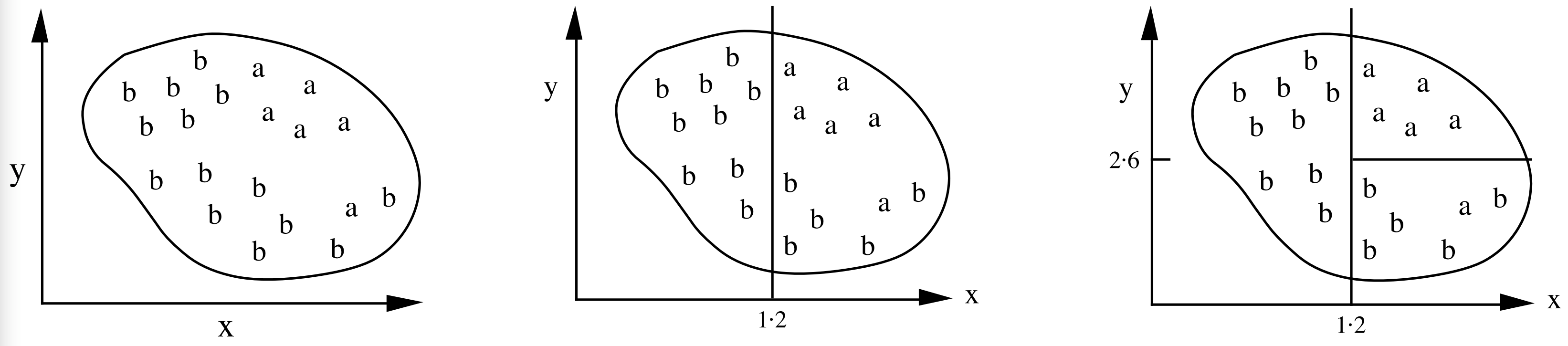
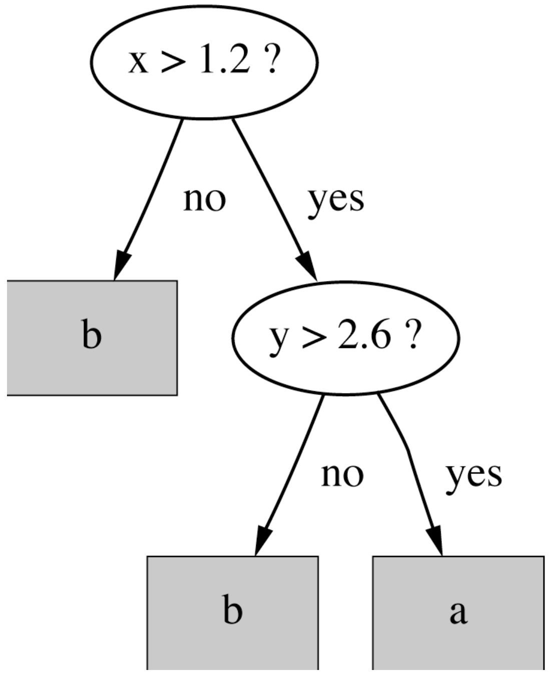


(a)

(b)



(a) covering algorithm; (b) decision tree

covering algorithms make rules, considering each class in turn

- add tests to rule under construction, aiming for a rule with max **accuracy**
  - unlike decision trees (pick attr that max info gain), pick attr-val pair that max prob of desired classification
  - *tot* = total # of instances covered by rule
  - *pos* = # of pos instances of class under this rule (rule makes *tot-pos* errors)
- maximize *pos / tot*

  if ? then recommendation = hard

age = young	2/8	(are correct for hard)
age = presbyopic	1/8	
astigmatism = yes	4/12 …..	

  if astigmatism = yes then recommendation = hard

continue covering, rule refining...

if astigmatism = yes then recommendation = hard

covers only 4 cases….

if astigmatism = yes and ? then recommendation = hard

age = young	2/4	(are still correct for hard)
age = presbyopic	1/4	
tear production rate = normal	4/6 ….	

if astigmatism = yes and tear production rate = normal
then recommendation = hard

…..

now consider rules for next class ….

a simple rule learner (separate-and-conquer)

```
for each class c
  e := {full set of instances};
  while e contains instances in c
     create rule r with empty LHS, predicting c;
     while r is not perfect and there are attrs
       for each attr a not in r, and each val v,
       consider adding a=v to LHS of r;
       select a, v to maximize accuracy pos/tot;
      (break ties by picking cond with max pos)
       add a=v to r;
     remove instances covered by r from e;
```

rules for each class need not be considered in order although instances covered by a rule are removed when rule is completed...

criteria for choosing a test to be added

- *pos/tot*
  - maximizes rule correctness
  - max: no neg examples covered
    - prefer test covering 1 pos, 0 neg, over test with 100 pos, 1 neg
  - finds & eliminates special cases first

- info gain measure (accuracy increase)

  pos [log(*pos/tot*) - log(*Pos/Tot*)]

  Pos, Tot: # of data covered by rule **before** new test added

  - maximizes pos coverage, minimizes neg coverage
  - finds hi-coverage, general rules first, exceptions later

making sensible rules - avoid overfitting

if astigmatism = yes and tear prod rate = normal then *hard*

correct 4 / 6 cases; default rule (always recommends *hard*): 4 / 24;

so rule greatly improves accuracy, with small (0.14%) prob of improvement being due to chance

if astigmatism = yes then *hard*

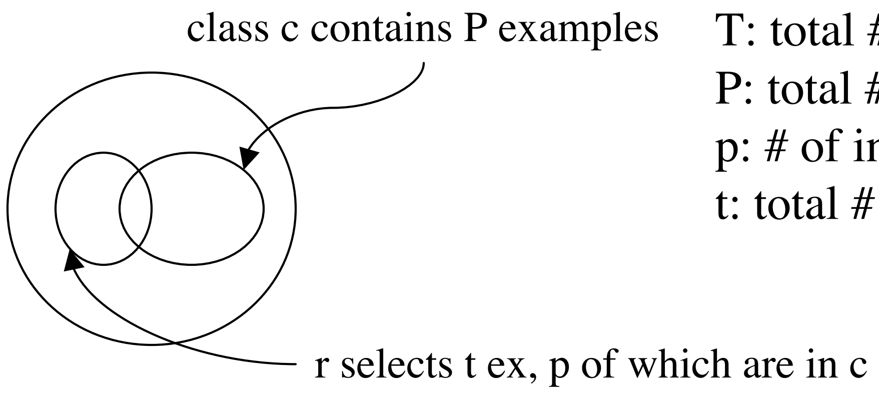
improves accuracy from 4/24 to 4/12, with 4.7% prob of improvement being due to chance (not a very good rule)

1. make a perfect rule, adding tests until full correctness
2. tentatively remove last test added, see if improvement prob being due to chance is small; if not, remove test; continue…

decision lists (rule order matters)

```
e := {instances};
while e not empty do
    for each class c for which e has an instance
          use basic covering alg to create best rule for c;
         *compute prob measure m(r) for rule r, and for rule with last
          test omitted, m(r-);
          while m(r-) < m(r) do remove last test from rule, repeat *;
    from (best for class) rules generated, pick one with min m(r);
    output the rule r;
    remove instances covered by rule r from e;
endwhile
// accuracy measure for growing, prob measure for pruning rules
// does not consider classes one by one but makes rule for every class and picks best one...
// m(r-) < m(r): since r- is better (less likely due to chance) than r (with test), just drop the test.
```

probability measure for rule evaluation


class c contains P examples
r selects t ex, p of which are in c

T: total # of instances in data set;
P: total # of instances of class c in data set;
p: # of inst. of class c selected by r;
t: total # of inst. selected by r;

e.g. r has accuracy 4/6 (p/t), default rule has accuracy 4/24 (P/T).

what's the probability that a random rule **with same coverage** as r has an accuracy improvement as good as (or better than) r?

use hypergeometric prob distribution
(assume selection without replacement)

prob measure for rule evaluation, cont.

prob [of t cases selected at random, exactly i are in class c] =

$$\frac{\binom{P}{i}\binom{T-P}{t-i}}{\binom{T}{t}}$$

m(r) can be used to compare 2 rules or find best pruned version

prob that random rule will do as well or better than r is

$m(r) = \sum_{i=p}^{\min(t,P)}$ prob [of t cases selected at random, exactly i are in class c]

small values of m(r) are good: it's unlikely this rule would have happened by chance; there are several cheaper approximations to m(r) ...

prob measure for rule evaluation, cont.

assuming large T, and sampling with replacement (i.e. assuming constant prob P/T that an instance is in c):

prob [of t cases selected at random, exactly i are in class c] =

$$\binom{t}{i}\left(\frac{P}{T}\right)^{i}\left(1-\frac{P}{T}\right)^{t-i}$$

use this in computing
$m(r) = \sum_{i=p}^{\min(t,P)}$ prob [of t cases selected at random, exactly i are in class c]

there are some still cheaper approximations to this approximation to m(r) ...

incremental reduced-error pruning

split training data into Grow set and Prune set, make a good rule using rule covering, delete a test and try truncated rule on Prune set, see if it's better…. repeat for each class …

disadvantage: fewer instances available for training, some instances might only be in Prune set, small Prune set might wrongly prefer a rule…

mitigate by **resplitting** into Grow and Prune …

test rule quality by m(r) or with simpler measures such as:

1. use accuracy p/t. Bad because it prefers rule that gets 1 instance right out of coverage of 1 to rule that gets 1000 right out of 1001
2. use (p - n)/t. Same problem.
3. use (p + (N - n)) / T where n = t-p (neg covered), N=T-P (total # of neg instances), N-n not covered neg instances. I.e. how well would rule discriminate predicted class if it were the only rule? Bad because it treats noncoverage of neg instances as equally important as coverage of pos instances; misleading in context of many other rules!

so stick with m(r) ….

incremental reduced-error pruning, cont.

```
e := {instances};
while e is not empty do
{ split e into Grow and Prune in ratio 2:1;
for each class c for which Grow and Prune both contain an instance
    use basic covering algorithm to make best 'perfect' rule for c;
    • compute worth w(r) for r and w(r-) [r without last test] on Prune;
    while w(r-) > w(r) remove last condition from r and repeat •;
from all rules made, select the rule with max w(r) (or min m(r));
output r;
e := e - {instances covered by r};
}
// worth could be 1 - m(r) …. or ….
```

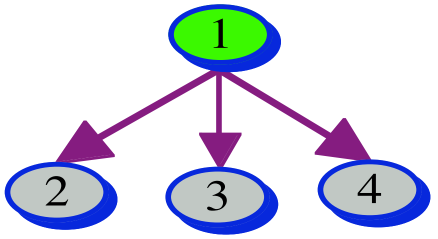
another method for getting rules

- **combine** DTI and rule covering
- DTI part
  - make a rule by building a **partial** pruned decision tree on current instances
    - it's partial because some branches may be left unexplored
  - turn leaf with largest coverage into **one** rule
    - this may produce a simple and general rule
  - discard tree
    - if data are relatively noise-free, only one path has to be built
- rule covering part
  - as in basic rule covering algorithm, once a rule is made, remove instances covered by it
    … until no more instances remain

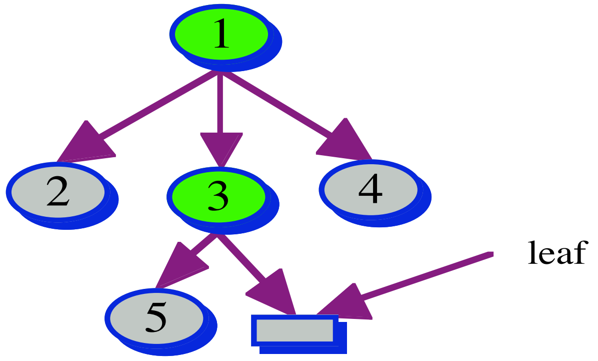
combining DTI and rule covering

```
expandSubset(s):
choose test T to split examples into subsets (using info gain);
sort subsets by increasing average entropy (low-entropy subsets may lead to small subtrees and general rules; later subsets may not be expanded …);
while there is unexpanded subset t' & all expanded subsets are leaves
    expandSubset(t');
if all expanded subsets are leaves & estimated error for subtree ≥ error for node, undo expansion into subsets & make node into leaf (i.e. subtree replacement);
```

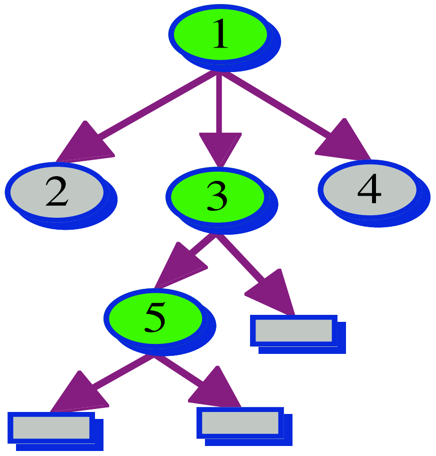
after replacement, continue backtracking with siblings of replaced node; if an inner node has all children non-leaves (i.e. when subtree replacement is not done), leave remaining subsets unexplored. From this partial tree, extract a single rule for the leaf that covers most instances



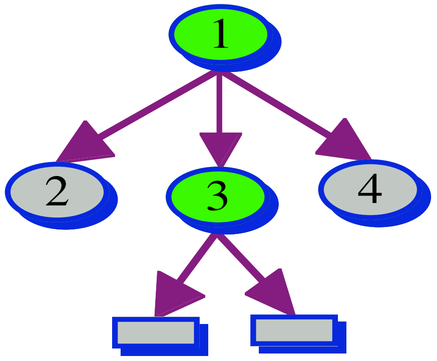
expand 3
because of low
entropy



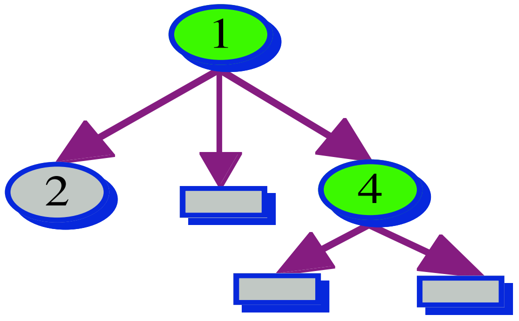
leaf has lower entropy but can't be
expanded; so backtrack and expand 5:



all children of 5 are
leaves, thus try and
accept pruning:



now consider 3 for
replacement, and accept;
backtrack to expand 4:



suppose 4 is not replaced.
Each leaf could be a rule. We
pick leaf with max coverage
to make **one** rule, then
discard the tree!

labor data: pruned tree vs decision list

```
wage-increase-first-year <= 2.5: bad
wage-increase-first-year > 2.5
|   statutory-holidays <= 10: bad
|   statutory-holidays > 10: good
```

```
1. wage-increase-first-year > 2.5 AND
longterm-disability-assistance = yes AND
statutory-holidays > 10: good
2. wage-increase-first-year <= 4 AND
working-hours > 36: bad
3. : good
```

iris data: tree vs decision list

```
petalwidth <= 0.6: Iris-setosa
petalwidth > 0.6
|   petalwidth <= 1.7
|   |   petallength <= 4.9: Iris-versicolor
|   |   petallength > 4.9
|   |   |   petalwidth <= 1.5: Iris-virginica
|   |   |   petalwidth > 1.5: Iris-versicolor
|   petalwidth > 1.7: Iris-virginica
```

```
1. petalwidth <= 0.6: Iris-setosa
2. petalwidth <= 1.7 AND
   petallength <= 4.9: Iris-versicolor
3. : Iris-virginica
```