

# Decision Trees

Yaji Sripada

Department of Computing Science

University of Aberdeen

# The Task

- Input: Collection of instances with a set of attributes  $x$  and a special nominal attribute  $Y$  called class attribute
- Output: A model that accurately predicts  $y$  from  $x$  on previously unseen instances
  - Previously unseen instances are called test set
- Usually, input collection is divided into
  - Training set for building the required model
  - Test set for evaluating the model built

# Example Data

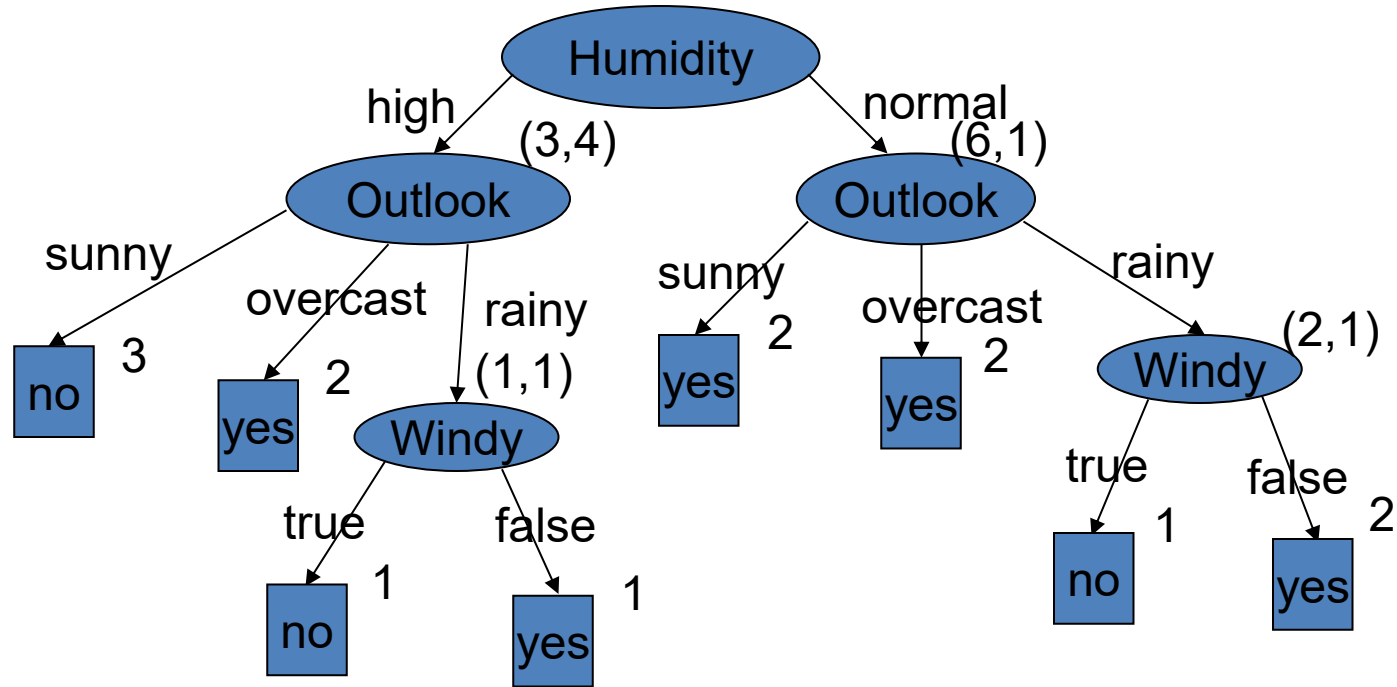
Class  
Attribute

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
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

# Decision Tree Construction

- Recursive procedure
  - Select an attribute to place at the root node
  - Make one branch for each possible value of the selected attribute
  - For each branch repeat the above two steps recursively
    - Using only those instances that actually reach the branch
  - Stop developing a branch if it has instances belonging to the same class
- Several decision trees are possible
  - Based on the order of the selection of the attributes

# Example Decision Tree 1



Observations:

Outlook and windy repeated  
in this tree

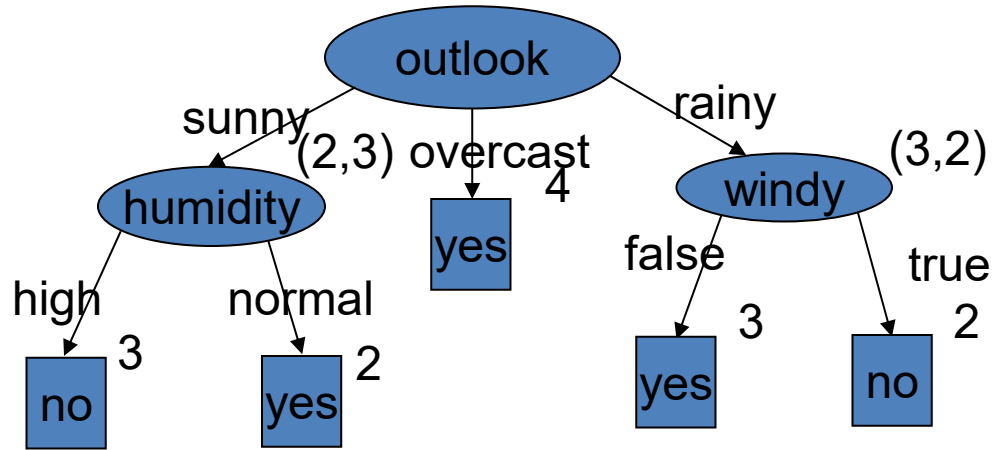
Windy tested only when  
outlook is rainy

Actions:

Start the tree with  
Outlook

Test windy when  
outlook is rainy

## Example Decision Tree 2



# Occam's Razor

- Principle stated by William of Ockham
  - “Other things being equal, simple theories are preferable to complex ones”
  - Informally,
    - “Keep it simple, stupid!!”
- This has been the guiding principle for developing scientific theories
- Applied to our two decision trees describing the weather data
  - Decision tree 2 is preferable to decision tree 1
- Small decision trees are better
  - Attribute ordering makes the difference

# Attribute Selection for Splitting

- In our case, outlook is a better attribute at the root than others
  - Because when outlook is at the root, one of its branches (overcast) immediately leads to a 'pure' daughter node, which terminates further tree growth
  - Pure nodes have the same class label for all the instances reaching that node
- We need a generic function that helps to rank-order attributes
  - The function should reward attributes that produce 'pure' daughter nodes

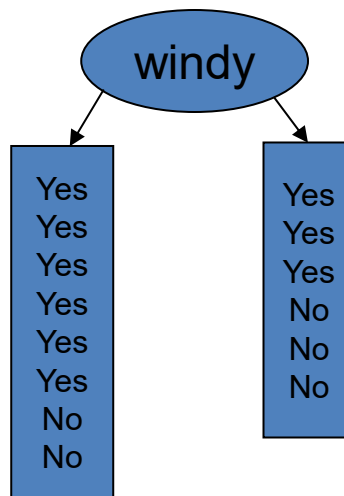
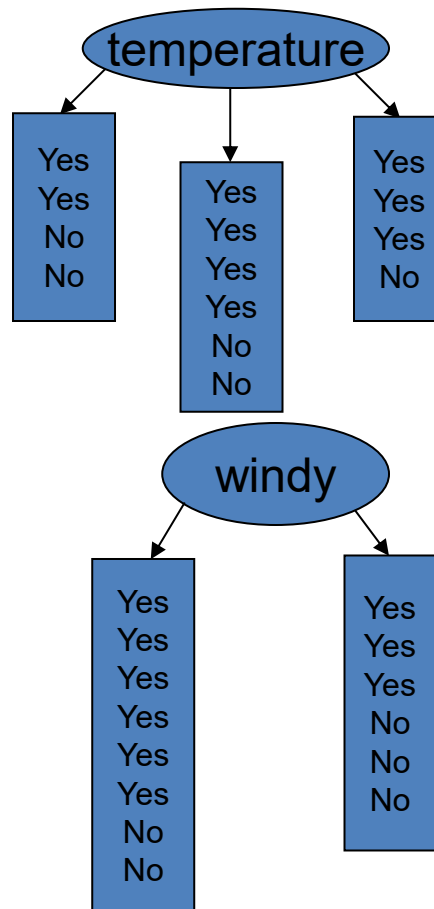
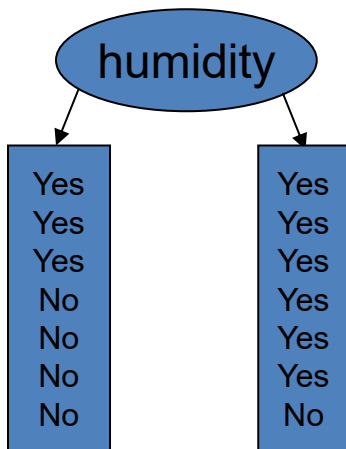
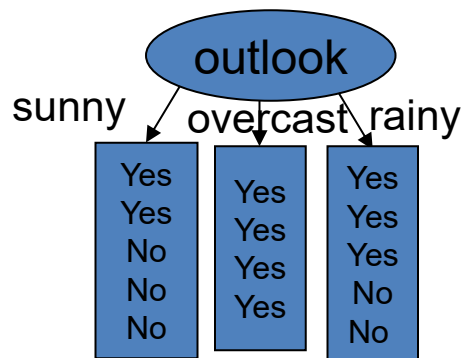
# Entropy

- Entropy is a measure of purity expressed in bits
- For a node in the tree, it represents the expected amount of information that would be needed to specify the class of a new instance that reached the node
- $\text{Entropy}(p_1, p_2, \dots, p_n) = -p_1 \log p_1 - p_2 \log p_2 \dots - p_n \log p_n$
- Logarithms are computed for base 2 because we want the entropy measure in bits
- $p_1, p_2, \dots, p_n$  are fractions that sum up to 1
- Because logarithms of fractions are negative, minus signs are used in the above formula to keep the entropy measure positive

## Entropy (2)

- For the weather data
  - $p_1$  = fraction of instances with play is true = 9/14
  - $p_2$  = fraction of instances with play is false = 5/14
  - $\text{entropy}(9/14, 5/14) = -9/14 \log 9/14 - 5/14 \log 5/14$   
 $= -9/14(\log 9 - \log 14) - 5/14(\log 5 - \log 14)$   
 $= -9/14 \log 9 + 9/14 \log 14 - 5/14 \log 5 + 5/14 \log 14$   
 $= -9/14 \log 9 - 5/14 \log 5 + 14/14 \log 14 = (-9 \log 9 - 5 \log 5 + 14 \log 14)/14 = 0.940$   
bits (fractions of bits allowed!!)

# Tree stumps for the weather data



## Entropy for the outlook stump

- Count the numbers of yes and no classes at the leaf nodes
  - [2,3], [4,0] and [3,2]
- Compute the entropy for each branch of the stump
  - $\text{Entropy}(2/5, 3/5) = 0.971$  bits
  - $\text{Entropy}(4/4, 0/4) = 0.0$  bits
  - $\text{Entropy}(3/5, 2/5) = 0.971$  bits

## Entropy for the outlook stump

- Compute the entropy for the whole stump
  - $\text{Entropy}([2,3],[4,0],[3,2]) = \begin{aligned} &5/14 * \text{Entropy}(2/5,3/5) + \\ &4/14 * \text{Entropy}(4/4,0/4) + \\ &5/14 * \text{Entropy}(3/5,2/5) \end{aligned}$   
 $= 5/14 * 0.971 + 4/14 * 0 + 5/14 * 0.971$   
 $= 0.693 \text{ bits}$
- Represents the information needed in bits to specify the class for a new instance using this stump

# Information Gain

- When the attribute outlook was not considered, the weather data set had an entropy of 0.940 bits (as computed on slide 10)
- Entropy for the outlook stub is 0.693 bits (as computed on the previous slide)
- We made a saving of  $(0.940 - 0.693)$  bits on the information needed to specify the class of a new instance using this stump
  - This is the informational value of creating the outlook node
  - $\text{Gain}(\text{outlook}) = (0.940 - 0.693) \text{ bits} = 0.247 \text{ bits}$

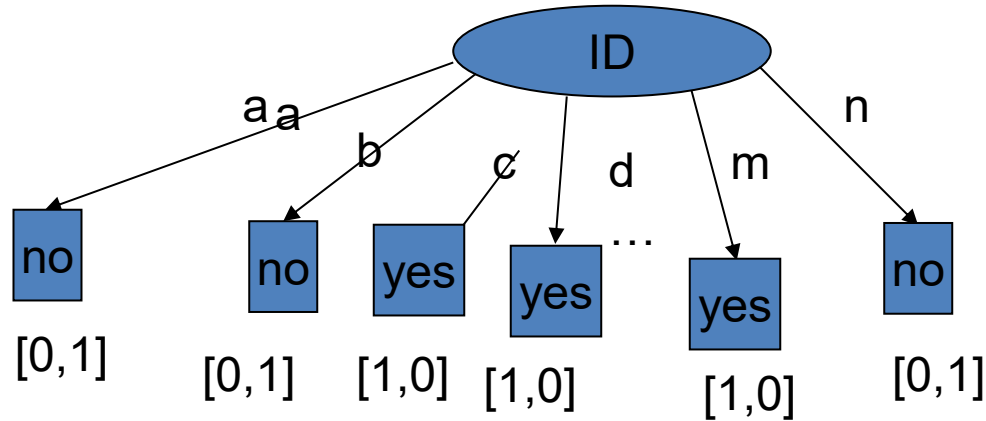
# Information Gain

- Similar computations for other stumps give us
  - $\text{Gain}(\text{temperature}) = 0.029$  bits
  - $\text{Gain}(\text{humidity}) = 0.152$  bits
  - $\text{Gain}(\text{windy}) = 0.048$
- Because information gain is maximum for outlook, it should be selected for the root node
- Continuing with the above procedure builds the example decision tree 2

# Weather Data with ID

ID	Outlook	Temperature	Humidity	Windy	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 the ID attribute



$$\begin{aligned} &\text{Entropy}([0,1],[0,1],[1,0],[1,0] \dots [1,0],[0,1]) = \\ &1/14 * \text{Entropy}(0/1,1/1) + 1/14 * \text{Entropy}(0/1,1/1) + \\ &1/14 * \text{Entropy}(1/1,0/1) + 1/14 * \text{Entropy}(1/1,0/1) + \dots + \\ &1/14 * \text{Entropy}(1/1,0/1) + 1/14 * \text{Entropy}(0/1,1/1) = \\ &1/14 * 0 + 1/14 * 0 + 1/14 * 0 \dots 1/14 * 0 = 0 \end{aligned}$$

$$\text{Gain}(\text{ID}) = 0.940 - 0 = 0.940 \text{ bits}$$

# Highly branching attributes

- Weather data with ID has different id values for each instance
  - Knowing the id value is enough to predict the class
  - Therefore, entropy for the stump with this attribute would be zero
    - » Gain is high (0.940) and therefore this attribute will be selected first for tree construction
  - The tree constructed would be useless
    - » Cannot predict new instances
    - » Tells nothing about the structure of the decision
- The above situation arises whenever an attribute leads to high branching
- A split always results in entropy reduction
  - Because a split reduces the number of classes
- Information gain does not account for this intrinsic information of a split

# Gain Ratio

- Gain ratio is a measure used to adjust for the intrinsic information of a split
- Intrinsic information of a split is computed using the number and size of the daughter nodes produced by the split
  - Without taking the class information into account

## Gain Ratio (2)

- Intrinsic Information for the ID stump
  - $\text{Entropy}([1,1,1,\dots,1]) = 14 * (-1/14 \log 1/14) = 3.807$  bits
- Gain Ratio = Information Gain / Intrinsic Information
- Gain ratio for the ID stump =  $0.940 / 3.807 = 0.247$
- Gain ratios for other stumps are
  - $\text{GainRatio}(\text{outlook}) = 0.157$ ,  $\text{GainRatio}(\text{temperature}) = 0.019$
  - $\text{GainRatio}(\text{humidity}) = 0.152$ ,  $\text{GainRatio}(\text{windy}) = 0.049$
- In this case, ID still wins but not by a huge margin
- Additional measures used to guard against such useless attributes

## Gain Ratio (3)

- Gain ratio might overcompensate for intrinsic information
  - Because humidity splits the data only into two branches, its GainRatio is nearly equal to that of outlook
- Possible to fix this by choosing an attribute
  - with high gain ratio and
  - has InfoGain equal to the average of the InfoGains for all the attributes
- Selection of attributes for splitting are based on other measures such as 'Gini'

# Numeric Attributes

- Numeric attributes can take many values
  - Creating branches for each value is not ideal
- The value range is usually split into two parts
- Splitting position is determined using the idea of information gain

## Numeric Attributes (2)

- Consider the following sorted temperature values and their class labels  
64 65 68 69 70 71 72 75 80 81 83 85  
Yes no yes yes yes no no yes no yes yes no
- We could have created a split at any of the 11 positions between the numbers
- For each split, compute the information gain
- Select the split that gives the highest information gain
- E.g.  $\text{temp} < 71.5$  produces 4 yes and 2 no &
- $\text{Temp} > 71.5$  produces 5 yes and 3 no
- $\text{Entropy}([4,2],[5,3]) = 6/14 * \text{Entropy}(4/6, 2/6) + 8/14 * \text{Entropy}(5/8, 3/8) = 0.939$  bits

# Missing Values

- An easy solution is to treat missing values as a new possible value for the attribute
  - E.g if Outlook has a missing value, we have rainy, overcast, sunny and missing as its possible values
  - This makes sense if the fact that the attribute is missing is significant (e.g. missing medical test results)
  - Also, the learning method needs no modification

## Missing Values (2)

- A more complex solution is to let the missing value receive a proportion of the value from each of the known values of the attribute
  - The proportion is estimated based on the proportions of instances with known value at a node
  - E.g. if Outlook has a missing value in one instance and has 4 instances with rainy, 2 with overcast and 3 with sunny, then the missing value is {4/9 rainy, 2/9 overcast, 3/9 sunny}
  - All the computations (such as information gain) are performed using these weighted values

# Overfitting

- When decision trees are grown until each leaf node is pure, they might learn unnecessary details from the training data
  - This is called overfitting
  - Unnecessary details could be just noise, or because the training data is not representative
- Overfitting makes the classifier perform poorly on independent test data

## Overfitting (2)

- Two solutions
  - Stop growing the tree earlier, before it overfits training data – hard to estimate when to stop in practice
  - Prune overfitted parts of the decision tree by evaluating the utility of pruning nodes from the tree using part of the training data as validation data

# Summary

- Attribute selection for splitting achieved using measures of ‘purity’
  - Information Gain
  - Gain Ratio
  - Etc.
- Issues related to decision tree construction
  - Numerical Attributes
  - Missing values
  - Overfitting and Pruning