decision trees

alt	bar	fri	hun	pat	price	rain	res	type	est	wait?
yes	no	no	yes	some	$$$	no	yes	fre	<10	yes
yes	no	no	yes	full	$	no	no	jap	30-60	no
no	yes	no	no	some	$	no	no	fast	<10	yes
yes	no	yes	yes	full	$	no	no	jap	10-30	yes
yes	no	yes	no	full	$$$	no	yes	fre	>60	no

waiting for a table in a restaurant: goal predicate **willWait**

attributes: **alternate** (other restaurants nearby?), **bar** (bar area to wait in?), **fri/sat, hungry, patrons** (none, some, full), **price** ($, $$, $$$), **rain, reservation, type** (french, italian, japanese, fast food), **waitEstimate** (<10 min, 10-30, 30-60, >60).

The tree is a conjunction of paths ending in yes-nodes, eg.

e.g. $\forall$ r (patrons(r, full) $\wedge$ waitEstimate(r, <10) $\wedge$ hungry(r, no) $\Rightarrow$ willWait(r))

decision trees, cont.

• easy to find a **trivial** tree

for each example 1 path/branch, testing each attribute in turn. Such a tree cannot be extrapolated to new examples!!!

Occam's razor:

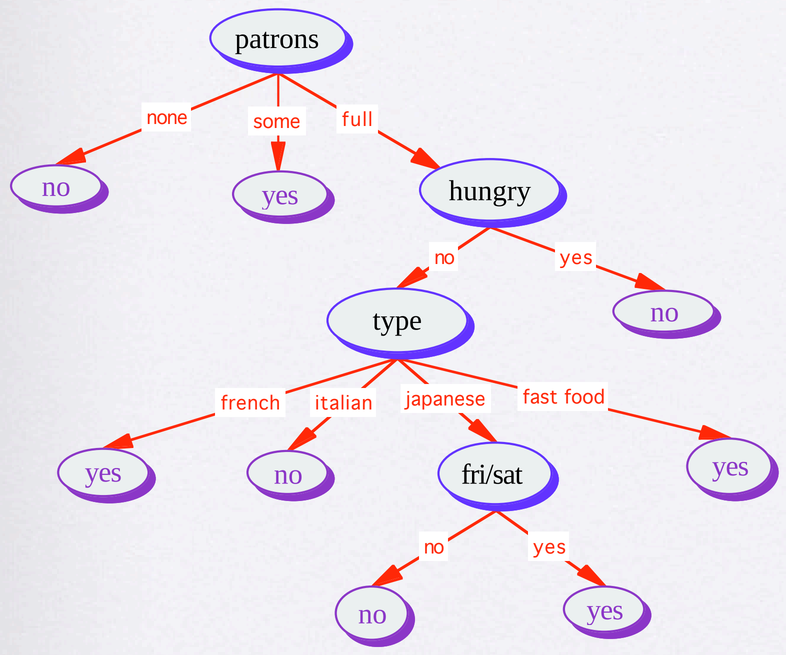
the most likely hyp is the simplest one consistent with the data

(*there are fewer simple hyp than complex ones, thus it is unlikely that <u>any</u> simple hyp consistent with many data is wildly incorrect*).

But: finding the smallest decision tree is intractable!

test most important attribute first (ie the attribute that makes the most difference to classification of an example). eg **type** is a poor attribute for root because it leaves us with 4 outcomes each of which has same # of pos and neg answers...

decision trees, cont.



this tree does not include **rain** etc because it can classify all examples without it.
example application: learning to fly a Cessna on a flight simulator. 3 pilots perform a flight plan 30 times each, 90000 examples, each with 20 state variables and action taken. Decision tree can fly plane by itself (better!)

decision trees, cont.

eg. BP uses expert system GASOIL for offshore oil platforms;
largest commercial expert system, with 2500 rules (10 man years).

Decision tree was given existing designs and resulting system was better and took 100 man days to develop.

decision trees: work well in **large** applications but representation is poor:

cannot express tests/attributes that refer to more than one object.

eg we cannot ask: is there a cheaper restaurant nearby.
Of course, we could introduce a new 1-place predicate:
CheaperRestaurantNearby, but this soon becomes intractable.

decision tree induction (DTI) for classification

- all (?) decisions / predictions are classifications
  - "should I cross the road?"
- DTI
  - searches space of decision trees
  - elaborates initially empty tree
  - easily handles disjunctive concepts
  - <u>non-incremental</u>

    uses all examples at each step
  - no backtracking
  - maintains only 1 hypothesis

    ≠ candidate elimination
  - <u>hill climbing with greedy heuristic</u>

    info gain; danger of local optimum trap
  - can handle <u>noise</u>, missing data and continuously valued attributes
  - O(n^3)

finding discriminatory tests

- choose attributes to reveal structure of domain
  - tests / attributes should be predictive
  - trees should be small

- attribute A with outcomes (values) $v_1, v_2, \ldots, v_n$ splits S={examples} into subsets $S_{v1}, S_{v2}, \ldots, S_{vn} \ni S_{vi} \subseteq S$:

  $S_{vi} = \{s \in S \mid A(s) = v_i\}$

  attribute A must be chosen <u>now</u>, without exploring subsequent divisions of the S_{vi}: **only info available is distribution of target classes in S and S_{vi}.**

- prob("random element of S $\in$ target class C_i") = **freq (C_i, S)/ |S|**

  (where freq(C_i, S) is # of cases in S that are also in C_i.)

info (in bits) conveyed by this message =

$-\log_2$ prob(message) = $-\log_2$ (freq (C_i, S)/ |S|)

amount of information

if message m is very probable for a receiver, its info content is low … if it is unexpected, it has high info content ….

$$\mathbf{I(m) = -\log(prob(m))}$$

expected info content (how much info does receiver expect from next message?):

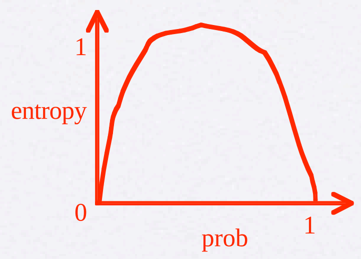
(neg)entropy = $\sum_{i=1}^{n} prob_i \, (-\log prob_i)$ =
$\mathbf{-\sum_{i=1}^{n} prob_i \, (\log prob_i)}$

expected info content (of source) also characterizes receiver's state of uncertainty

entropy

- entropy(S) = - $p_+ \log_2 p_+$ - $p_- \log_2 p_-$

where p_+ = proportion of positive examples (p/(p+n))



e.g.: |S| = 14 examples, 9+, 5-.

entropy([9+, 5-])=

$-(9/14)\log_2(9/14)-(5/14)\log_2(5/14)=0.94$

- entropy = 0 if all examples (e.g.) ∈ **pos**, i.e. $p_+=1$:

  entropy(S) = -1 log 1- 0 log 0 = 0

  entropy = 1 if there are equally many **pos** and **neg** examples

- **entropy = min average # of bits of info needed to encode classification of an arbitrary member of S**

entropy and info gain

entropy(S) = $\sum_{i=1}^{c}$ - $p_i \log_2 p_i$

= -$\sum_{i=1}^{c}$ freq(C_i, S)/|S| * $\log_2$ freq(C_i, S)/|S| if target attribute has c > 2 values.

- expected info for the tree rooted at A, **after** the value of A is known
  (note: S_v = {s S | A(s) = v})

  $$\mathbf{info_A(S) = \sum_{v \in Values(A)} |S_v|/|S| * entropy(S_v)}$$

  $info_A$(S) = info <u>after</u> S has been partitioned into subsets/branches S_v: weighted sum over subsets such that weight for branch *v* = proportion of objects in S that belong to S_v.

- expected entropy reduction due to A

  $$\mathbf{gain(S, A) = entropy(S) - info_A(S)}$$

  gain(S,A) = info about target value, given value of A = # of bits saved when encoding target value of arbitrary ex. in S, after knowing value *v* of A.

- **choose test / attribute so as to maximize gain**
  • gain is mutual info between A and S.
  • if entropy(S) is constant, max gain means min $info_A$(S) !

weather example

$S = [9+, 5-]$
entropy(S) = $-(9/14)\log_2(9/14)-(5/14)\log_2(5/14) = 0.94$

Windy has values *strong* and *weak*;
suppose: $S_{weak} = [6+, 2-]$, $S_{strong} = [3+, 3-]$

- **gain(S, Windy)** =

$$\text{entropy}(S) - \sum_{v\in\{weak,\ strong\}} |S_v|/|S| * \text{entropy}(S_v) =$$

$$\text{entropy}(S)-(8/14)\text{entropy}(S_{weak})-(6/14)\text{entropy}(S_{strong})$$

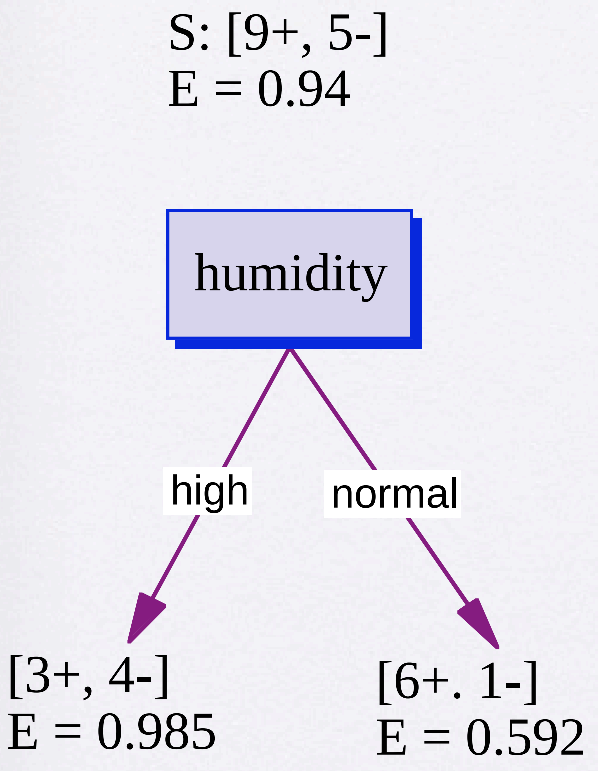
[where entropy(S_{weak}) = -(6/8) log (6/8) - (2/8) log (2/8) = 0.811]
= 0.94-(8/14)0.811-(6/14)1.0 = **0.048 bits**

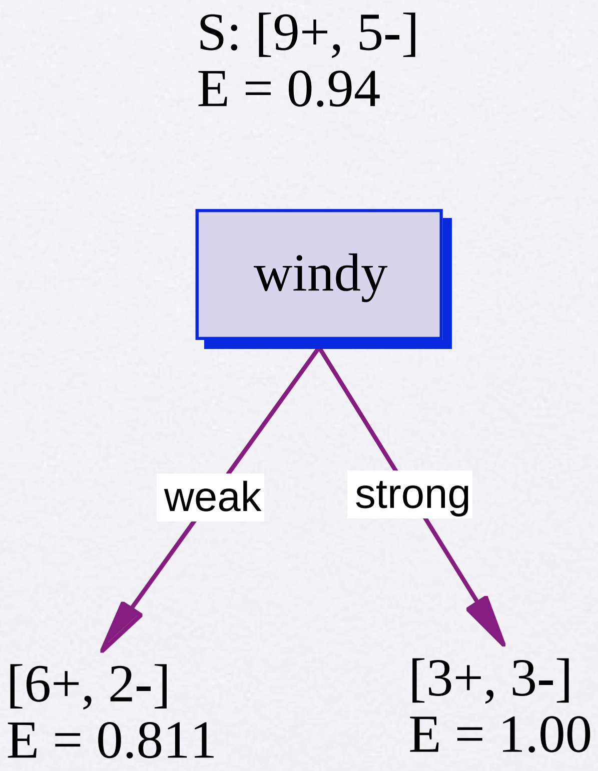
Outlook has values *sunny* (2+,3-), *overcast* (4+,0-) and *rainy* (3+,2-).
info$_{outlook}$(S) = (5/14)[-(2/5)log(2/5)-(3/5)log(3/5)] ; 5 sunny days
+(4/14)[-(4/4)log(4/4)-(0/4)log(0/4)] ; 4 overcast days
+(5/14)[-(3/5)log(3/5)-(2/5)log(2/5)] ; 5 rainy days
= **0.694 bits**

- **gain(S, Outlook)** = 0.94 - info$_{outlook}$(S) = **0.246 bits**

btw, one can compute info without working out fractions, e.g.:
-2/9 * log 2/9 - 3/9 * log 3/9 - 4/9 * log 4/9 = [-2 log 2 -3 log 3 -4 log 4 + 9 log 9] / 9

weather example, cont.

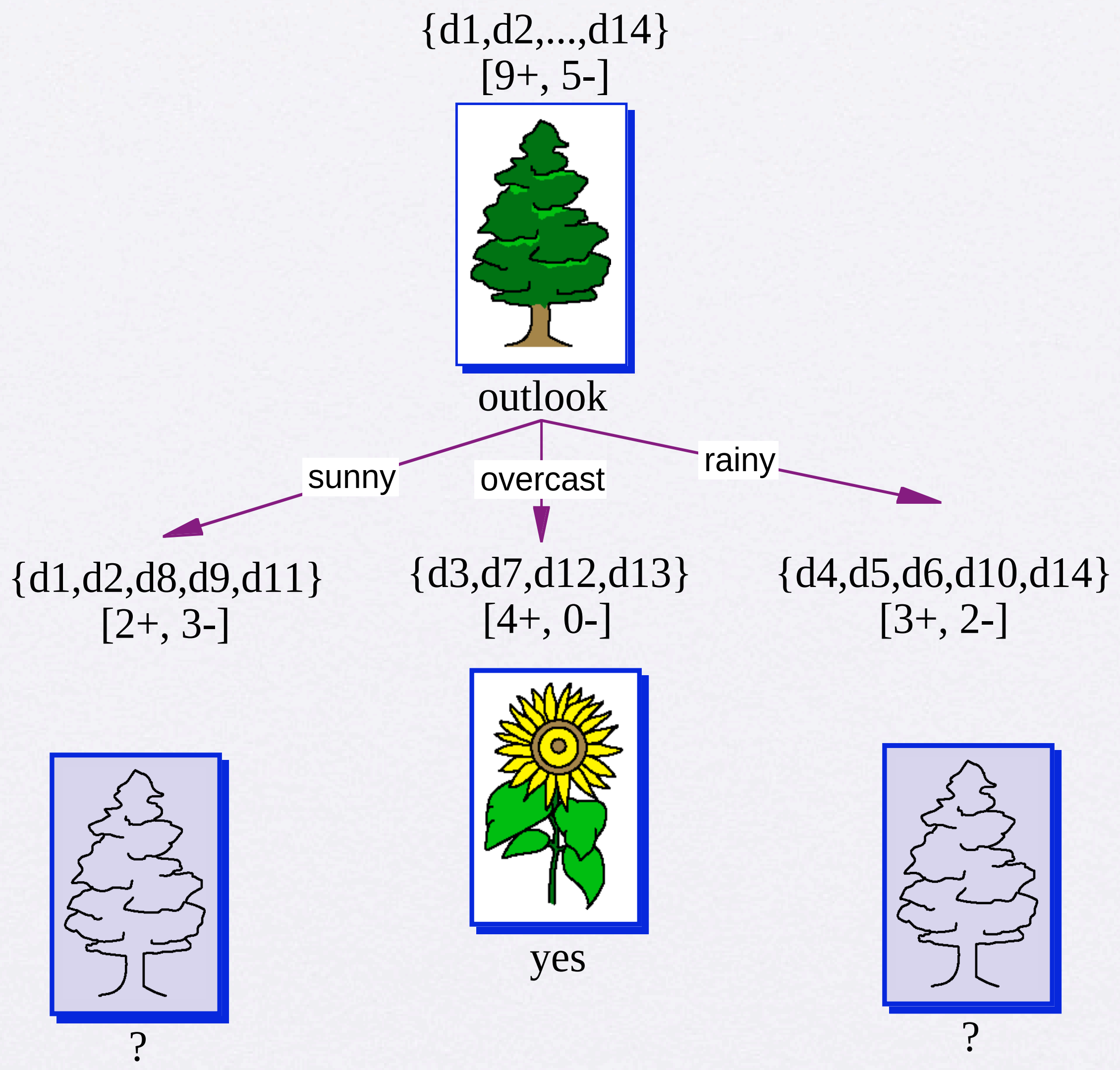




gain(S, humidity)
= .94 - (7/14).985 - (7/14).592
= .151

gain(S, windy)
= .94 - (8/14).811 -(6/14).1.0
= .048

weather example, cont.



informal DTI algorithm

```
repeat recursively
    choose attribute with max gain and make it root of subtree;
    (exclude attributes already used higher up on tree)
    make branches for each value v of A;
    sort examples down to each child node;
    (use only examples associated with this non-terminal node)
until
    each attribute has been used along this path or
    all examples associated with this leaf have same target attribute
    value.
```

$O(n^3) \approx \approx$ |training set S|* |attributes| * |non-terminal nodes| per iteration; no exp growth!

DTI algorithm

- find most important attribute test to split up the examples: each outcome is a new tree with fewer examples and one fewer attribute.
- case 1: if there are no examples left (no such example has been observed), return a default value calculated from majority classification at node's parent.
- case 2: if all remaining examples are pos (or all neg), then we are done: answer yes (or no).
- case 3: if no attributes left, but pos and neg examples, we have a problem: these examples have same description but different classification!!!
  maybe some data are incorrect: **noise**;
  maybe attributes do not fully describe situation;
  maybe situation is nondeterministic.
  ...Use a majority vote...
- case 4: if there are some pos and some neg examples, choose best attrib to split them.

DTI algorithm, cont.

```
decision-tree-learning(examples, attributes, default) returns a decision tree

if examples is empty then return default

!     else if all examples have same classification then return the classification

!     else if attributes is empty then return majority-value(examples)
!     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 := decision-tree-learning($examples_i$,
                        !!     !      attributes - best, majority-value(examples))
         !  !      add a branch to tree with label $v_i$ and subtree subtree
              end
return tree
```

highly branching attributes

info gain favors attributes with many values

e.g. with an IDCode attribute (different value for each instance),
info (needed for classific) = 0 and thus *info gain* = info at root; so IDCode is most favored attribute for splitting but is totally useless for prediction.
e.g. another bad attr: Date (with values à la 9/17/)

info gain may lead to complex trees!
replace *info gain* by ***gain ratio***

gain ratio(S,A) = info gain(S, A) / split info(S, A)

split info(S, A) $= -\sum_i |S_i| / |S| \log_2 |S_i| / |S|$

where S_i *is subset of S for which A has value* v_i

- consider # and size of child nodes produced by attribute, ignoring other info about classes
- split info is large for highly branching attributes

highly branching attributes, cont.

for **IDCode attribute**, all counts = 1 and so info value of split *info([1,1,…1])*
= -1/14 * log 1/14 * 14 (because 1/14 occurs 14 times) = 3.807 bits
(0.94/3.807=0.246 … still preferred but less so…)

- ***split info*** = # of bits needed to determine which branch each instance is assigned to -- grows with # of branches
(quick & dirty: use # values as weight..)

- gain ratio = info gain / split info
   outlook: splits data into 3 subsets of size 5,4,5; so
   split info([5,4,5]) = -5/14*log5/14-4/14*log4/14-5/14*log5/14 = 1.577,
   info gain = 0.940 - 0.693 = 0.247,
   gain ratio = 0.247/1.577 = 0.156

   humidity comes close now because its split info([7,7]) = 1.0….

- pick attr with max gain ratio **if** its info gain ≥ average info gain for all examined attributes

inductive bias of DTI

- inductive bias
  assumptions BK s.t. BK ∧ data (deductively?) justify learner's classification of unseen data

- DTI efficiently approximates DTI'
  • DTI' considers **-breadth-first-** <u>all</u> trees of depth 1, then <u>all</u> trees of depth 2 etc; returns smallest consistent tree found at earliest depth.
  • DTI uses greedy heuristic (info gain) instead of breadth-first, and is not guaranteed to find the simplest tree.

- inductive bias of DTI ≈ Occam's razor
  • prefer shorter, simpler trees…
  • prefer trees that have high info-gain attributes closer to the root

DTI with numeric attributes

2-way splits for numeric attrs

temp: 64 65 68 69 70 71 72 75 80 81 83 85
(11 breakpoint positions)

compute info gain as usual:
temp < 71.5: 4 yes, 2 no;

temp > 71.5: 5 yes, 3 no;

i.e. info = 6/14 * info(4,2) + 8/14 * info(5,3) = .939 bits

for reals, test against an **interval**...

since splits are binary, a numeric attr - unlike a nominal one, may be tested several times

may lead to ugly trees

missing values

1. treat missing values as a real further attr value
   - this may make missing values too significant
   - makes trees ugly
2. ignore all instances with some missing values
   - this loses too much info (maybe attr with missing value is not even needed in tree!)
   - ok if there are only very few
3. count how many items go down each branch and use most popular branch if value for a test instance is missing
4. notionally split instance into pieces; weight branches in proportion to their popularity and send pieces down; use weights to compute gain..

attributes with different costs

suppose each test of A costs $1000.- …
how to learn a good tree with min expected cost?

replace *gain* by

$$\frac{gain^2(S,A)}{Cost(A)}$$

or

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

where $w \in [0,1]$ is importance of cost

how to avoid overfitting

- stop growing when data split is not statistically significant ("pre-pruning") or
- grow full tree, then **post-prune**
- How to select "best" tree:

  Measure performance over training data
  Measure performance over separate validation data set
  MDL (Minimal Description Length):
  minimize
  size(tree) +size(misclassifications(tree))

- how to use available data

  Training set: learn the tree
  Validation set: prune the tree
  Test set: estimate future classification accuracy

  Best if they're independent but if data are limited, maybe best to overlap ...

reduced error pruning

Split data into training and validation set
Do until further pruning is harmful:

1. Evaluate impact on validation set of pruning each possible node (plus those below it)
2. Greedily remove the one that most improves validation set accuracy

    produces smallest version of most accurate subtree.

Or

1. Convert tree to equivalent set of rules
2. Prune each rule independently of others
3. Sort final rules into desired sequence for use

    Perhaps most frequently used method

What if data is limited? use C5 hack….

tree pruning

- **postpruning - decreases accuracy on training set but may increase accuracy on test sets!**

subtree replacement
bottom up;
replaces internal nodes by leafs, based on **error rates**
subtree raising
replaces internal node by node below, based on **error rates** ;
requires reclassification of children of eliminated node;
expensive - usually applied only to subtree of most popular branch
prune if estimated error rate of replacement < error rate of original node

error rates (at nodes) should be estimated with *independent* test set but C5 uses training data themselves!

error rates (E, N from training data!!)

consider {instances} reaching each node and <u>assume majority class represents the node</u>: this gives E "errors" out of N instances. Assume the N instances are generated by Bernoulli process with true prob of error *q*. Given a confidence *c* (say, c=25%), find confidence limits *z* s. t.

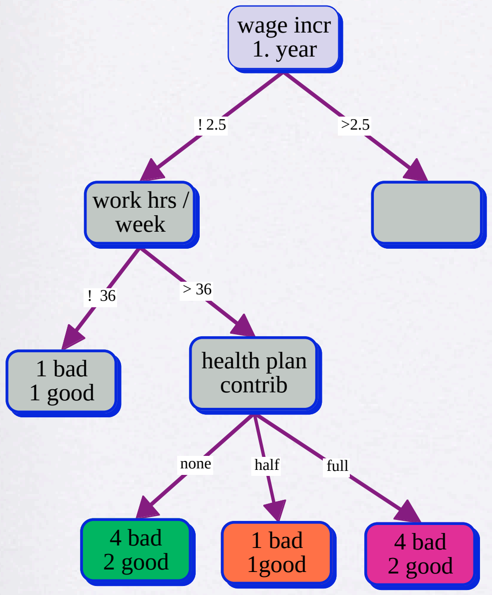
$$\Pr\left[\frac{E/N - q}{\sqrt{q(1-q)/N}} > z\right] = c$$

E/N is observed error rate. Use upper confidence limit for *q* as pessimistic estimate for error rate *e* at the node:

$$e = \frac{E/N + \frac{z^2}{2N} + z\sqrt{\frac{E/N}{N} - \frac{(E/N)^2}{N} + \frac{z^2}{4N^2}}}{1 + \frac{z^2}{N}}$$

z = # of standard deviations corresponding to *c*
(*z*=0.69 for *c*=25%)

error rates, cont.



majority class = bad, thus **E=2**, N=6, E/ N=0.33, thus *e* = 0.47.
Training set error rate for this leaf = 0.33,
we use estimate 47%.

E=1, N=2, *e* = 0.72!!!.
same as leftmost leaf.

combined error estimate of the 3 children, taking # of covered instances into account (ratio 6:2:6), is 0.51. (0.47/6 + 0.72/2 + 0.47/6 = 0.0783 + 0.36 + 0.0783 = 0.51)
Error estimate of parent (health plan contrib): 9 bad (majority class), 5 good, so training set error rate = 5/14; so *e* = 0.46.
So the children are pruned away!!! etc etc up the tree….

estimated errors exceed sometimes the max of 0.5!!!! this shows that the approach is statistically shaky: assumes normal distribution, using upper confidence limit; use of statistics from training set… works well in practice...
Ratio 6:2:6: 0.47/6 = 0.0783 + 0.72/2=0.36 + 0.47/6 = 0.0783 = 0.51.

disadvantages of decision trees

many identical samples have no more effect than one

can't handle contradictory samples (well) …

    but then, who can?

with noisy data, tree gets bushy, overfitting;
worst case: 1 leaf per training instance

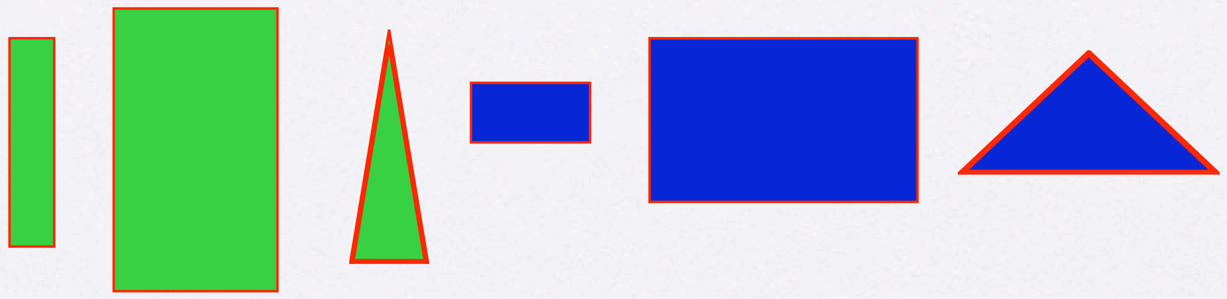
    thus: don't aim for completely correct rules, e.g. don't subdivide a group with < k% of data

expressively poor description language

    feature vector: Vbl > Constant, Vbl = Constant, …

    can't express relations, connectives

re: poor expressiveness



learn *lying* vs *standing*, given **width** and **height** attributes..

the system would learn **bad** rules like
! **if width " 7.5 and height # 4 then lying** ….
Such rules would fail on new data!!

wanted instead: ***relations between attributes***
if width > height then lying ….
not: widthGreaterThanHeight as a new predicate!!!!!

or better still:
! **if width(block) > height(block) then lying(block)**