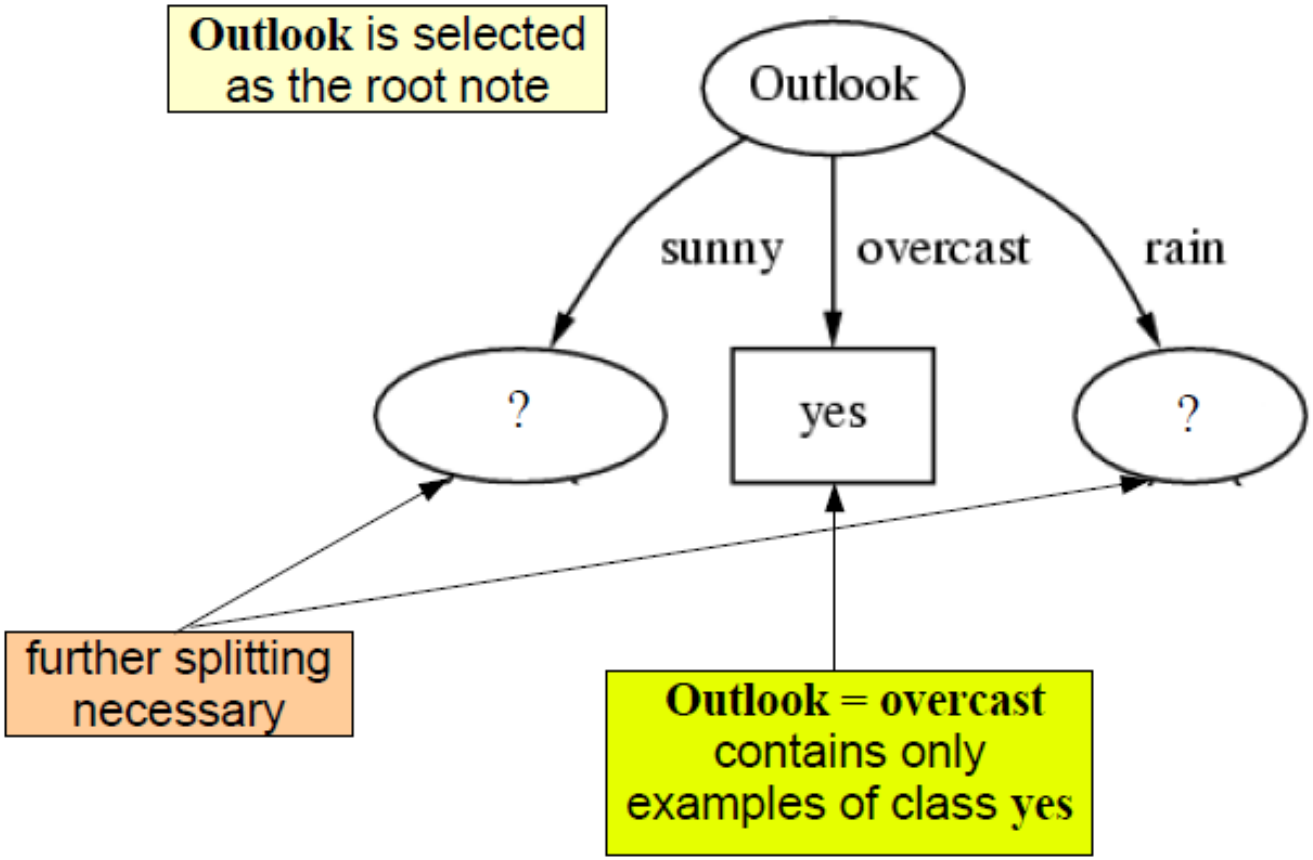
$$\text{gain("Outlook")} = 0.247 \text{ bits}$$

$$\text{gain("Temperature")} = 0.029 \text{ bits}$$

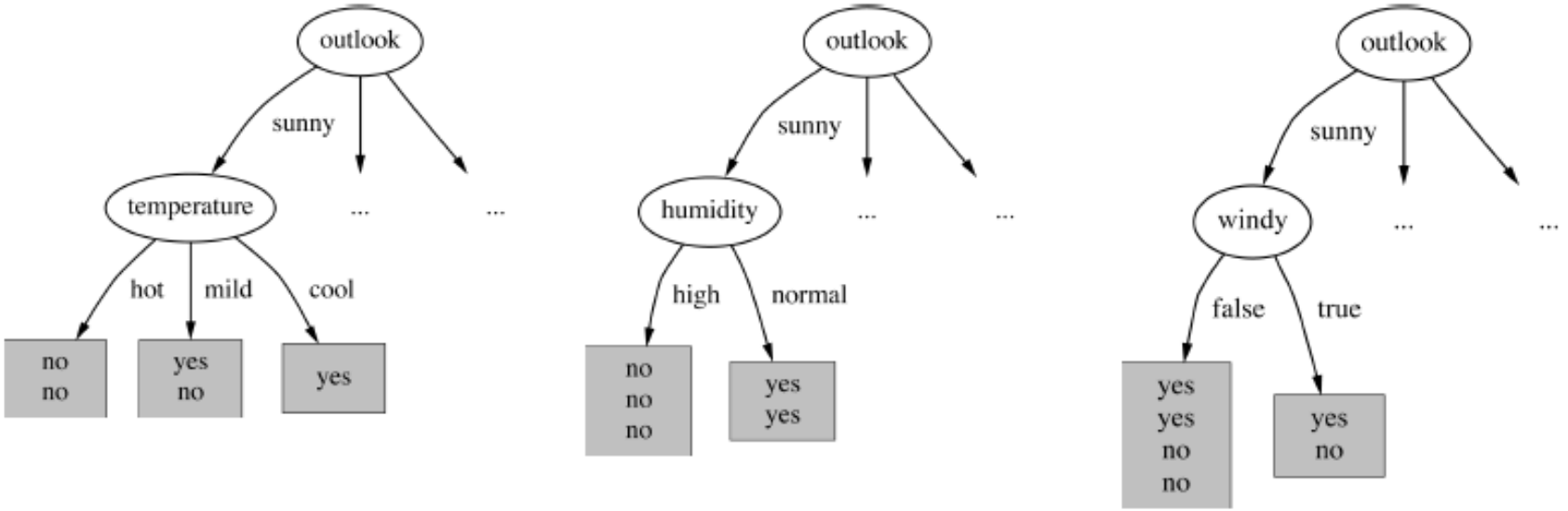
$$\text{gain("Humidity")} = 0.152 \text{ bits}$$

$$\text{gain("Windy")} = 0.048 \text{ bits}$$

Cont'd


Outlook is selected as the root note
Outlook
sunny
overcast
rain
?
yes
?
further splitting necessary
Outlook = overcast contains only examples of class yes

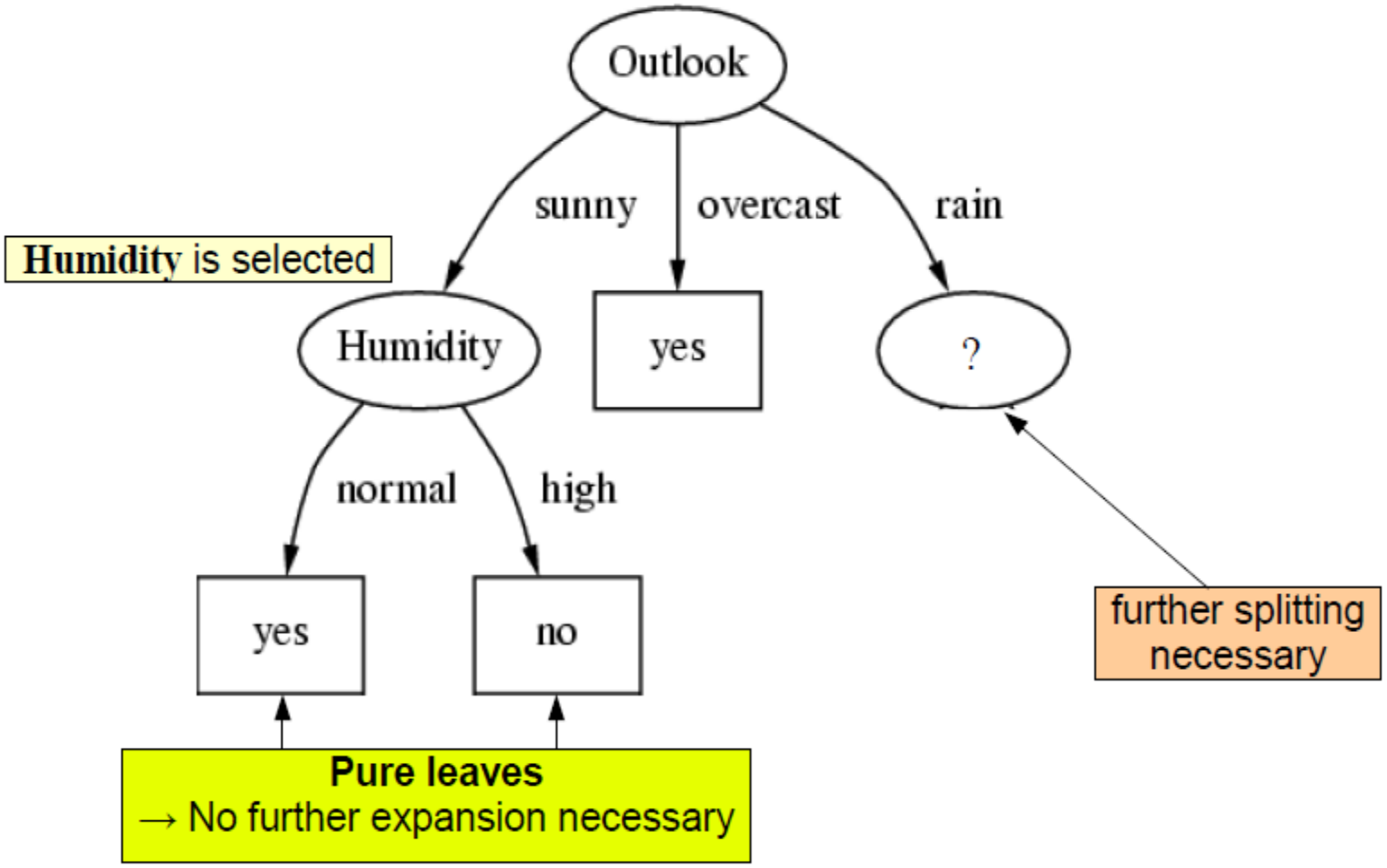
Continuing to split



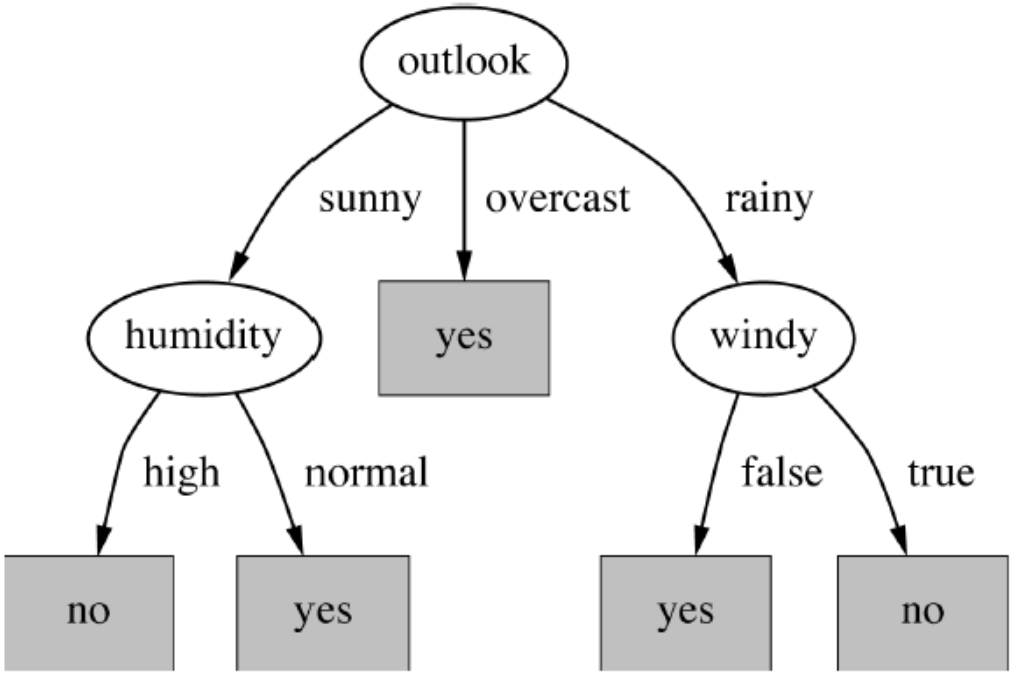
$$\begin{aligned}
&\text{Gain}(\textit{Temperature}) &&= 0.571 \text{ bits} \\
&\text{Gain}(\textit{Humidity}) &&= 0.971 \text{ bits} \\
&\text{Gain}(\textit{Windy}) &&= 0.020 \text{ bits}
\end{aligned}$$

Humidity is selected

Continuing to split



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

*CART Splitting Criteria: Gini Index

- Many alternative measures to Information Gain
- Most popular alternative: Gini index
  - used in e.g., in CART (Classification And Regression Trees)
  - impurity measure (instead of entropy)

$$Gini(S) = 1 - \sum_i p_i^2$$

  - average Gini index (instead of average entropy / information)

$$Gini(S, A) = \sum_i \frac{|S_i|}{|S|} \cdot Gini(S_i)$$

  - Gini Gain
    - could be defined analogously to information gain
    - but typically avg. Gini index is minimized instead of maximizing Gini gain

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

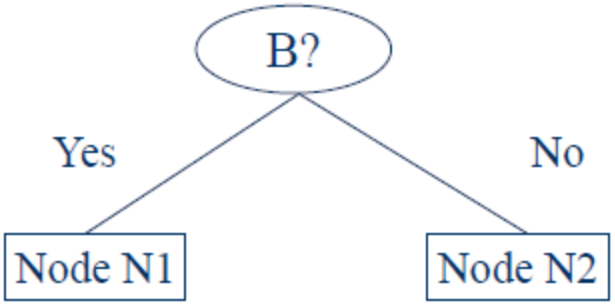
C1	**2**
C2	**4**

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

Gini = 1 – $(2/6)^2$ – $(4/6)^2$ = 0.444

Binary Attributes: Computing GINI Index

- Splits into two partitions
- Effect of Weighing partitions:
  - Larger and Purer Partitions are sought for.



	Parent
C1	6
C2	6
Gini = 0.500	

Gini(N1)
= 1 – $(5/7)^2$ – $(2/7)^2$
= 0.408

Gini(N2)
= 1 – $(1/5)^2$ – $(4/5)^2$
= 0.32

	N1	N2
C1	5	1
C2	2	4

Gini(Children)
= 7/12 * 0.408 +
5/12 * 0.32
= 0.371

Categorical Attributes: Computing Gini Index

- For each distinct value, gather counts for each class in the dataset
- Use the count matrix to make decisions

Multi-way split

	CarType		
	Family	Sports	Luxury
C1	1	2	1
C2	4	1	1
Gini	0.393		

Two-way split
(find best partition of values)

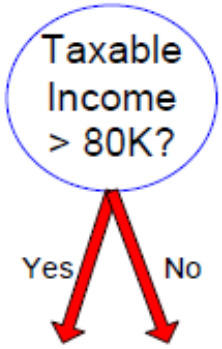
	CarType	
	{Sports, Luxury}	{Family}
C1	3	1
C2	2	4
Gini	0.400	

	CarType	
	{Sports}	{Family, Luxury}
C1	2	2
C2	1	5
Gini	0.419	

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 ≥ 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

Cheat	No		No		No		Yes		Yes		Yes		No		No		No		No	
Taxable Income																				
Sorted Values →	60		70		75		85		90		95		100		120		125		220	
Split Positions →	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	

Splitting Criteria based on Classification Error

- Classification error at a node t :

$$Error(t) = 1 - \max_i P(i \mid t)$$

- Measures misclassification error made by a node.
  - 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

Examples for Computing Error

$$Error(t) = 1 - \max_i P(i \mid t)$$

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

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

Error = 1 – max (0, 1) = 1 – 1 = 0

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

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

Error = 1 – max (1/6, 5/6) = 1 – 5/6 = 1/6

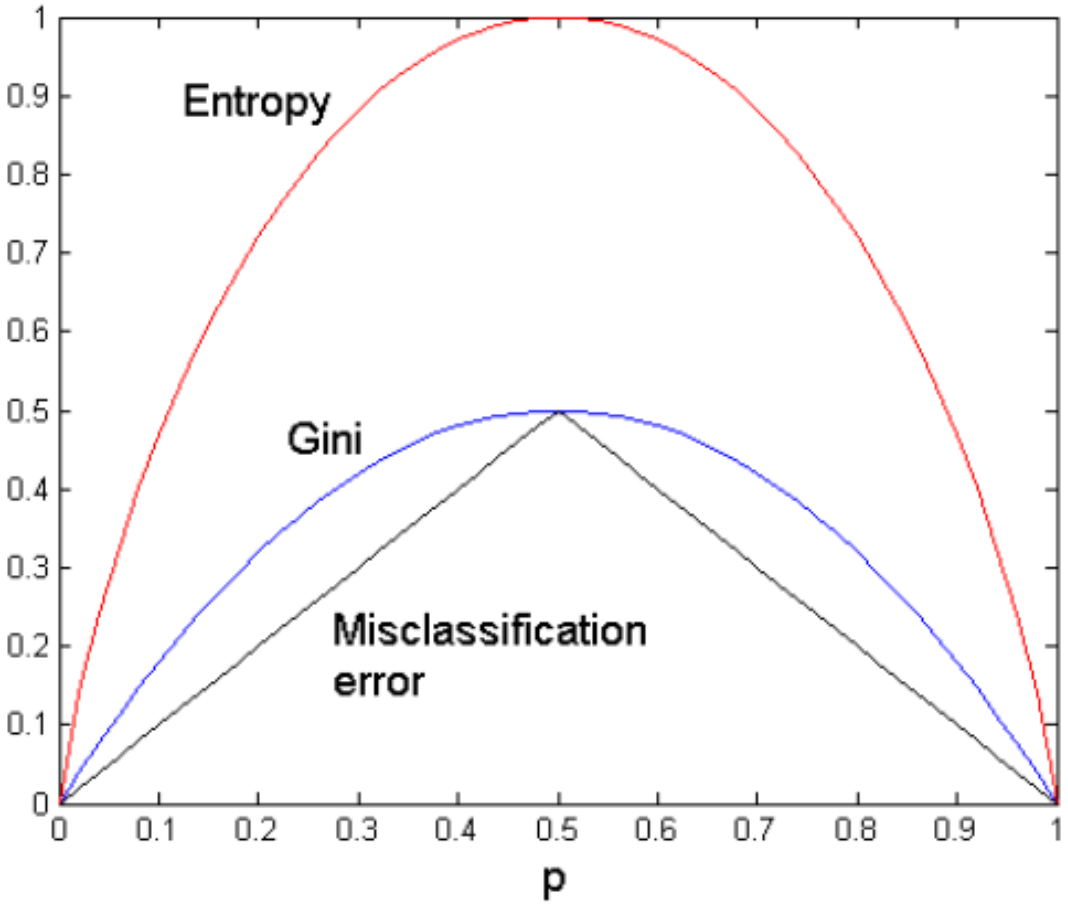
C1	**2**
C2	**4**

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

Error = 1 – max (2/6, 4/6) = 1 – 4/6 = 1/3

Comparison among Splitting Criteria

For a 2-class problem:



Missing values

- Examples are classified as usual
  - if we are lucky, attributes with missing values are not tested by the tree
- If an attribute with a missing value needs to be tested:
  - split the instance into fractional instances (*pieces*)
  - one piece for each outgoing branch of the node
  - a piece going down a branch receives a weight proportional to the popularity of the branch
  - weights sum to 1
- Info gain or gain ratio work with fractional instances
  - use sums of weights instead of counts
- during classification, split the instance in the same way
  - Merge probability distribution using weights of fractional instances

Overfitting and Pruning

- The smaller the complexity of a concept, the less danger that it overfits the data
  - A polynomial of degree n can always fit $n+1$ points
- Thus, learning algorithms try to keep the learned concepts simple
  - Note a „perfect“ fit on the training data can always be found for a decision tree! (except when data are contradictory)

Pre-Pruning:

- stop growing a branch when information becomes unreliable

Post-Pruning:

- grow a decision tree that correctly classifies all training data
- simplify it later by replacing some nodes with leafs

- 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
- Most popular test: *chi-squared test*
- 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

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

→ In a dataset that contains XOR attributes a and b, and several irrelevant (e.g., random) attributes, ID3 can not distinguish between relevant and irrelevant attributes

→ Prepruning won't expand the root node

  - Structure is only visible in fully expanded tree
- But:
  - XOR-type problems rare in practice
  - prepruning is faster than postpruning

Post-Pruning

- basic idea
  - first grow a full tree to capture all possible attribute interactions
  - later remove those that are due to chance

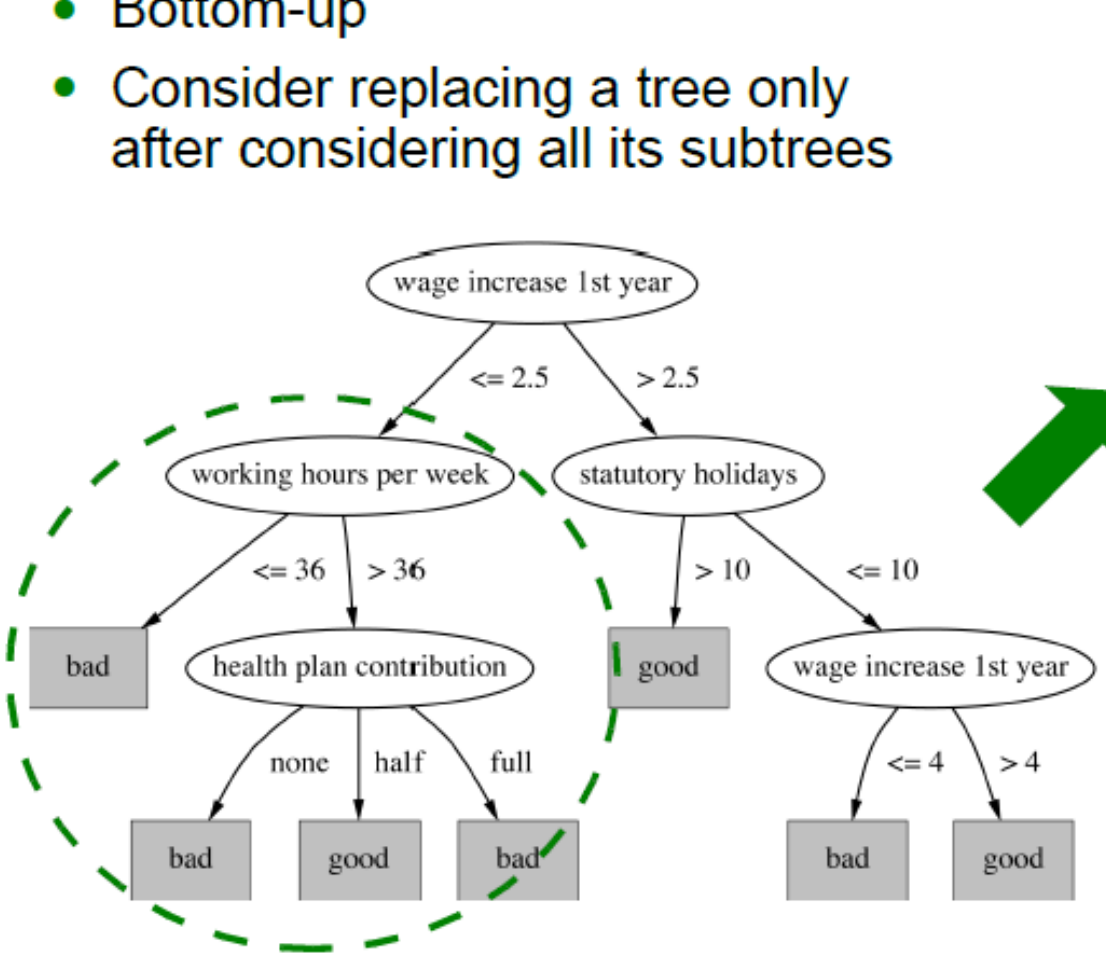
1. learn a complete and consistent decision tree that classifies all examples in the training set correctly
2. as long as the performance increases
   - try simplification operators on the tree
   - evaluate the resulting trees
   - make the replacement the results in the best estimated performance
3. return the resulting decision tree

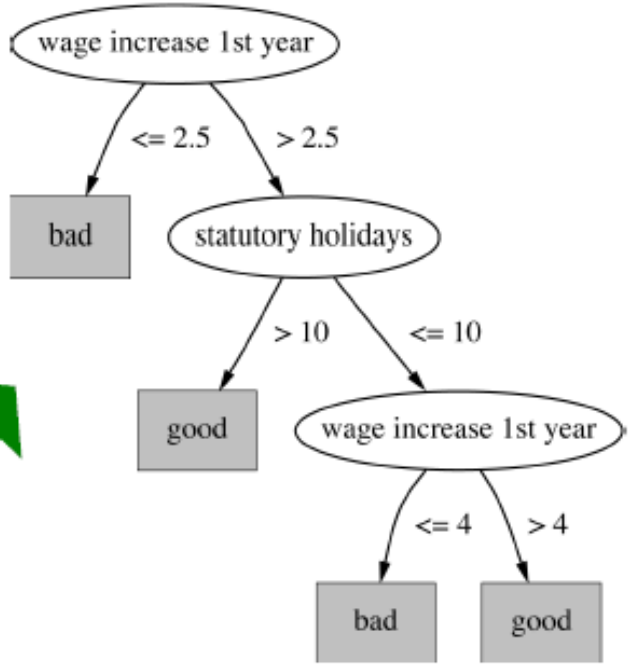
Postpruning

- Two subtree simplification operators
  - Subtree replacement
  - Subtree raising
- Possible performance evaluation strategies
  - error estimation
    - on separate pruning set („reduced error pruning“)
    - with confidence intervals (C4.5's method)
  - significance testing
  - MDL principle

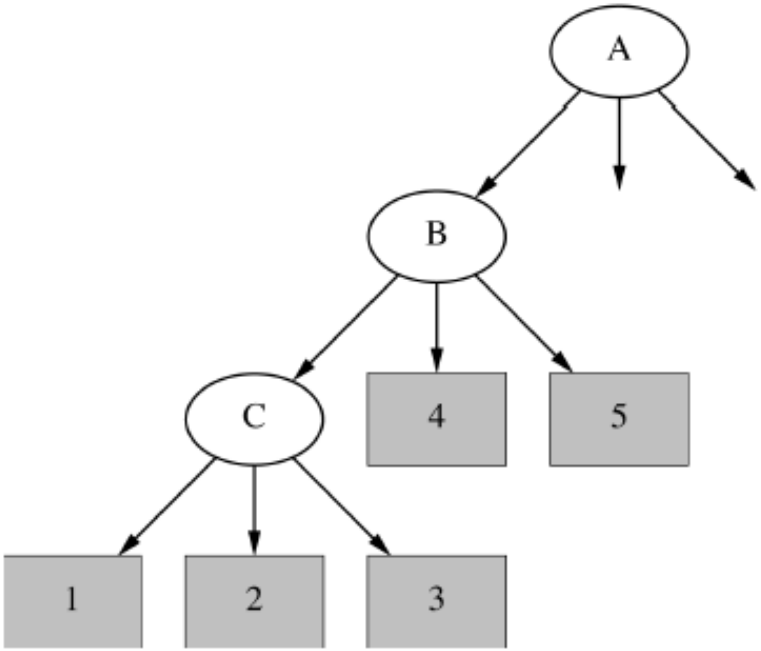
Subtree replacement

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

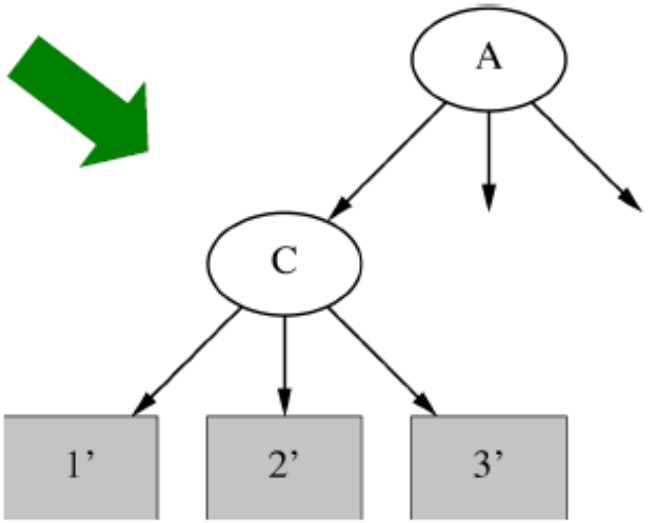




Subtree raising



- Delete node B
- Redistribute instances of leaves 4 and 5 into C



Regression Problems

- Regression Task
  - the target variable is numerical instead of discrete
- Two principal approaches
  - Discretize the numerical target variable
    - e.g., equal-with intervals, or equal-frequency
    - and use a classification learning algorithm
  - Adapt the classification algorithm to regression data

  → Regression Trees and Model Trees

Regression Problems

Differences to Decision Trees (Classification Trees)

- Leaf Nodes:
  - Predict the average value of all instances in this leaf
- Splitting criterion:
  - Minimize the variance of the values in each subset S_i
  - Standard deviation reduction

$$SDR(A, S) = SD(S) - \sum_i \frac{|S_i|}{|S|} SD(S_i)$$

- Termination criteria:

  Very important! (otherwise only single points in each leaf)
  - lower bound on standard deviation in a node
  - lower bound on number of examples in a node
- Pruning criterion:
  - Numeric error measures, e.g. Mean-Squared Error

Model Trees

- In a Leaf node
  - Classification Trees predict a class value
  - Regression Trees predict the average value of all instances in the model
  - Model Trees use a linear model for making the predictions
    - growing of the tree is as with Regression Trees
- Linear Model:
  - $LM(x) = \sum w_i v_i(x)$ where $v_i(x)$ is the value of attribute A_i for example x and w_i is a weight
  - The attributes that have been used in the path of the tree can be ignored
- Weights can be fitted with standard math packages
  - Minimize the Mean Squared Error $MSE = \sum_j (y_j - r_j)^2$

Stopping Criteria for Tree Induction

- Stop expanding a node when all the records belong to the same class
- Stop expanding a node when all the records have similar attribute values
- Early termination

Summary

- Classification Problems require the prediction of a discrete target value
  - can be solved using decision tree learning
  - iteratively select the best attribute and split up the values according to this attribute
- Regression Problems require the prediction of a numerical target value
  - can be solved with regression trees and model trees
  - difference is in the models that are used at the leafs
  - are grown like decision trees, but with different splitting criteria
- Overfitting is a serious problem!
  - simpler, seemingly less accurate trees are often preferable
  - evaluation has to be done on separate test sets