COMP3308/3608, Lecture 7

ARTIFICIAL INTELLIGENCE

Decision Trees

Reference: Witten, Frank, Hall and Hall: ch.4.3 and ch.6.1

Russell and Norvig: p.697-707

Outline

Core topics:

- Constructing decision trees
- Entropy and information gain
- DT’s decision boundary

Additional topics:

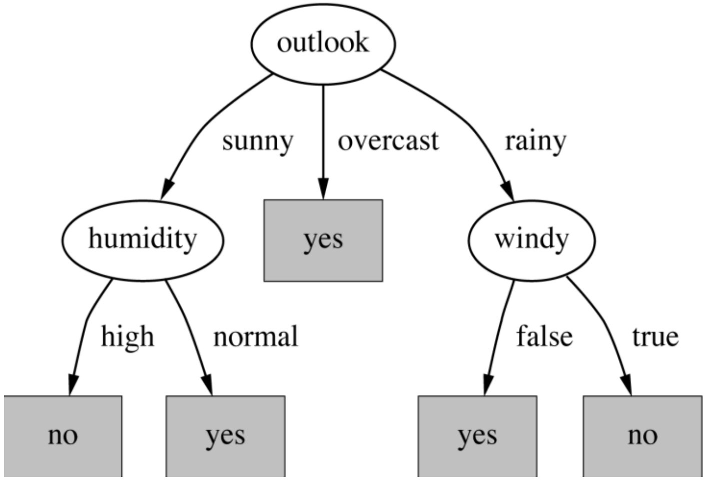
- Avoiding overfitting by pruning
- Dealing with numeric attributes
- Alternative measures for selecting attributes
- Dealing with missing values
- Handling attributes with different costs

Decision Trees (DTs)

- **DTs are supervised classifiers**
  - **Most popular and well researched ML/DM technique**
  - **Developed in parallel in ML by Ross Quinlan (USyd) and in statistics by Breiman, Friedman, Olshen and Stone “Classification and regression trees”, 1984**
- **Quinlan has refined the DT algorithm over the years**
  - **ID3, 1986**
  - **C4.5, 1993**
  - **C5.0 (See 5 on Windows) – commercial version of C4.5 used in many DM packages**
- **2 DT versions in WEKA**
  - **id3 – nominal attributes only**
  - **j48 – both nominal and numeric**
- **Many other implementations available for free and not for free**

DT Example

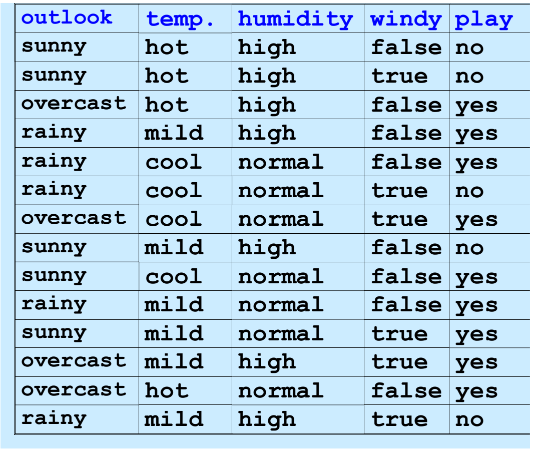
- **DT representation:**
  - **each *non-leaf node* tests the values of an attribute**
  - **each *branch* corresponds to an attribute value**
  - **each *leaf node* assigns a class**



- **To predict the class for a new example: start from the root and test the values of the attributes, until you reach a leaf node; return the class of the leaf node.**

What would be the prediction for

`outlook=sunny, temperature=cool, humidity=high, windy=true`



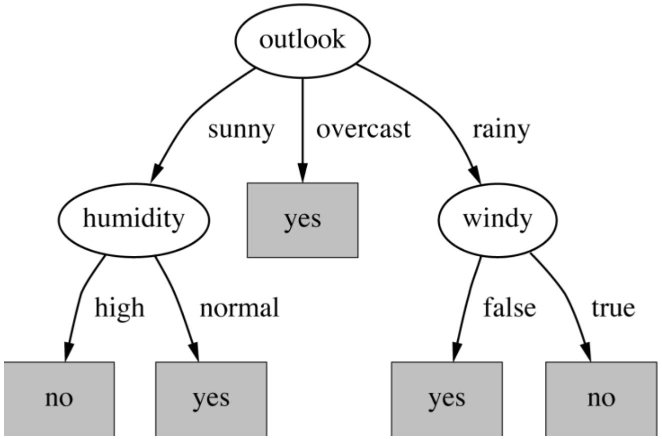
outlook	temp.	humidity	windy	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

- **Another way to look at DTs:**

Use your data to build a tree of questions with answers at the leaves. To answer a new query, start from the tree root, answer the questions until you reach a leaf node and return its answer.

Constructing DTs (ID3 algorithm)

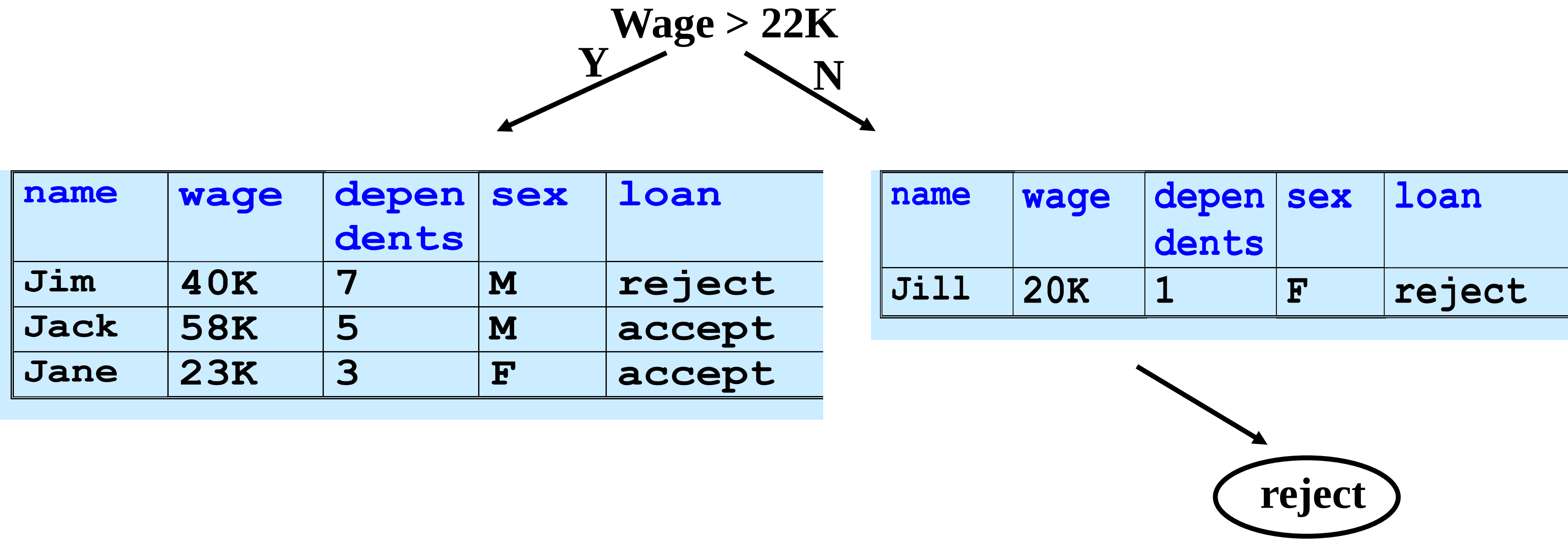
- **Top-down in recursive *divide-and-conquer* fashion:**
  1. **The best attribute is selected for root node and a branch is created for each possible attribute value**
  2. **The examples are split into subsets, one for each branch extending from the node**
  3. **The procedure is repeated recursively for each branch, using only the examples that reach the branch**
  4. **Stop if all examples have the same class; make a leaf node corresponding to this class**



Building a DT – Example; First Node

- Assume that we know how to select the best attribute at each step

name	wage	dependents	sex	loan
Jim	40K	7	M	reject
Jill	20K	1	F	reject
Jack	58K	5	M	accept
Jane	23K	3	F	accept

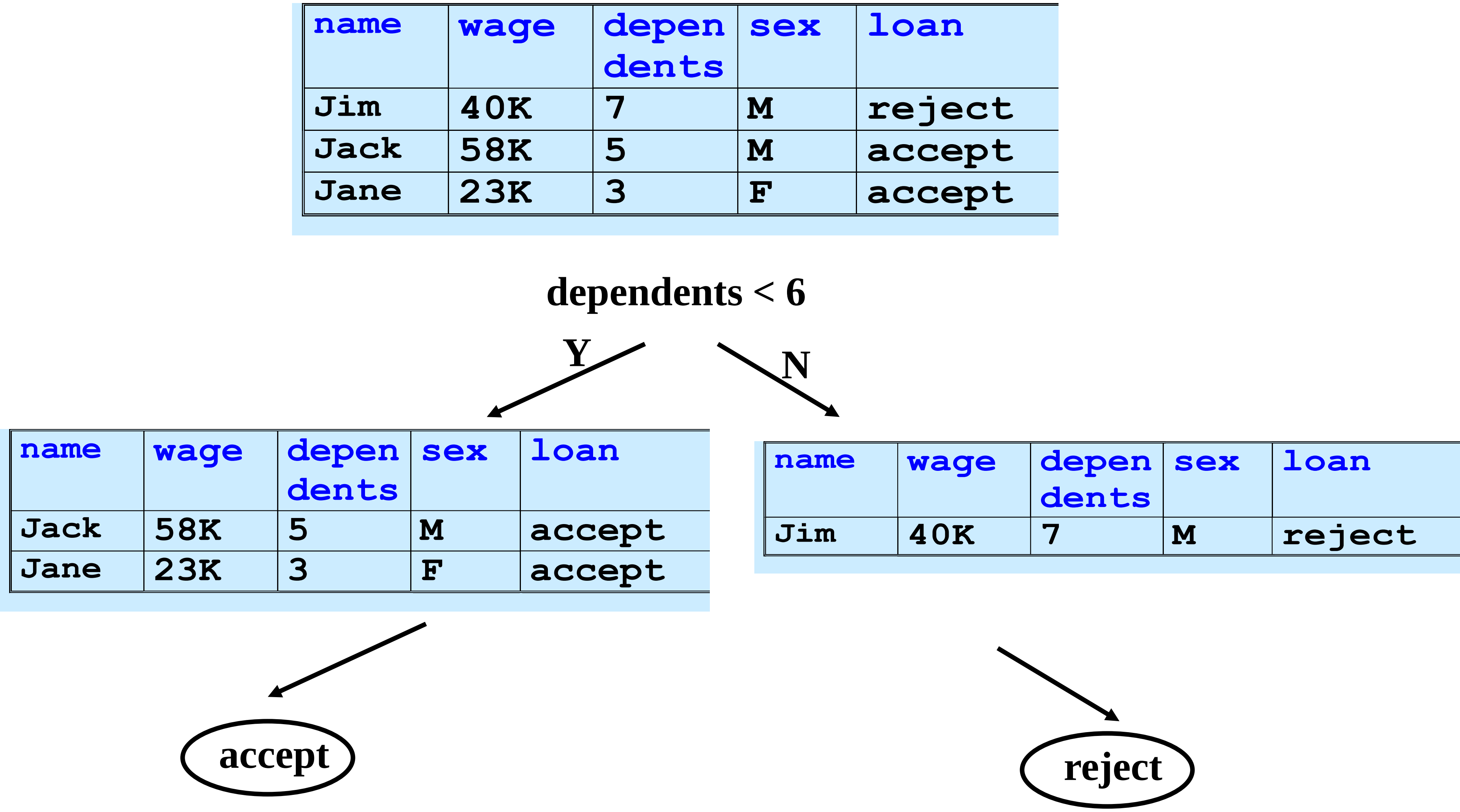


name	wage	dependents	sex	loan
Jim	40K	7	M	reject
Jack	58K	5	M	accept
Jane	23K	3	F	accept

name	wage	dependents	sex	loan
Jill	20K	1	F	reject

Example created by Eric McCreath

Second Node



name	wage	dependents	sex	loan
Jim	40K	7	M	reject
Jack	58K	5	M	accept
Jane	23K	3	F	accept

dependents < 6

Y

name	wage	dependents	sex	loan
Jack	58K	5	M	accept
Jane	23K	3	F	accept

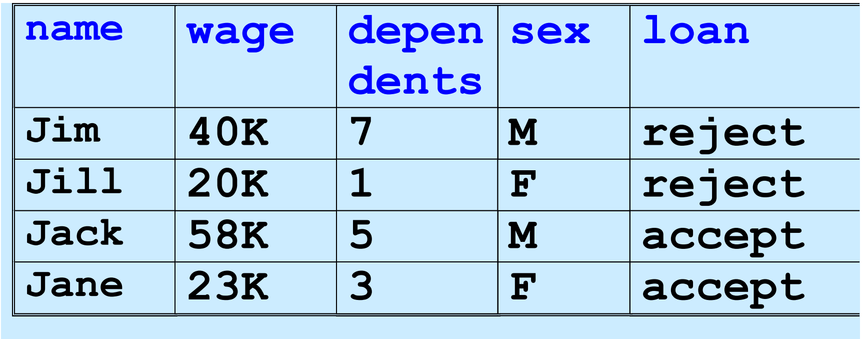
accept

N

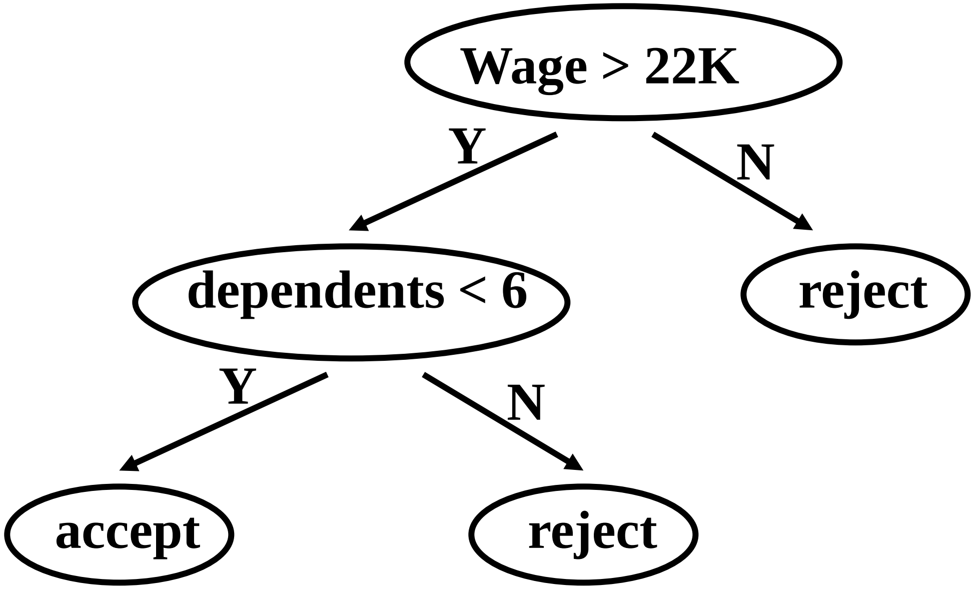
name	wage	dependents	sex	loan
Jim	40K	7	M	reject

reject

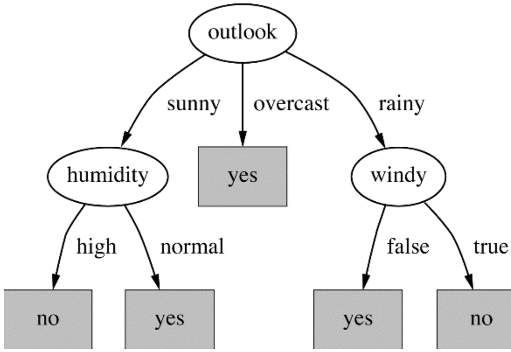
Final Tree



name	wage	dependents	sex	loan
Jim	40K	7	M	reject
Jill	20K	1	F	reject
Jack	58K	5	M	accept
Jane	23K	3	F	accept



Stopping Criterion Again – It is More Complex



- **4. Stop if**
  - **a)** ***[as before, the typical case]***

    all examples in the subset following the branch have the same class

    => make a leaf node corresponding to this class

  else

  - **b)** ***[additional condition]***

    Cannot split any further - all examples in the subset have the same attribute values but different classes (noise in data) => make a leaf node and label it with the <u>majority class of the subset</u>

  else

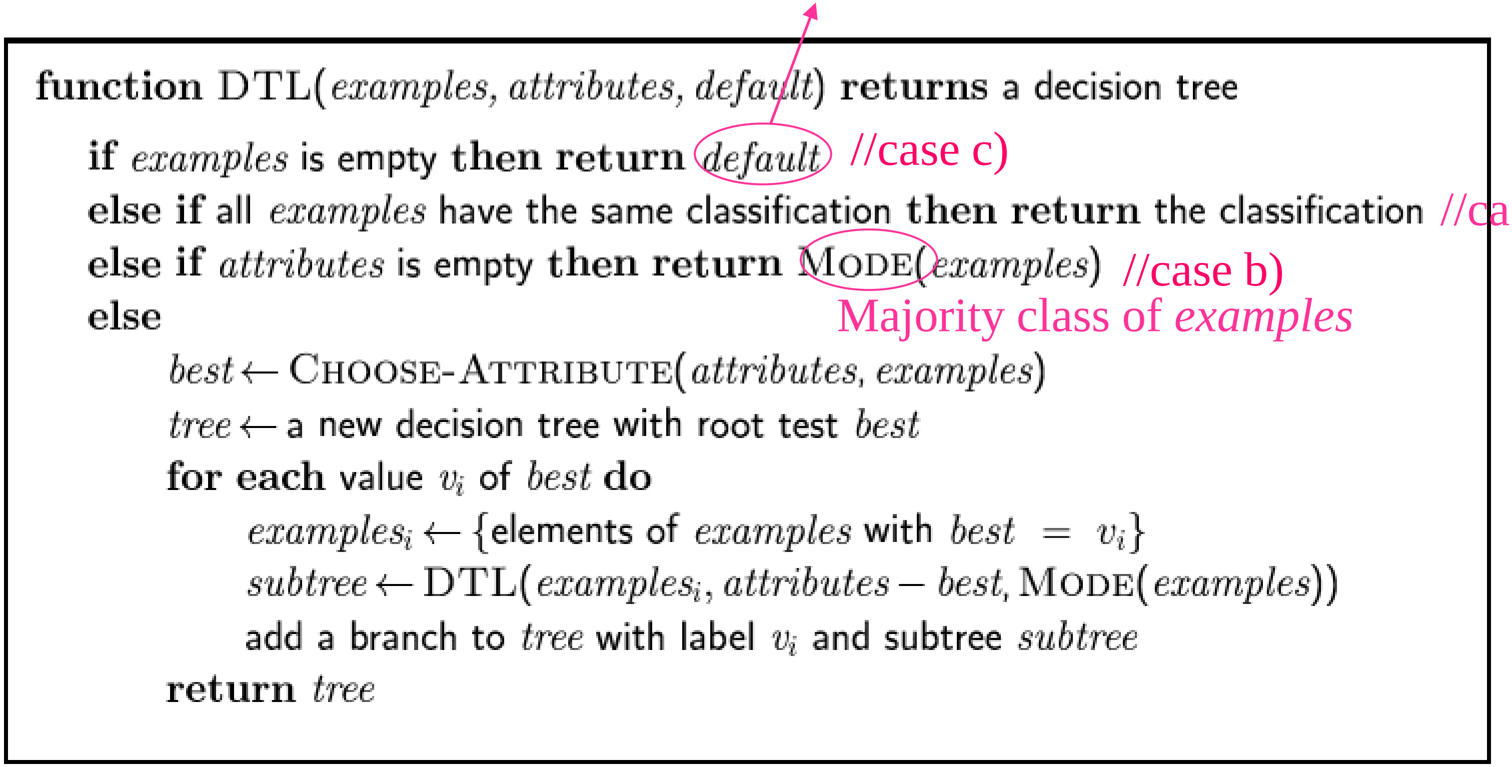
  - **c)** ***[additional condition]***

    The subset is empty (e.g. there are no examples with the attribute value for the branch) => create a leaf node and label it with the <u>majority class of the subset of the parent</u>

Pseudo-Code

- **Aim: find a tree consistent with the training examples**
- **Idea: recursively choose the best attribute as root of sub-tree**

Majority class of the sub-set of the parent node



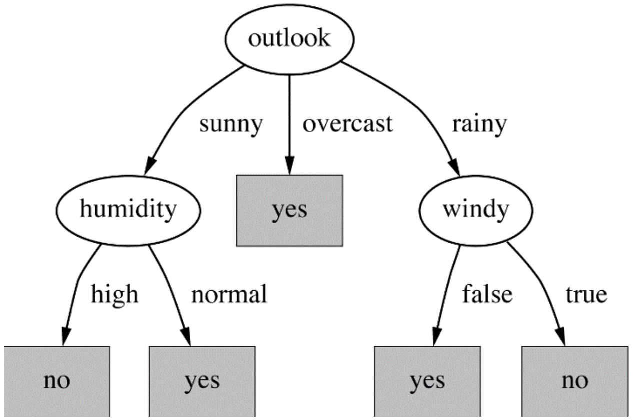
```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default  //case c)
    else if all examples have the same classification then return the classification  //case a)
    else if attributes is empty then return MODE(examples)  //case b)
    else
        best ← CHOOSE-ATTRIBUTE(attributes, examples)
        tree ← a new decision tree with root test best
        for each value $v_i$ of best do
            $examples_i$ ← {elements of examples with best = $v_i$}
            subtree ← DTL($examples_i$, attributes − best, MODE(examples))
            add a branch to tree with label $v_i$ and subtree subtree
        return tree
```

Majority class of *examples*

DT = A Set of Rules

- **DTs can be expressed as a set of mutually exclusive rules**
  - **Each rule is a conjunction of tests (1 rule = 1 path in the tree)**
  - **Rules are connected with disjunction**
  - **=> DT = disjunction of conjunctions**

Extract the rules for the weather tree!

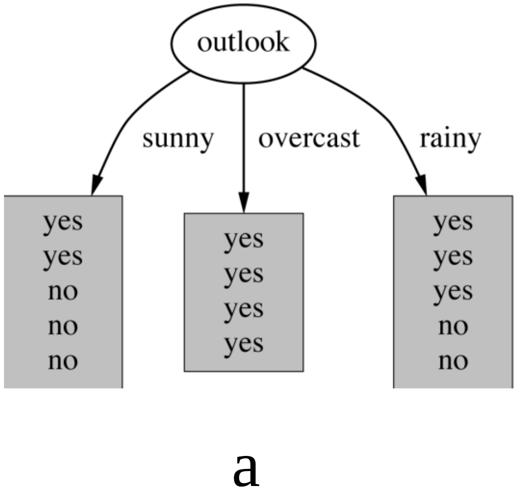


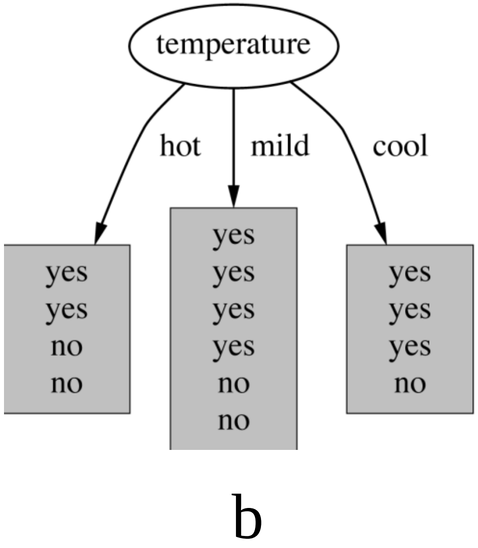
Expressiveness of DTs

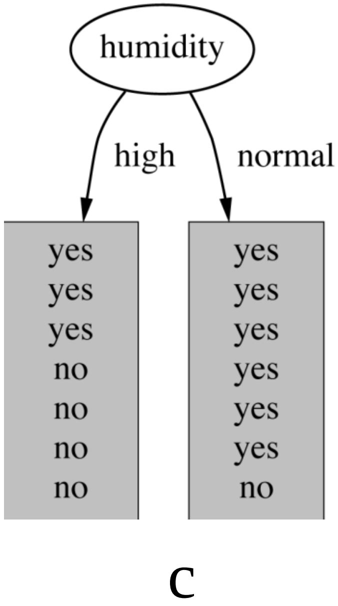
- **DTs can express any function of the input attributes**
  - **Reason: there is a consistent DT for any training set with 1 path to a leaf for each example**
    - **i.e. lookup-table, explicitly encoding the training data**
    - **the above is true unless *f(x)* is nondeterministic in *x***
  - **But such DT will probably not generalize well to new examples**
  - **=> Prefer to find more compact decision trees**

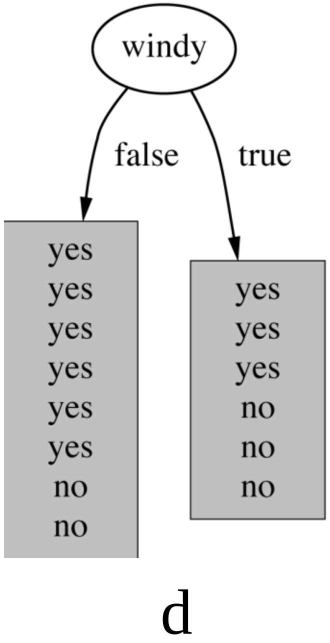
How to Find the Best Attribute?

- **Four DTs for the weather data; which is the best choice?**









- **A heuristic is needed!**

- **A measure of *‘purity’* of each node would be a good choice, as the “pure” subsets (containing only *yes* or *no* examples) will not have to be split further and the recursive process will terminate**

=> at each step we can choose the attribute which produces the purest children nodes; such measure of purity is called ...

Entropy (= Information Content) – Interpretation 1

- **Given a set of examples with their class**
  - **e.g. the weather data - 9 examples from class *yes* and 5 examples from class *no***
- **Entropy measures the homogeneity (purity) of a set of examples with respect to their class**
- **The smaller the entropy, the greater the purity of the set**

- **Standard measure in signal compression, information theory, physics**

Entropy – Formal Definition

- **Entropy H(S) of data set S:** $H(S) = I(S) = -\sum_i P_i . \log_2 P_i$

  P_i - proportion of examples that belong to class *i*

- **Different notation used in textbooks, we will use H(S) and I(S)**

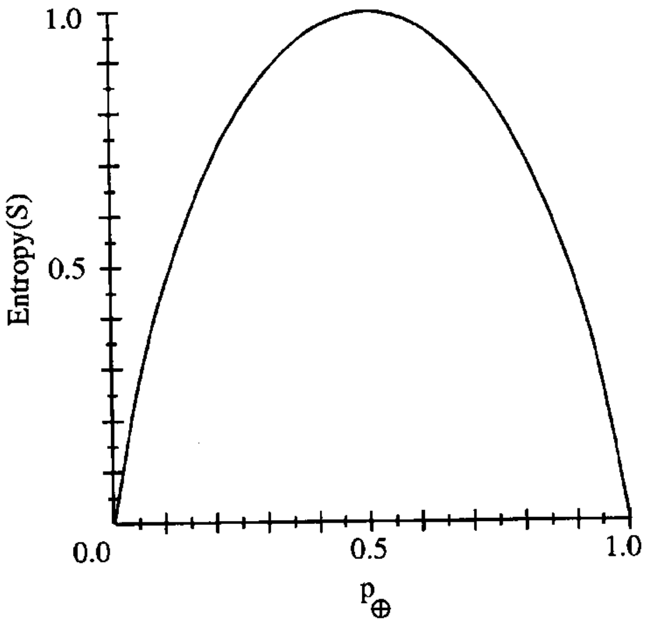
- **For our example: weather data - 9 yes and 5 no examples:**

$$H(S) = -P_{yes} \log_2 P_{yes} - P_{no} \log_2 P_{no} = I(P_{yes}, P_{no}) = I(\frac{9}{14}, \frac{5}{14}) = -\frac{9}{14} \log_2 \frac{9}{14} - \frac{5}{14} \log_2 \frac{5}{14} = 0.940 \, bits$$

- **log to the base 2 – entropy is measured in bits**

- **Note: log 0 is not defined, but when we calculate entropy we will assume that it is 0, i.e.** $\log_2 0 = 0$

Range of the Entropy for Binary Classification



graph from Tom Mitchell, Machine Learning, McGraw Hill, 1997

- **2 classes: yes and no**
- **on x: p - the proportion of positive examples**
  => the proportion of negative examples is 1-p
- **on y: the entropy H(S)**

$$H(S) = I(p,(1-p)) =$$
$$= -p\log_2 p - (1-p)\log_2(1-p)$$

- **H(S) ∈ [0,1]**
- **H(S)=0 => all examples in S belong to the same class (no disorder, min entropy)**
- **H(S)=1 => equal number of yes & no (S is as disordered as it can be; max entropy)**

Entropy – Interpretation 2 (Information Theory Interpretation)

- Sender and Receiver
- Sender sends information to Receiver about the outcome of an event (e.g. flipping a coin, 2 possibilities: heads or tails)
- Receiver has some prior knowledge about the outcome, e.g.
  - Case 1: knows that the coin is honest
  - Case 2: knows that the coin is rigged so that it comes heads 75% of the time
- In which case an answer telling the outcome of a toss will be *more informative* for the Receiver (i.e. will be *less predictable*)?

Information Theory

- **Information theory, Shannon and Weaver 1949:**
  - **Given a set of possible answers (messages) $M=\{m_1,m_2,\dots,m_n\}$ and a probability $P(m_i)$ for the occurrence of each answer, the *expected information content (entropy)* of the actual answer is:**

$$H(M) = -\sum_{i}^{n} P(m_i).\log_2 P(m_i) = entropy(M)$$

  - **Shows the amount of *surprise* of the receiver by the answer based on the probability of the answers (i.e. based on the prior knowledge of the receiver about the possible answers)**
    - **The less the receiver knows, the more information is provided (the more informative the answer is)**

Another Example

- **Example 2: Basketball game outcome**
- **Two teams A and B are playing basketball**
  - **Case 1: Equal probability to win**
  - **Case 2: Michael Jordan (a famous basketball player) is playing for A and the probability of A to win is 90%**
  - **In which case an answer telling the outcome of the game will contain more information? (i.e. will be less predictable)**

Entropy Calculation for the Examples

Calculate the entropy for the two examples: coin flipping and basketball game

- **Example 1: Flipping coin**
  - **Case 1 (honest coin):**

$H(coin\_toss)= I(1/2, 1/2)= -(1/2)\log(1/2) -(1/2)\log(1/2) = 1$ bit

  - **Case 2 (rigged coin):**

$H(coin\_toss) = I(1/4, 3/4)= -(1/4)\log(1/4) -(3/4)\log(3/4) = 0.811$ bits

- **Example 2: Basketball game outcome**
  - **Case 1 (equal probability to win):**

$H(game\_outcomes) = I(1/2, 1/2)= 1$ bit

  - **Case 2 (non-equal probability to win):**

$H(game\_outcome) = I(90/100, 10/100)=$

$=-(90/100) \log(90/100) -(10/100)\log(10/100) = \ldots < 1$ bit

Entropy – Interpretation 3

adapted from http://www.cs.cmu.edu/awm/tutorials

- Suppose that X is a random variable with 4 possible values A, B, C and D; P(X=A)=P(X=B)=P(X=C)=P(X=D)=1/4.
- You must transmit a sequence of values of X over a serial binary channel. How?
  - Solution: encode each symbol with 2 bits, e.g. A=00, B=01, C=10, D=11
    - ABBBCADBADCB…
    - 000101011000110100111001…

- Now you are told that the probabilities are not equal, e.g. P(X=A)=1/2, P(X=B)=1/4, P(X=C)=P(X=D)=1/8.
- Can you invent a better coding? E.g. a coding that uses less than 2 bits on average per symbol, e.g. 1.75 bits?
- A=? 0 C=? 110
- B=? 10 D=? 111

Calculating the Number of Bits per Symbol

- **P(X=A)=1/2, P(X=B)=1/4, P(X=C)=P(X=D)=1/8**
- **Coding used: A=0, B=10, C=110, D=111**

Average number of bits per symbol =?

1/2*1+1/4*2+1/8*3+1/8*3 =1+3/4=1.75 bits

- **Is this the smallest number of bits per symbol?**
- **Yes, see the next 2 slides.**

Entropy – Interpretation 3; General Case

adapted from http://www.cs.cmu.edu/~awm/tutorials

- **Suppose that X is a random variable with m possible values V1…Vm; P(X=V1)=P1, P(X=V2)=P2,…, P(X=Vm)=Pm.**
- **The smallest possible number of bits per symbol on average needed to transmit a stream of symbols drawn from X's distribution is**

$$H(X) = -\sum_{i=1}^{m} P_i . \log_2 P_i$$

- **High entropy – the values of X are all over place**
  - **The histogram of the frequency distribution of values of X will be flat**
- **Low entropy – the values of X are more predictable**
  - **The histogram of the frequency distribution of values of X have many lows and one or two highs**

Let’s Check the Entropy!

- **Let’s calculate the entropy (= minimum number of bits per symbol on average) and compare with our coding which required 1.75 bits**

- **The probabilities were:**
  $P(X=A)=1/2$, $P(X=B)=1/4$, $P(X=C)=P(X=D)=1/8$

- **$H(X)=?$**

Let’s Check the Entropy!

- **Let’s calculate the entropy (= minimum number of bits per symbol on average) and compare with our coding which required 1.75 bits**

- **The probabilities were:**

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

- **$H(X)=I(1/2,1/4,1/8,1/8)=-1/2\log1/2-1/4\log1/4-1/8\log1/8-1/8\log1/8=1/2+1/2+3/8+3/8=1.75$ bits**
- **=> the entropy is 1.75 bits and our coding also uses 1.75 bits**
- **=> our coding is good – it uses the smallest number of bits per symbol on average!**

Information Gain

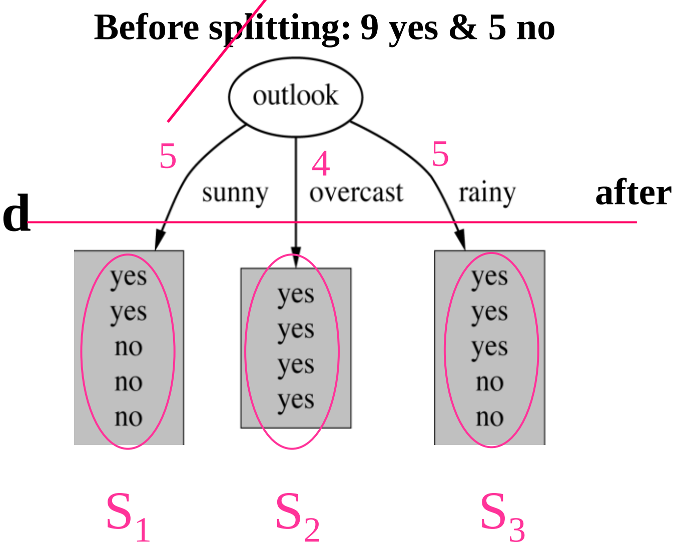
- Back to DTs…
- Entropy measures
  - the disorder of a set of training examples with respect to their class Y
  - shows the amount of surprise of the receiver by the answer Y based on the probability of the answers
  - the smallest possible number of bits per symbol (on average) needed to transmit a stream of symbols drawn from Y's distribution
- We can use the first one to define a *measure for the effectiveness of an attribute* to classify the training data
- This measure is called *information gain*
- It measures the *reduction in entropy* caused by using this attribute to partition the set of training examples
- The best attribute is the one with the *highest information gain* (i.e. with the *biggest reduction in entropy*)

Information Gain - Example

- Let’s calculate the information gain of the attribute `outlook`
- Information gain measures reduction in entropy
- It is a difference of 2 terms: *T1 –T2*
- *T1* is the entropy of the set of examples S associated with the parent node before the split

$$T1 = H(S) = I(\frac{9}{14}, \frac{5}{14}) = 0.940\ bits$$

5 examples go along this branch



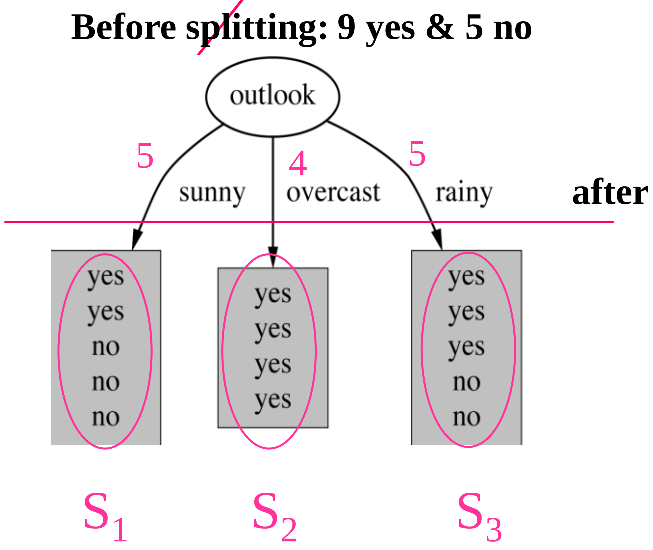
- *T2* is the remaining entropy in S, after S is split by the attribute (e.g. `outlook`)
  - It takes into consideration the entropies of the child nodes and the distribution of the examples along each child node
  - E.g. for a split on `outlook,` it will consider the entropies of S1, S2 and S3 and the proportion of examples following each branch (5/14, 4/14, 5/15):

$$T2 = H(S \mid outlook) = \frac{5}{14} \cdot H(S_1) + \frac{4}{14} \cdot H(S_2) + \frac{5}{14} \cdot H(S_3)$$

Information Gain – Example (cont.)

5 examples go along this branch

- **Gain = T1-T2**
- ***T1* is the entropy of the set of examples S at the parent node before the split**
- ***T2* is the remaining entropy in S, after S is split by an attribute**



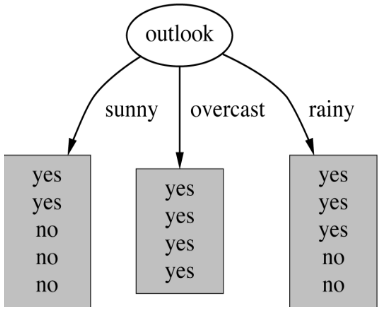
- **The larger the difference, the higher the information gain**
- **The <u>best attribute</u> is the one with highest information gain**
  - **it <u>reduces most</u> the entropy of the parent node**
  - **it is expected to lead to pure partitions fastest**

Information Gain – Formal Definition

- **The information gain of an attribute A, Gain(S|A), is the <u>reduction in entropy</u> caused by the partitioning of the set of examples S using A**

$$Gain(S \mid A) = H(S) - \sum_{j \in values(A)} P(A = v_j) H(S \mid A = v_j) =$$

$$= H(S) - \sum_{j \in values(A)} \frac{|S_v|}{|S|} H(S \mid A = v_j)$$



Gain(S|A) is the information gain of an attribute A relative to S

Values(A) is the set of all possible values for A

S_v is the subset of S for which A has value v

=entropy of the original data set S

=conditional entropy H(Y|A);
=expected value of the entropy after S is partitioned by A
=called *Reminder* in Russell and Norvig

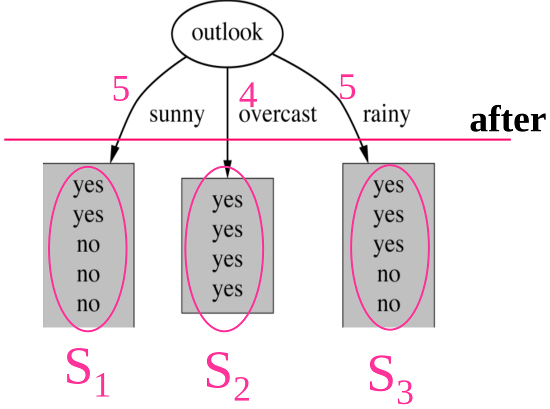
Computing the Reminder for outlook

- **Fully computing the second term (the reminder) of H(S|outlook):**

$$T2 = H(S\,|\,outlook) = \frac{5}{14}\cdot H(S_1) + \frac{4}{14}\cdot H(S_2) + \frac{5}{14}\cdot H(S_3)$$

we will create a leaf in this branch (if `outlook` is chosen for splitting)

Before splitting: 9 yes & 5 no



$$H(S\,|\,outlook = sunny) = I(\frac{2}{5},\frac{3}{5}) = -\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5} = 0.971\,bits$$

$$H(S\,|\,outlook = overcast) = I(\frac{4}{4},\frac{0}{4}) = -\frac{4}{4}\log_2\frac{4}{4} - \frac{0}{4}\log_2\frac{0}{4} = \boxed{0\,bits}$$

$$H(S\,|\,outlook = rainy) = I(\frac{3}{5},\frac{2}{5}) = -\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5} = 0.971\,bits$$

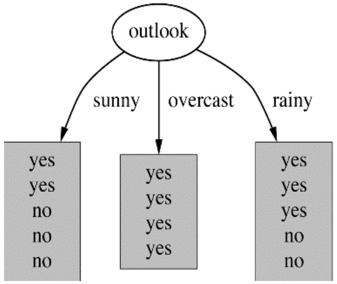
$$H(S\,|\,outlook) = \frac{5}{14}\cdot 0.971 + \frac{4}{14}\cdot 0 + \frac{5}{14}\cdot 0.971 = 0.693\,bits$$

(This is the amount of information that we expect would be necessary to determine the class of an example, given the tree structure *a*; how difficult it will be to predict the class given this tree)

Computing the Information Gain for outlook

- => the *information gain* that would be gained by branching on `outlook` is:

Gain(S|outlook)=H(S)-H(S|outlook)=
0.940-0.639=0.247 bits

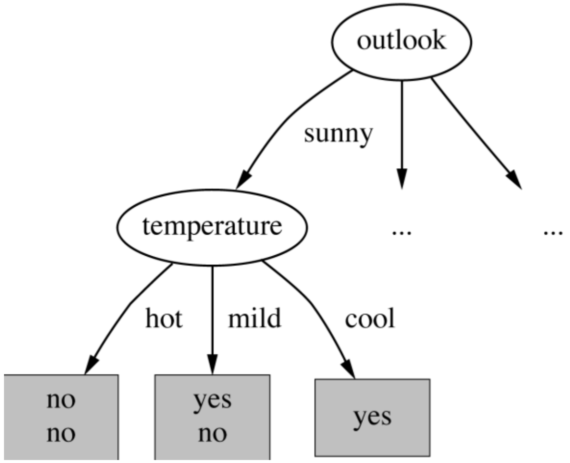


- Similarly, the information gain for the other three attributes is:

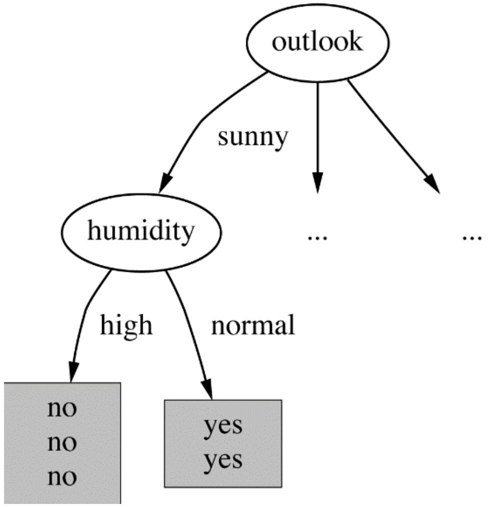
Gain(S|temperature)=0.029 bits
Gain(S|humidity)=0.152 bits
Gain(S|windy)=0.048 bits

- => we select `outlook` for splitting of the tree because it has the highest information gain

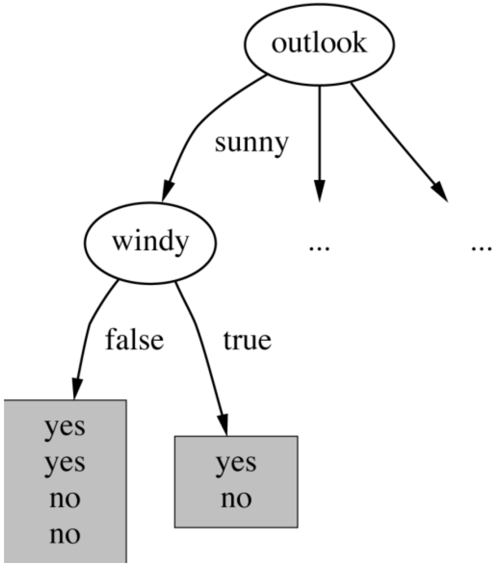
Continuing to Split



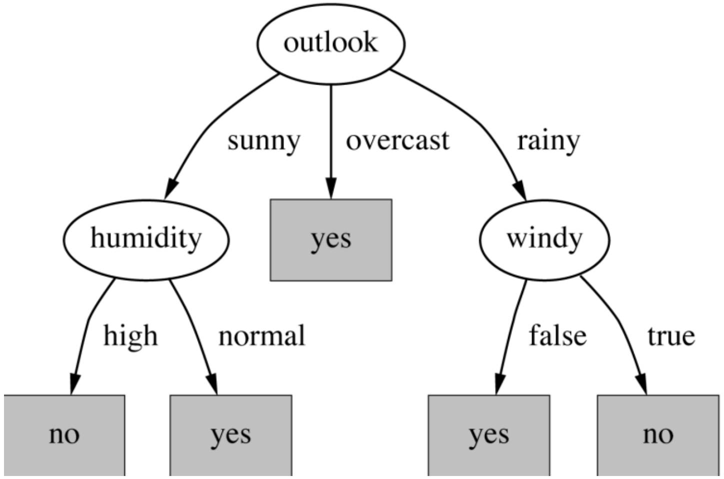
Gain(S, temperature)=0.571 bits



Gain(S, humidity)=0.971 bits



Gain(S, windy)=0.020 bits

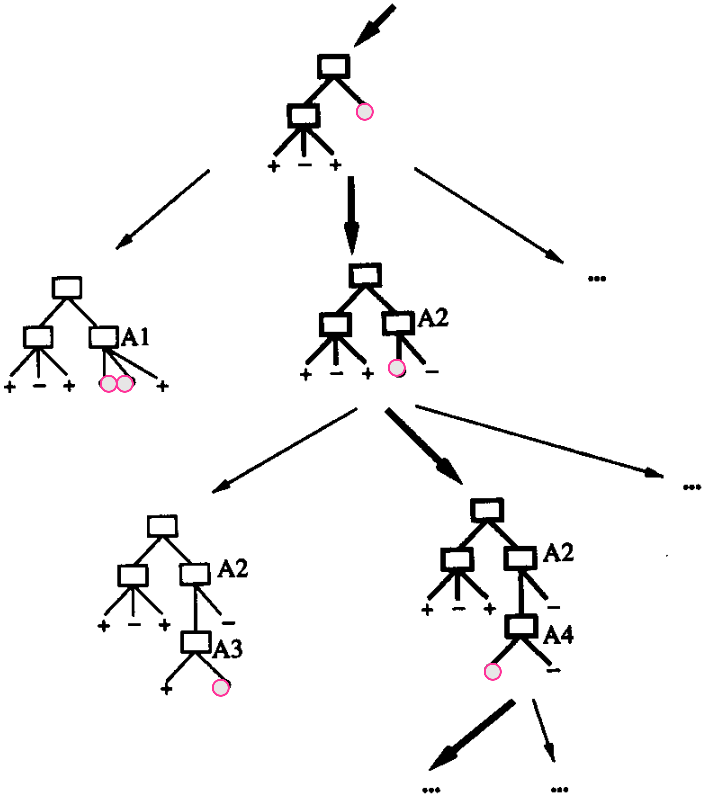


Final DT

Building DT as a Search Problem

- DT algorithm searches the hypothesis space for a hypothesis that *fits* (correctly classifies) the training data
  - What is the search strategy?
  - Simple to complex search (starting with an empty tree and progressively considering more elaborate hypotheses)
  - Hill climbing with information gain as an evaluation function
- Information gain is an evaluation function of how good the current state is (how close we are to a goal state, i.e. a tree that classifies correctly all training examples)

image ref: Tom Mitchell, Machine Learning, McGraw Hill, 1997

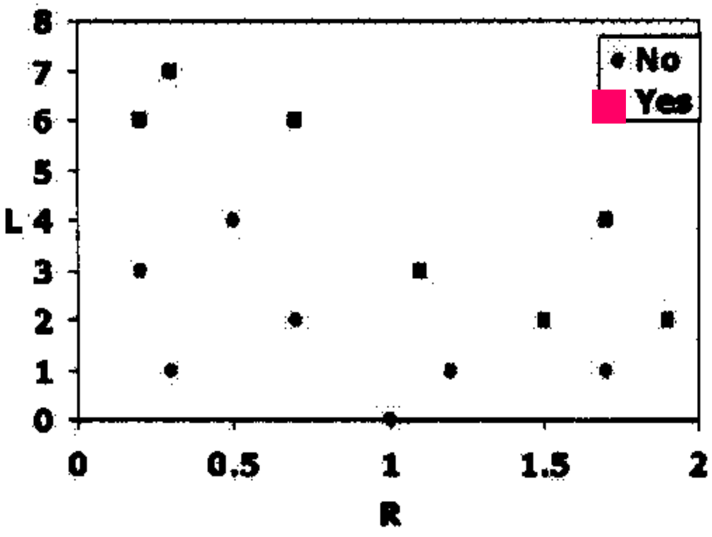


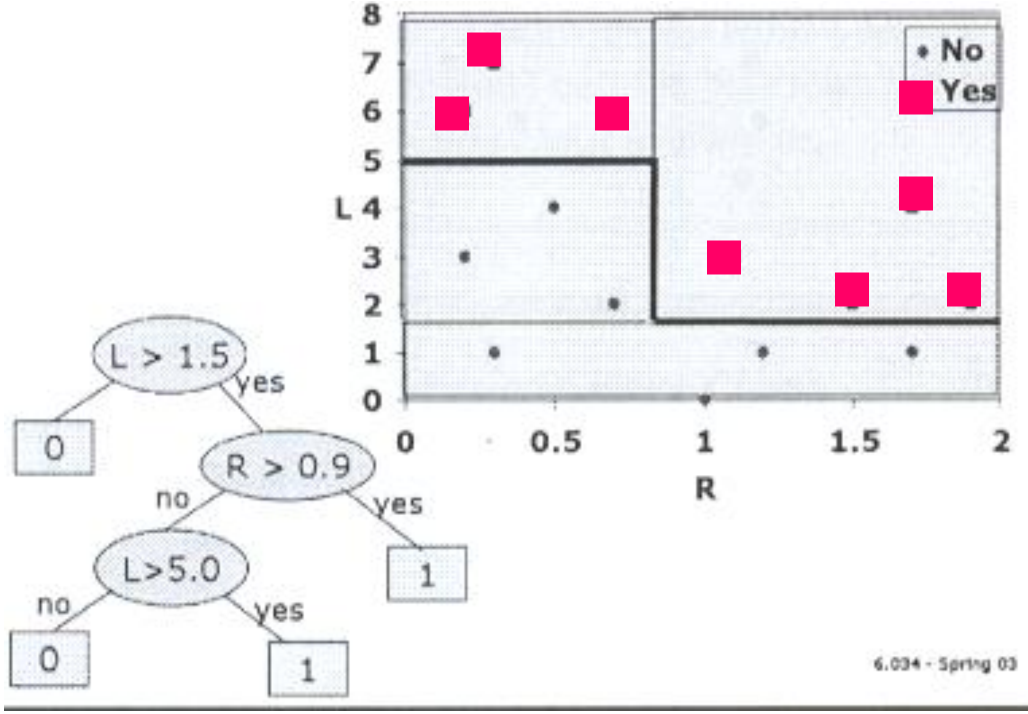
DT Decision Boundary

Example taken from 6.034 AI, MIT

- DTs define a decision boundary in the feature space
- Example: binary classification, numeric attributes (binary DT)

Bankruptcy Example





2 attributes:
R – ratio of earnings to expenses
L – number of late payments on credit cards over the past year

Additional Topics

- **Avoiding Overfitting**
- **Dealing with Numeric Attributes**
- **Alternative Measures for Selecting Attributes**
- **Dealing with Missing Attributes**
- **Handling Attributes with Different Costs**

Overfitting

- ***Overfitting:*** **the error on the training data is very small but the error on new data (test data) is high**

  => The classifier has memorized the training examples but has not extracted pattern from them and is not classifying well the new examples!

- **More formal definition of overfitting (generic, not only for DTs) [Tom Mitchell, Machine Learning, McGraw Hill, 1997]:**
  - **given: H - a hypothesis space, a hypothesis $h \in H$,**

    D - entire distribution of instances, train - training instances
  - **h is said to overfit the training data if there exist some alternative hypothesis $h' \in H$:**

    $$error_{train}(h) < error_{train}(h') \quad \& \quad error_D(h) > error_D(h')$$

- **Overfitting is a problem not only for DTs but for most of the classifiers**

Overfitting in DTs

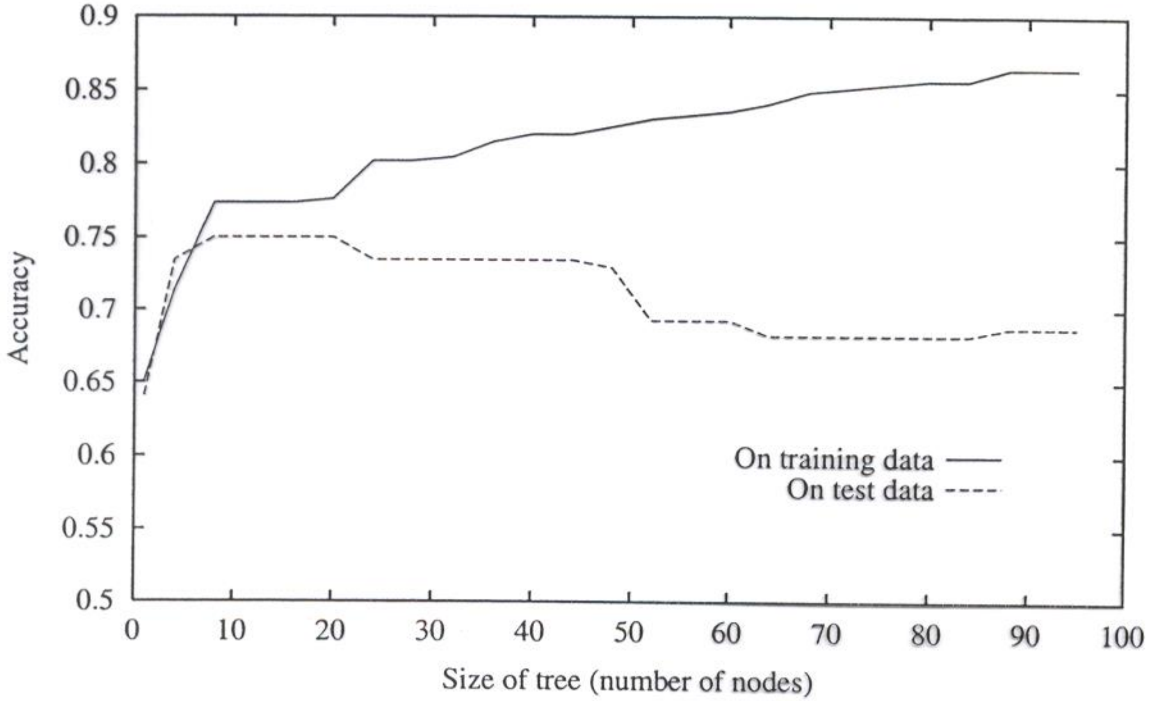


image ref: Tom Mitchell, Machine Learning, McGraw Hill, 1997

Overfitting in DTs - Reasons

- **A DT grows each branch of the tree deeply enough to perfectly classify the training examples**
- **In doing this (and depending on the data), the tree may become very specific (like a look-up table) and not represent patterns, but only memorize the data**

- **Problems with the training data -> problems with the induced DT**
  - **Training data is too small -> not enough representative examples to build a model that can generalize well on new data**
  - **Noise in the training data, e.g. incorrectly classified examples -> DT learns them but they are incorrect -> will incorrectly classify new examples, see next slide**

Read at home

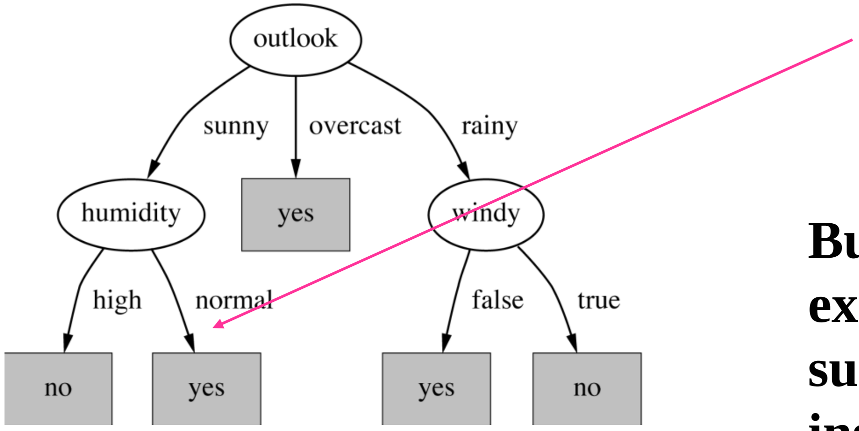
Overfitting Due to Noise - Example

- **How can it be possible for tree h to fit the training examples better than h' but to perform worse on new examples?**

  - **Example - noise in the labeling of a training example**

    E.g. adding to the weather data the following positive example that is incorrectly labeled as negative: outlook=sunny, temperature=hot, humidity=normal, wind=yes, play=no

Original tree h'



New tree h:

further tests below this node;
=> h will be more complex that h'

But h is a result of fitting the noisy training example=> we expect h' to outperform h on subsequent data drawn from the same instance distribution

Tree Pruning

- **Used to avoid overfitting in DTs; 2 main approaches:**
  - ***pre-pruning*** **- stop growing the tree earlier, before it reaches the point where it perfectly classifies the training data**
  - ***post-pruning*** **– fully grow the tree (allowing it to perfectly cover the training data) and then prune it**
- ***Post-pruning*** **is more successful and widely used; 2 main approaches:**
  - ***Tree pruning*** **- directly prune the tree**
    - ***sub-tree replacement***
    - ***sub-tree raising***
  - ***Rule pruning*** **- convert the tree into a set of rules and then prune them**

When to Stop Pruning?

- **How to determine when to stop pruning?**
- **Solution: estimate accuracy using**
  - **validation set or**
  - **training data – too optimistic (heuristic based on some statistical reasoning but the statistical underpinning is rather weak)**

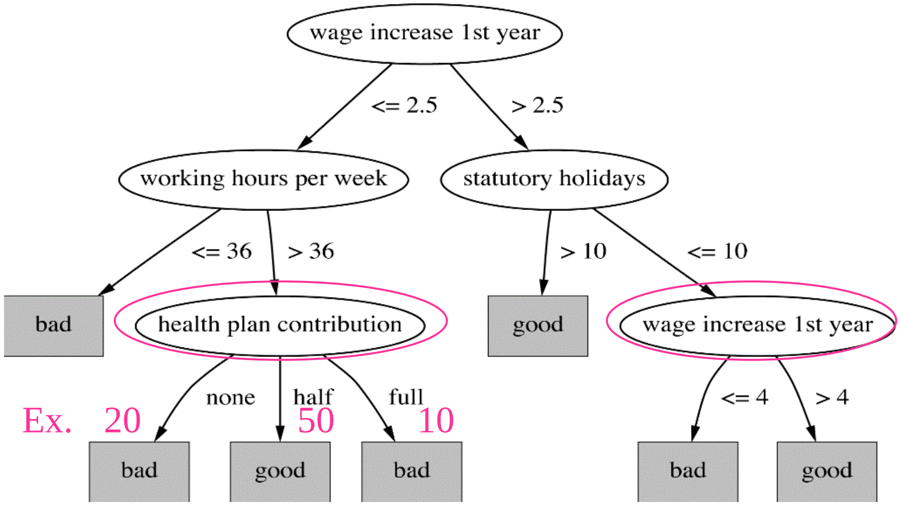
Using Validation Set

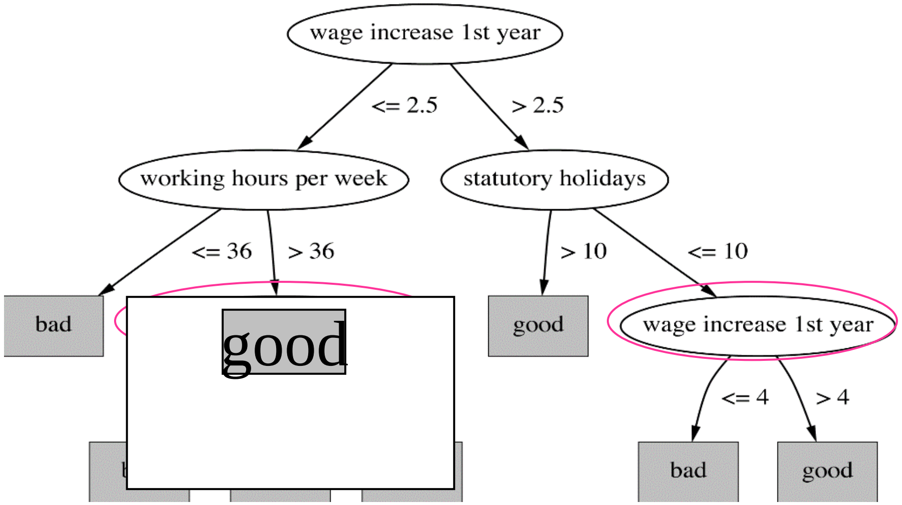
- Available data is separated into 3 sets:
  - training set - used to build the DT
  - validation set - to evaluate the impact of pruning and decide when to stop
  - test data – to evaluate how good the final DT is

- Motivation
  - Even though the DT may be misled by random errors and coincidental regularities within the training set, the validation set is unlikely to exhibit the same random fluctuations => the validation set can provide a *safety check* against overfitting of the training set
  - The validation set should be large enough; typically: 1/2 of the available examples are used as training set, 1/4 as validation set and 1/4 as test set
- Disadvantage: the tree is built on less data
  - When the data is limited, withholding part of it for validation reduces even further the examples available for training

Tree Post-Pruning by Sub-Tree Replacement

- Each non-leaf node (i.e. each test node) is a candidate for pruning
- Start from the leaves and work towards the root
- For each candidate node
  - Remove the sub-tree rooted at it
  - Replace it with a leaf with class=majority class of examples that go along the candidate node
  - Compare the new tree with the old tree by calculating the accuracy on the validation set for both
  - If the accuracy of the new tree is better or the same as the accuracy of the old tree, keep the new tree (we say that the candidate node is pruned)

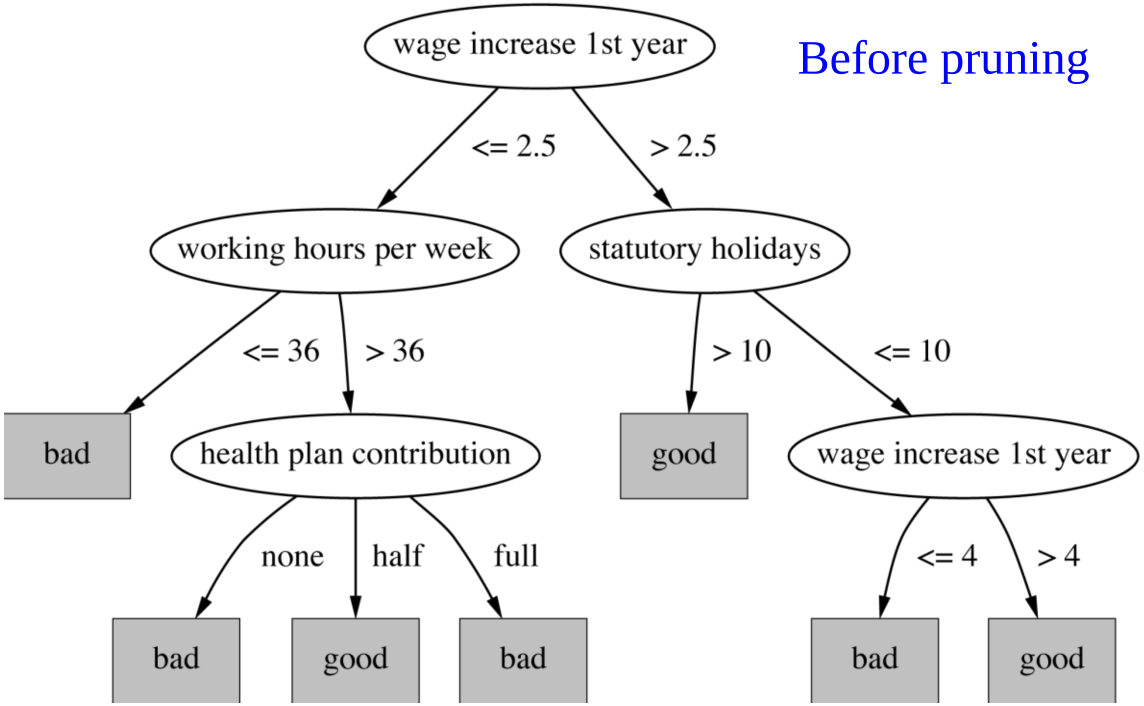




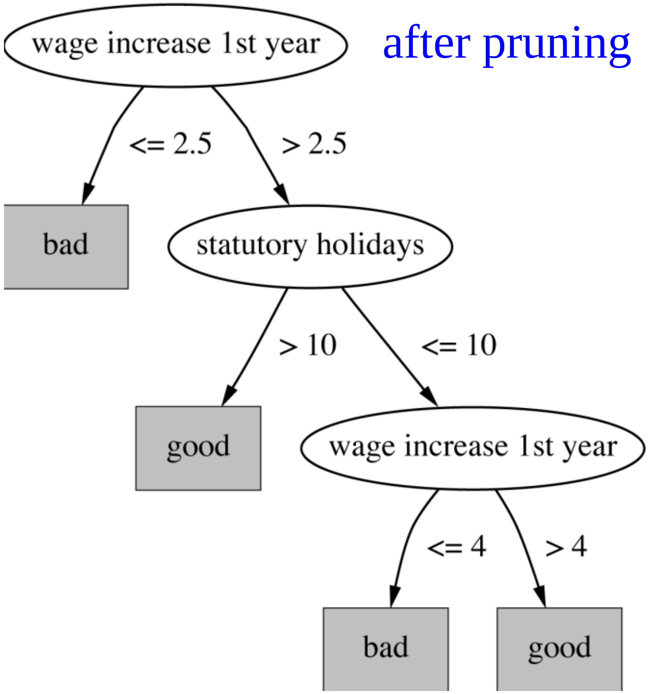
- At each step, we will evaluate all candidate nodes and accept the best pruning – i.e. the one that will improve the accuracy most. No pruning if the new tree is worse than the old tree.

Tree Post-Pruning by Sub-Tree Replacement - 2

- **Continue iteratively until further pruning is harmful, i.e. it decreases the accuracy on the validation set**

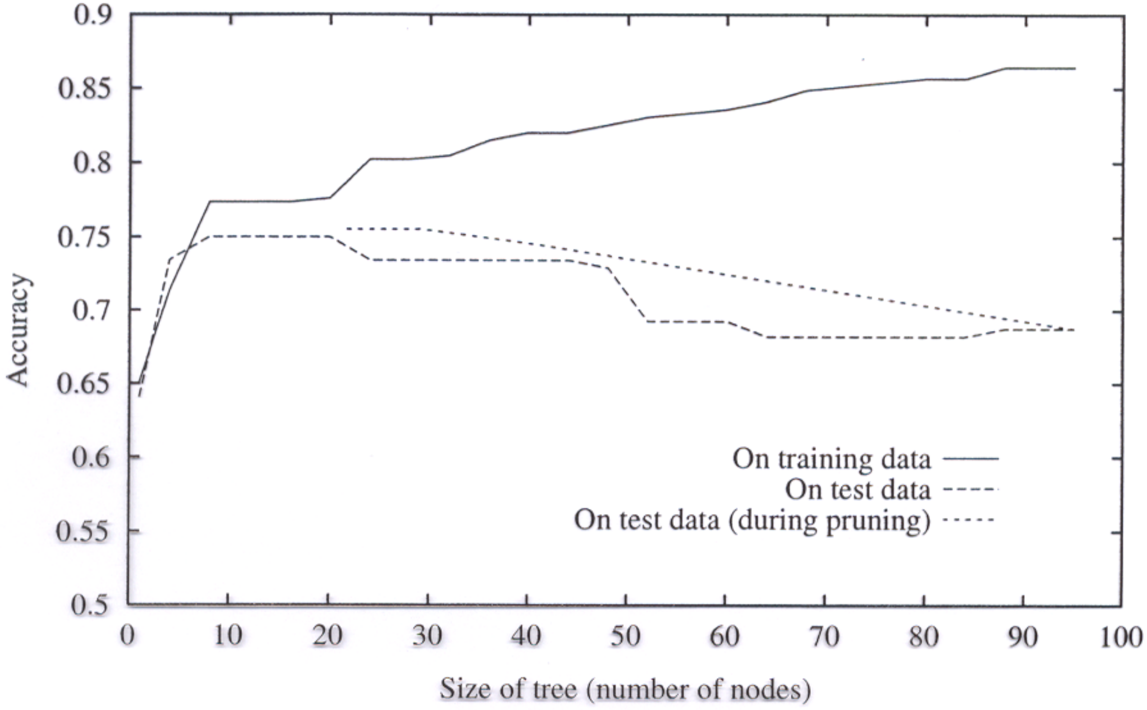


Before pruning



after pruning

Effect of Tree Post-Pruning by Sub-Tree Replacement

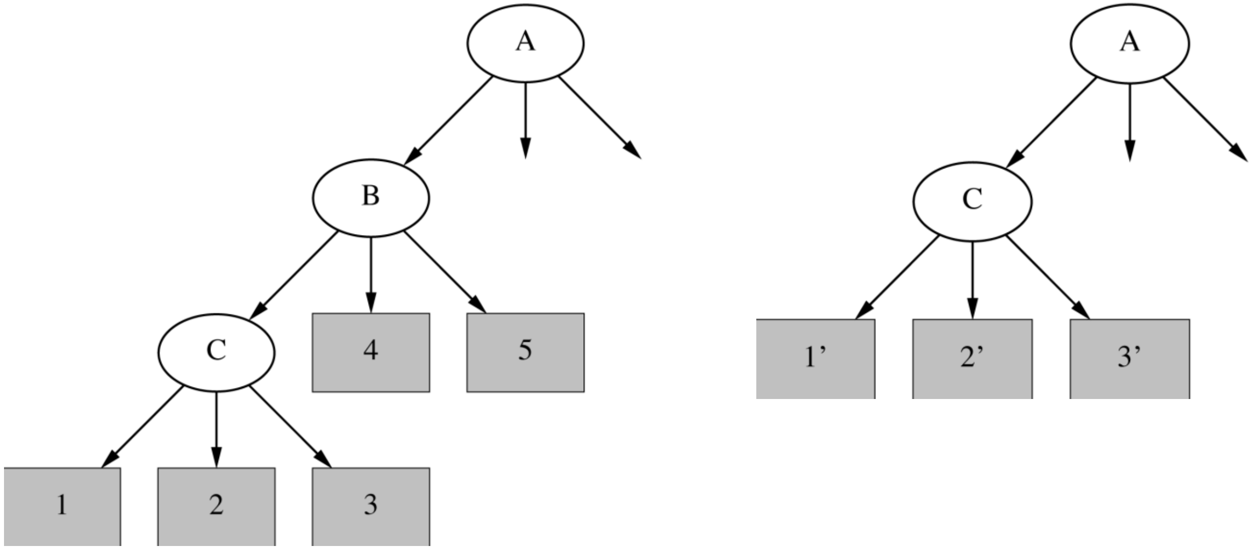


- **The accuracy on test data increases as nodes are pruned (accuracy on validation set is not shown)**

graph from Tom Mitchell, Machine Learning, McGraw Hill, 1997

Post-Pruning: Sub-Tree Raising

- **More complex operation than sub-tree replacement**



Node C is raised to subsume node B

examples at nodes 4 , 5 are re-classified into the new sub-tree headed by C => 1', 2', 3'

- **sub-tree raising is potentially time consuming operation => it is restricted to raising the sub-tree of the most popular branch**
  - **e.g. raise C only if the branch from B to C has more examples than the branches from B to 4 or from B to 5**
  - **otherwise, if e.g. 4 were the majority child of B, consider raising 4 to replace B and re-classifying all examples under C, as well as the examples from 5, into the new node**

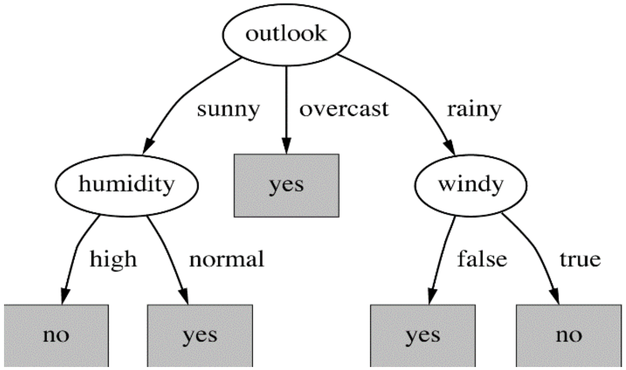
Rule Post-Pruning - Example

- 1. Grow the tree until it fits the training data
- 2. Convert it into an equivalent set of rules by creating 1 rule for each path from the root to a leaf , e.g.:

  Rule1: if `(outlook=sunny) & humidity=high)` then `play=no`

  …

  Rule 5: …



- Prune each rule by removing the pre-conditions that are not harmful, i.e. the estimated rule accuracy is the same or higher
  - estimated accuracy – accuracy on validation set or training set (see previous slides)
- Sort the pruned rules by their estimated accuracy, and consider them in this sequence when classifying subsequent instances

Rule Post-Pruning - cont.

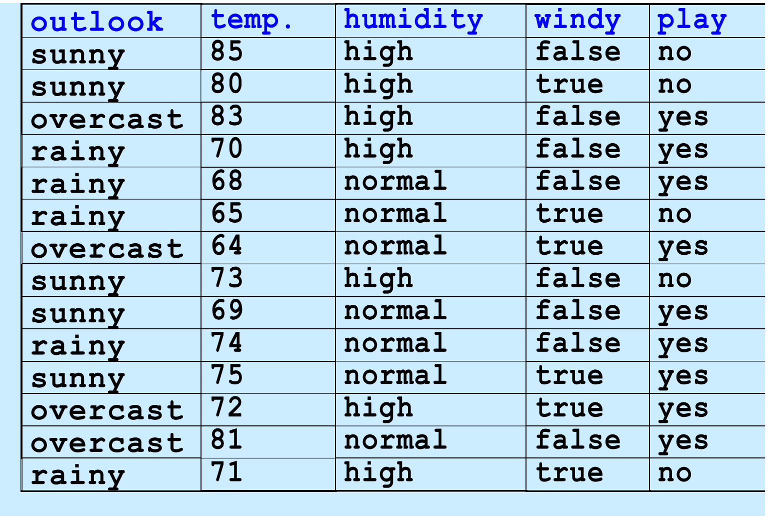
- **Why convert DT to rules before pruning?**

- **Bigger flexibility**
  - **When trees are pruned: only 2 choices - to remove the node completely or retain it**
  - **When rules are pruned - less restrictions**
    - **Pre-conditions (not nodes) are removed**
    - **Each branch in the tree (i.e. each rule) is treated separately**
    - **Removes the distinction between attribute tests that occur near the root of the tree and those near the leaves**

- **Advantage of rules over trees - easier to read a set of rules than a tree**

Numeric Attributes

- Numerical attributes need to be converted into nominal - this is called discretization



outlook	temp.	humidity	windy	play
sunny	85	high	false	no
sunny	80	high	true	no
overcast	83	high	false	yes
rainy	70	high	false	yes
rainy	68	normal	false	yes
rainy	65	normal	true	no
overcast	64	normal	true	yes
sunny	73	high	false	no
sunny	69	normal	false	yes
rainy	74	normal	false	yes
sunny	75	normal	true	yes
overcast	72	high	true	yes
overcast	81	normal	false	yes
rainy	71	high	true	no

- In DTs, for a numerical attribute we restrict the possibilities to a binary split (e.g. temp<60)

- Discretization procedure:
  - Sort the examples according the values of the numerical attribute
  - Identify adjacent examples that differ in their class and generate a set of candidate splits (split points are placed halfway)
  - Evaluate Gain (or other measure) for every possible split point and choose the best split point
  - Gain for best split point is Gain for the attribute

Numeric Attributes - Example

- values of **temperature**:

64 | 65 | 68 69 70 | 71 72 73 | 74 75 | 80 | 81 83 | 85
yes | no | yes yes yes | no no no | yes yes | no | yes yes | no

- **7 possible splits; consider split between 70 and 71**
- **Information gain for**
  - **- 1) temperature < 70.5 : 4 yes & 1 no**
  - **- 2) temperature =>70.5 : 4 yes & 5 no**

$$H(S) = -\frac{8}{14}\log_2\frac{8}{14} - \frac{6}{14}\log_2\frac{6}{14} = 0.985\,bits$$

$$H(S_{temp<70.5}) = -\frac{4}{5}\log_2\frac{4}{5} - \frac{1}{5}\log_2\frac{1}{5} = 0.722\,bits$$

$$H(S_{temp=>70.5}) = -\frac{4}{9}\log_2\frac{4}{9} - \frac{5}{9}\log_2\frac{5}{9} = 0.991\,bits$$

$$H(S\,|\,temp70.5) = \frac{5}{14}0.722 + \frac{9}{14}0.991 = 0.895\,bits$$

$$Gain(S\,|\,temp70.5) = 0.985 - 0.895 = 0.09\,bits$$

$$Gain(S\,|\,A) = H(S) - \sum_{v\in Values(A)} \frac{|S_v|}{|S|} H(S_v)$$

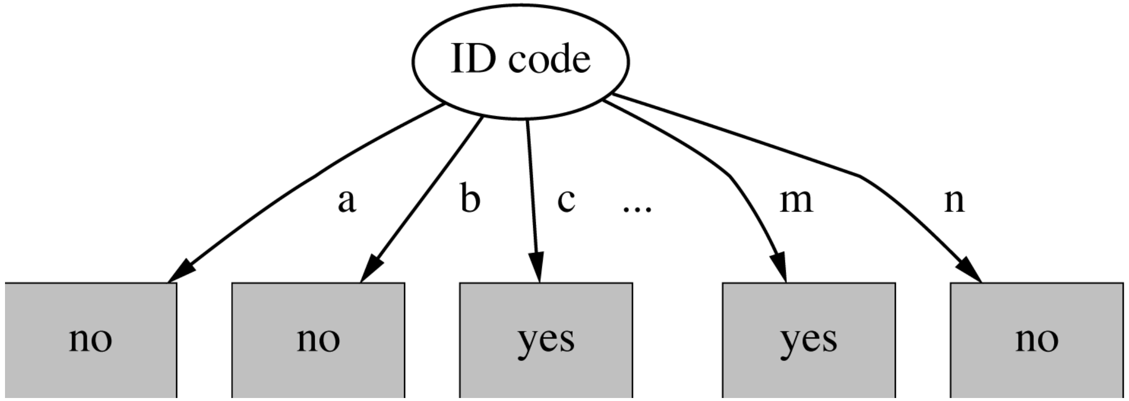
Alternative Measures for Selecting Attributes

- **Problem: if an attribute is highly-branching (with a large number of values), information gain will select it!**
- **Reason: highly-branching attributes are more likely to create pure subsets ->low entropy -> high information gain**
  - **Example: imagine using ID code as one of the attributes; it will split the training data into 1-example subsets, its IG will be high and it and will be selected as the best attribute**
- **This is likely to lead to overfitting**

Highly-Branching Attributes - Example

Weather data with ID code

ID code	outlook	temp.	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 code attribute

Highly-Branching Attributes -2

- **split based on `IDcode`:**

$$H(S_a) = -\frac{0}{1}\log_2\frac{0}{1} - \frac{1}{1}\log_2\frac{1}{1} = 0\,bits$$

...

$$H(S_m) = -\frac{0}{1}\log_2\frac{0}{1} - \frac{1}{1}\log_2\frac{1}{1} = 0\,bits$$

- **the weighted sum of entropies:**

$$H(S\,|\,IDcode) = \frac{1}{14}\cdot 0 + \ldots + \frac{1}{14}\cdot 0 = 0bits$$

- **entropy at the root**

$$H(S) = -P_{yes}\log_2 P_{yes} - P_{no}\log_2 P_{no} = -\frac{9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14} = 0.940\,bits$$

- **Gain**

$$Gain(S, Idcode) = H(S) - \sum_{v\in Values(Idcode)} \frac{|S_v|}{|S|} H(S_v) = 0.940bits$$

Gain Ratio

- *Gain ratio*: a modification of the Gain that reduces its bias towards highly branching attributes

- It takes the number and size of branches into account when choosing an attribute

- It penalizes highly-branching attributes by incorporating *SplitInformation:*

$$splitInformation(S \mid A) = -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

- SplitInformation is the entropy of S with respect to A:

- Gain ratio:

$$GainRatio(S \mid A) = \frac{Gain(S \mid A)}{SplitInformation(S \mid A)}$$

Gain Ratio

- **Computing Gain ratio for IDcode:**

$$SplitInformation(S \mid Idcode) = -\frac{1}{14}\log_2\frac{1}{14} * 14 = 3.807 bits$$

$$GainRatio(S \mid IdCode) = \frac{0.940}{3.807} = 0.246 bits$$

- **Gain was 0.94 => GainRatio is significantly lower**

- **Gain ratio for the other attributes of the weather data:**
  - **`outlook` - Gain=0.247, GainRatio=0.156**
  - **`temperature` - Gain=0.029, GainRatio=0.021**
  - **`humidity` - Gain=0.152, GainRatio=0.152**
  - **`windy` - Gain=0.048, GainRatio=0.049**

Gain Ratio (2)

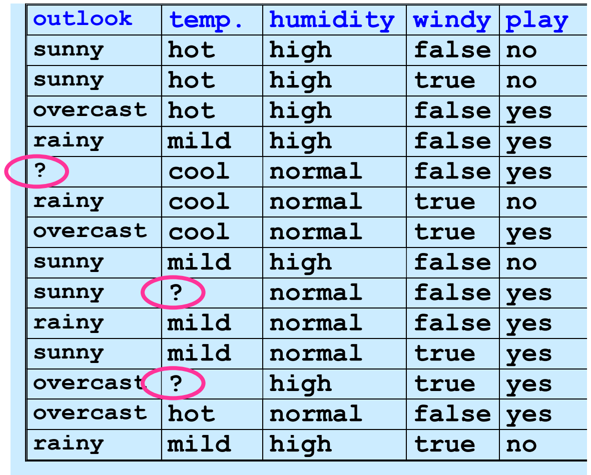
- `outlook` - Gain=0.247, GainRatio=0.156
- `temperature` - Gain=0.029, GainRatio=0.021
- `humidity` - Gain=0.152, GainRatio=0.152
- `windy` - Gain=0.048, GainRatio=0.049
- `IdCode` – Gain=0.94, GainRatio=0.246

- Using GainRatio
  - weather data without `IdCode` (the standard weather dataset):
    - GainRatio will select `outlook` (as Gain) but `humidity` is now much closer as it splits the data into 2 subsets instead of 3 as `outlook,` i.e. GainRatio penalizes `outlook`
  - weather data with `IdCode:`
    - `IDcode` will still be the selected attribute although its advantage is greatly reduced

COMP3608 only

Handling Examples with Missing Values

- **When building DT, what to do with the missing attribute values?**
- **Task: At node *n*, need to compute Gain for attribute *A* but some of the examples that have reached *n* have missing values for *A***
- **In week 5 we discussed generic approaches for dealing with missing values**
- **DTs (C4.5) uses a specific and more sophisticated method**



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
?	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	?	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	?	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

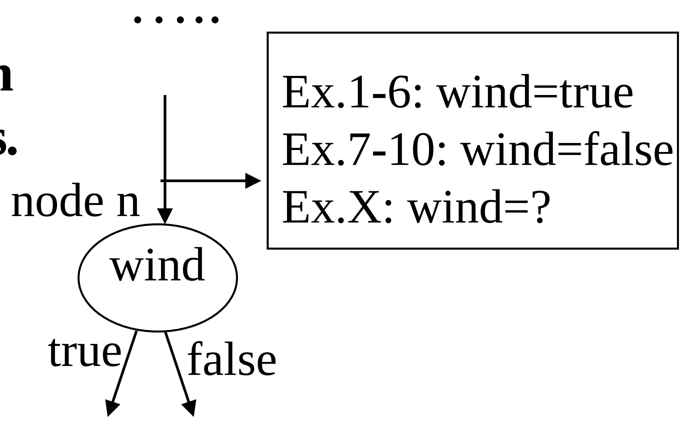
COMP3608 only

Handling Examples with Missing Values - 2

- **As we build the DT, we are at node *n* and need to calculate Gain(wind). 11 examples have reached node *n*: ex.X with a missing value for *wind* and 10 other examples without missing values. How to calculate *Gain(wind)*?**
- **Solution: 1) Assign probability to each branch of *wind* using the examples without missing values that have reached *n. 2)* Use these probabilities in *Gain(wind)* for the examples with missing values.**

⇒ P(wind=true)=0.6, P(wind=false)=0.4

⇒ For ex.X: *0.6 of ex.X* is distributed down the branch for wind=true and 0.4 *of ex.X* down the branch for wind=false



Ex.1-6: wind=true
Ex.7-10: wind=false
Ex.X: wind=?

- **These fractional examples are used to compute *Gain(wind)* and can be further subdivided at subsequent branches of the tree if another missing attribute value must be tested**
- **The same fractioning strategy can be used for classification of new examples with missing values**

Handling Attributes with Different Costs

- **Consider medical diagnosis and the following attributes:**
  - ***temperature, biopsyResult, pulse, bloodTestResult***
- **These attributes have different costs**
  - **monetary cost of the test, cost in terms of patient comfort, etc.**
- **Prefer DTs that use low-cost attributes where possible, relying on high-cost attributes only when needed to produce reliable classification**
- **How to learn a DT that contains low cost attributes?**
- **Incorporate the cost in *Gain* by penalizing attributes with high cost:**

Tan & Schlimmer

$$\frac{Gain^2(S|A)}{Cost(A)}$$

Nunez , where w $\in$ [0,1] determines cost importance

$$\frac{2^{Gain(S|A)}-1}{(Cost(A)+1)^w}$$

Components of DT

- **Model (structure)**
  - **Tree - not pre-specified but derived from data**
- **Search method (how the data is searched by the algorithm to build the structure)**
  - **2 phases: grow and prune**
  - **Growing the tree: hill climbing search guided by information gain; uses training data set**
  - **Pruning the tree: various approaches**
    - **e.g. sub-tree replacement + validation set to decide how much to prune - performs a search similar to greedy search to choose the best sub-tree to be replaced with a leaf node**

DTs - Summary

- **Very popular ML technique**
- **Easy to implement**
- **Efficient**
  - **Cost of building the tree O(*mn* log *n*), *n* instances and *m* attributes**
  - **Cost of pruning the tree with sub-tree replacement O(n)**
  - **Cost of pruning by sub-tree lifting: O(*n* (log *n*)^2)**
  - **=>total cost (building+pruning): O(*mn* log *n*) + O(*n* (log *n*)^2)**
  - **Reference: Witten, Frank, Hall and Pal p.217-218**
- **Interpretable**
  - **The output of the DT (the tree = a set of rules) is considered easier to understand by humans and use for decision making than the output of other algorithms, e.g. NNs and SVMs (if the tree is not too big)**