

# Machine Learning

## Chapter 6. Implementations

# 6

## Implementation: Real machine learning schemes

- ❖ Decision trees: from ID3 to C4.5
  - ❑ Pruning, missing values, numeric attributes, efficiency
- ❖ Decision rules: from PRISM to Induct and PART
  - ❑ Missing values, numeric attributes, computing significance, rules with exceptions
- ❖ Extended linear classification: support vectors
  - ❑ Non-linear boundaries, max margin hyperplane, kernels
- ❖ Instance-based learning
  - ❑ Speed up, combat noise, attribute weighting, generalized exemplars
- ❖ Numeric prediction
  - ❑ Regression/model trees, locally weighted regression
- ❖ Clustering: hierarchical, incremental, probabilistic
  - ❑ K-means, heuristic, mixture model, EM, Bayesian

# Industrial-strength algorithms

- ❖ For an algorithm to be useful in a wide range of real-world applications it must:
  - ❑ Permit numeric attributes
  - ❑ Allow missing values
  - ❑ Be robust in the presence of noise
  - ❑ Be able to approximate arbitrary concept descriptions (at least in principle)
- ❖ Basic schemes need to be extended to fulfill these requirements



# Decision trees

## ❖ Extending ID3:

- ❑ to permit numeric attributes: *straightforward*
- ❑ to dealing sensibly with missing values: *trickier*
- ❑ stability for noisy data:  
*requires pruning mechanism*

## ❖ End result: C4.5 (Quinlan)

- ❑ Best-known and (probably) most widely-used learning algorithm
- ❑ Commercial successor: C5.0

# Numeric attributes

- ❖ Standard method: binary splits
  - ❑ E.g.  $\text{temp} < 45$
- ❖ Unlike nominal attributes, every attribute has many possible split points
- ❖ Solution is straightforward extension:
  - ❑ Evaluate info gain (or other measure) for every possible split point of attribute
  - ❑ Choose “best” split point
  - ❑ Info gain for best split point is info gain for attribute
- ❖ Computationally more demanding

# Weather data (again!)

Outlook	Temperature	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	Normal	False	Yes
...				

Outlook	Temperature	Humidity	Windy	Play
Sunny	85	85	False	No
Sunny	80	90	True	No
Overcast	83	86	False	Yes
Rainy	75	80	False	Yes
...	...	...	...	...

If outlook = sunny and humidity > 83 then play = no
If outlook = rainy and windy = true then play = no
If outlook = overcast then play = yes
If humidity = normal then play = yes
If none of the above then play = yes

# Example

❖ Split on temperature attribute:

64	65	68	69	70	71	72	72	75	75	80	81	83	85
Yes	No	Yes	Yes	Yes	No	No	Yes	Yes	Yes	No	Yes	Yes	No

❑ E.g. temperature < 71.5: yes/4, no/2  
temperature ≥ 71.5: yes/5, no/3

❑  $\text{Info}([4,2],[5,3])$   
=  $6/14 \text{ info}([4,2]) + 8/14 \text{ info}([5,3])$   
= 0.939 bits

- ❖ Place split points halfway between values
- ❖ Can evaluate all split points in one pass!

# Avoid repeated sorting!

- ❖ Sort instances by the values of the numeric attribute
  - ❑ Time complexity for sorting:  $O(n \log n)$
- ❖ Does this have to be repeated at each node of the tree?
- ❖ No! Sort order for children can be derived from sort order for parent
  - ❑ Time complexity of derivation:  $O(n)$
  - ❑ Drawback: need to create and store an array of sorted indices for each numeric attribute

# Binary vs multiway splits

- ❖ Splitting (multi-way) on a nominal attribute exhausts all information in that attribute
  - ❑ Nominal attribute is tested (at most) once on any path in the tree
- ❖ Not so for binary splits on numeric attributes!
  - ❑ Numeric attribute may be tested several times along a path in the tree
- ❖ Disadvantage: tree is hard to read
- ❖ Remedy:
  - ❑ pre-discretize numeric attributes, *or*
  - ❑ use multi-way splits instead of binary ones

# Computing multi-way splits

- ❖ Simple and efficient way of generating multi-way splits: greedy algorithm
- ❖ Dynamic programming can find optimum multi-way split in  $O(n^2)$  time
  - $\text{imp}(k, i, j)$  is the impurity of the best split of values  $x_i \dots x_j$  into  $k$  sub-intervals
  - $\text{imp}(k, 1, i) = \min_{0 < j < i} \text{imp}(k-1, 1, j) + \text{imp}(1, j+1, i)$
  - $\text{imp}(k, 1, N)$  gives us the best  $k$ -way split
- ❖ In practice, greedy algorithm works as well

# Missing values

- ❖ Split instances with missing values into pieces
  - ❑ A piece going down a branch receives a weight proportional to the popularity of the branch
  - ❑ weights sum to 1
- ❖ Info gain works with fractional instances
  - ❑ use sums of weights instead of counts
- ❖ During classification, split the instance into pieces in the same way
  - ❑ Merge probability distribution using weights

# Pruning

- ❖ Prevent overfitting to noise in the data
- ❖ “Prune” the decision tree
- ❖ Two strategies:
  - *Postpruning*  
take a fully-grown decision tree and discard unreliable parts
  - *Prepruning*  
stop growing a branch when information becomes unreliable
- ❖ 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

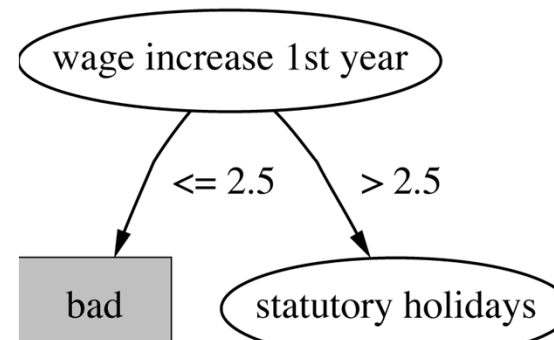
	a	b	class
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0

- ❖ 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
  - ❑ Structure is only visible in fully expanded tree
  - ❑ Prepruning won't expand the root node
- ❖ But: XOR-type problems rare in practice
- ❖ And: prepruning faster than postpruning

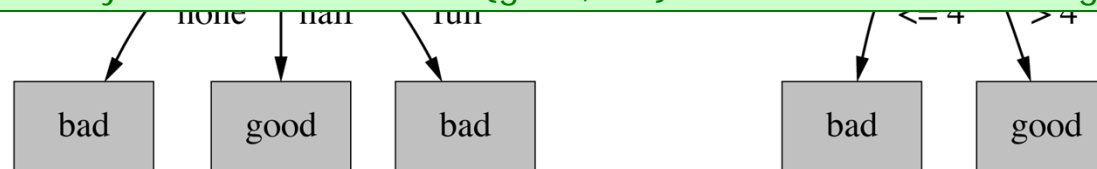
# Postpruning

- ❖ First, build full tree
- ❖ Then, prune it
  - ❑ Fully-grown tree shows all attribute interactions
- ❖ Problem: some subtrees might be due to chance effects
- ❖ Two pruning operations:
  - ❑ *Subtree replacement*
  - ❑ *Subtree raising*
- ❖ Possible strategies:
  - ❑ error estimation
  - ❑ significance testing
  - ❑ MDL principle

# Subtree replacement

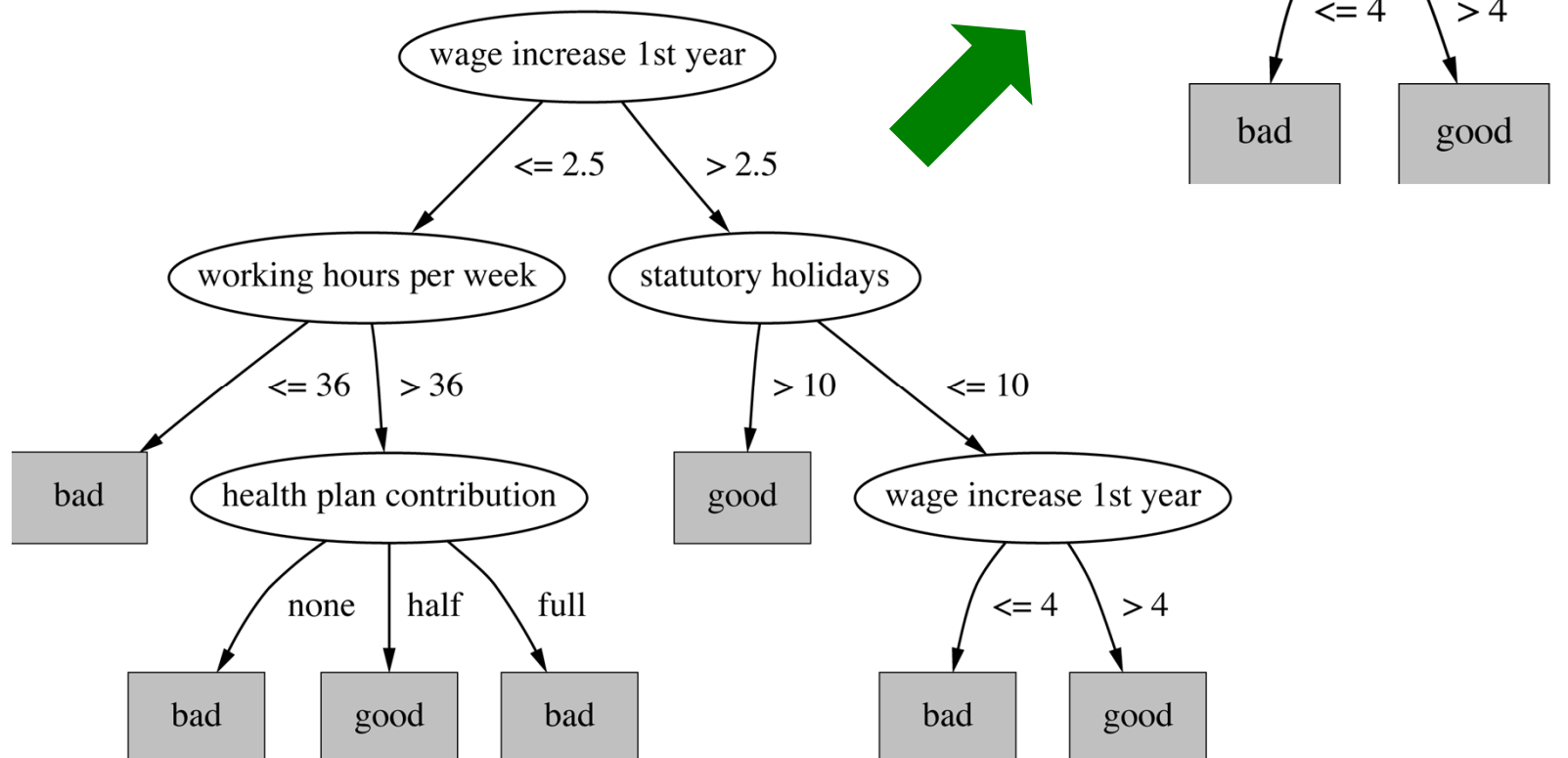


Attribute	Type	1	2	3	...	40
Duration	(Number of years)	1	2	3		2
Wage increase first year	Percentage	2%	4%	4.3%		4.5
Wage increase second year	Percentage	?	5%	4.4%		4.0
Wage increase third year	Percentage	?	?	?		?
Cost of living adjustment	{none,tcf,tc}	none	tcf	?		none
Working hours per week	(Number of hours)	28	35	38		40
Pension	{none,ret-allw, empl-cntr}	none	?	?		?
Standby pay	Percentage	?	13%	?		?
Shift-work supplement	Percentage	?	5%	4%		4
Education allowance	{yes,no}	yes	?	?		?
Statutory holidays	(Number of days)	11	15	12		12
Vacation	{below-avg,avg,gen}	avg	gen	gen		avg
Long-term disability assistance	{yes,no}	no	?	?		yes
Dental plan contribution	{none,half,full}	none	?	full		full
Bereavement assistance	{yes,no}	no	?	?		yes
Health plan contribution	{none,half,full}	none	?	full		half
Acceptability of contract	{good,bad}	bad	good	good		good



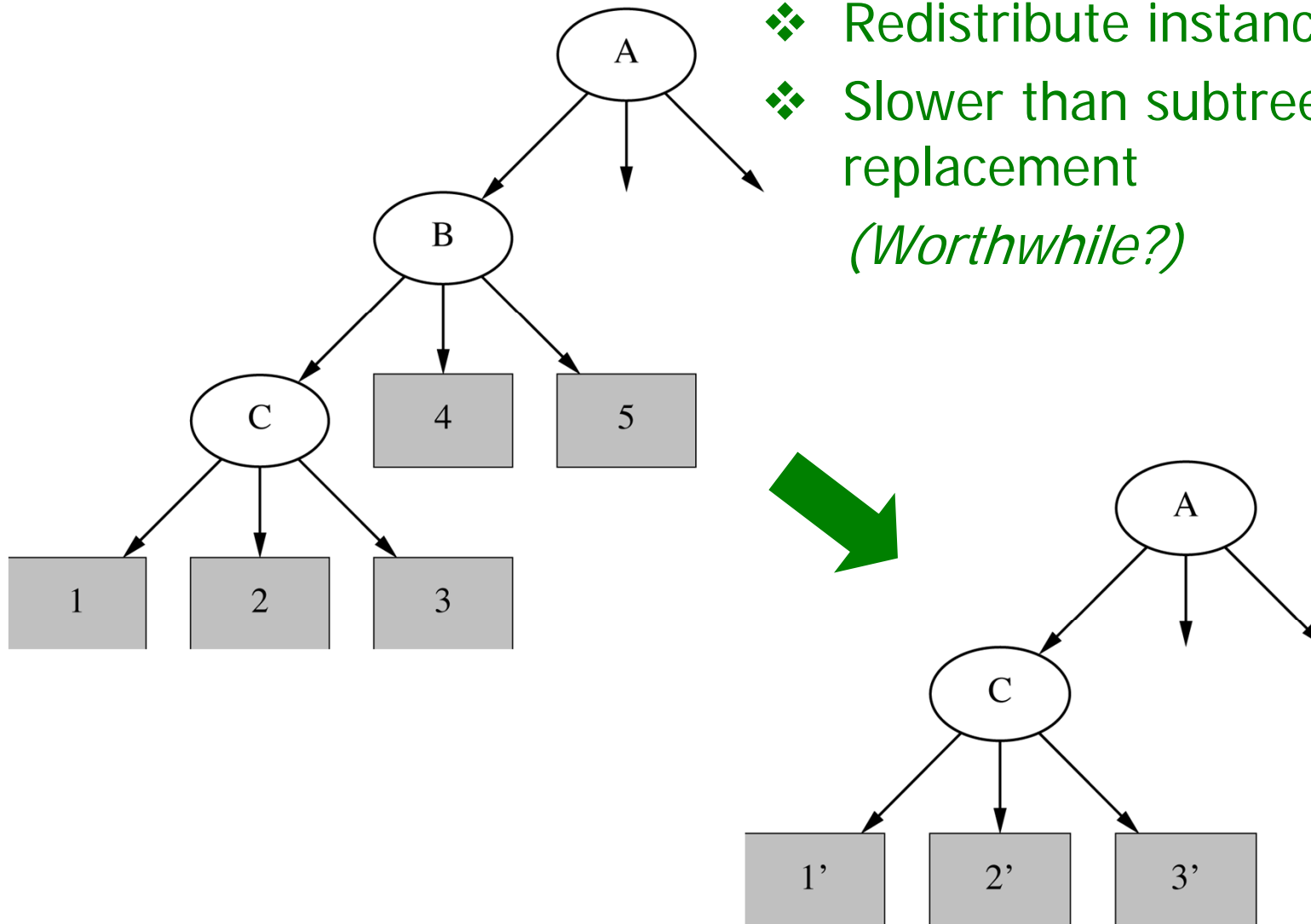
# Subtree replacement

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



# Subtree raising

- ❖ Delete node
- ❖ Redistribute instances
- ❖ Slower than subtree replacement  
*(Worthwhile?)*



# Estimating error rates

- ❖ Prune only if it reduces the estimated error
- ❖ Error on the training data is NOT a useful estimator  
*(would result in almost no pruning)*
- ❖ Use hold-out set for pruning  
("reduced-error pruning")
- ❖ C4.5's method
  - ❑ Derive confidence interval from training data
  - ❑ Use a heuristic limit, derived from this, for pruning
  - ❑ Standard Bernoulli-process-based method
  - ❑ Shaky statistical assumptions (based on training data)

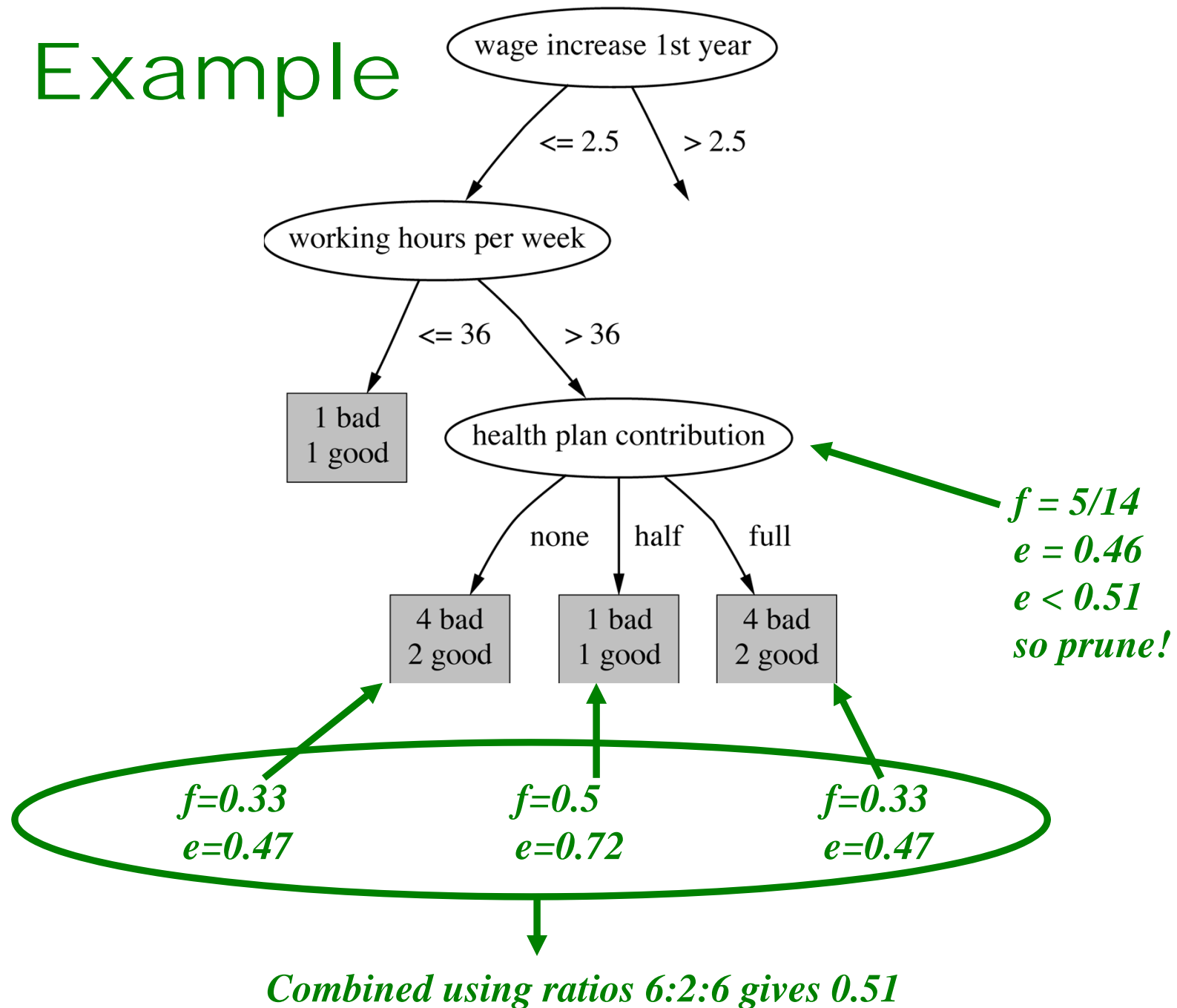
# C4.5's method

- ❖ Error estimate for subtree is weighted sum of error estimates for all its leaves
- ❖ Error estimate for a node:

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

- ❖ If  $c = 25\%$  then  $z = 0.69$  (from normal distribution)
- ❖  $f$  is the error on the training data
- ❖  $N$  is the number of instances covered by the leaf

# Example



# Complexity of tree induction

## ❖ Assume

- ❑  $m$  attributes
- ❑  $n$  training instances
- ❑ tree depth  $O(\log n)$

❖ Building a tree  $O(m n \log n)$

❖ Subtree replacement  $O(n)$

❖ Subtree raising  $O(n (\log n)^2)$

- ❑ Every instance may have to be redistributed at every node between its leaf and the root

- ❑ Cost for redistribution (on average):  $O(\log n)$

❖ Total cost:  $O(m n \log n) + O(n (\log n)^2)$

# From trees to rules

- ❖ Simple way: one rule for each leaf
- ❖ C4.5rules: greedily prune conditions from each rule if this reduces its estimated error
  - ❑ Can produce duplicate rules
  - ❑ Check for this at the end
- ❖ Then
  - ❑ look at each class in turn
  - ❑ consider the rules for that class
  - ❑ find a “good” subset (guided by MDL)
- ❖ Then rank the subsets to avoid conflicts
- ❖ Finally, remove rules (greedily) if this decreases error on the training data

# C4.5: choices and options

- ❖ C4.5rules slow for large and noisy datasets
- ❖ Commercial version C5.0rules uses a different technique
  - ❑ Much faster and a bit more accurate
- ❖ C4.5 has two parameters
  - ❑ Confidence value (default 25%):  
lower values incur heavier pruning
  - ❑ Minimum number of instances in the two most popular branches (default 2)

# Discussion

## *TDIDT: Top-Down Induction of Decision Trees*

- ❖ The most extensively studied method of machine learning used in data mining
- ❖ Different criteria for attribute/test selection rarely make a large difference
- ❖ Different pruning methods mainly change the size of the resulting pruned tree
- ❖ C4.5 builds *univariate* decision trees
- ❖ Some TDITDT systems can build *multivariate* trees (e.g. CART)



# Classification rules

- ❖ Common procedure: *separate-and-conquer*
- ❖ Differences:
  - ❑ Search method (e.g. greedy, beam search, ...)
  - ❑ Test selection criteria (e.g. accuracy, ...)
  - ❑ Pruning method (e.g. MDL, hold-out set, ...)
  - ❑ Stopping criterion (e.g. minimum accuracy)
  - ❑ Post-processing step
- ❖ Also: Decision list
  - VS.
  - one rule set for each class

# Test selection criteria

- ❖ Basic covering algorithm:
  - ❑ keep adding conditions to a rule to improve its accuracy
  - ❑ Add the condition that improves accuracy the most
- ❖ Measure 1:  $p/t$ 
  - ❑  $t$  total instances covered by rule  
 $p$  number of these that are positive
  - ❑ Produce rules that don't cover *negative* instances, as quickly as possible
  - ❑ May produce rules with very small coverage —special cases or noise?
- ❖ Measure 2: Information gain  $p (\log(p/t) - \log(P/T))$ 
  - ❑  $P$  and  $T$  the positive and total numbers before the new condition was added
  - ❑ Information gain emphasizes positive rather than negative instances
- ❖ These interact with the pruning mechanism used

# Missing values, numeric attributes

- ❖ Common treatment of missing values:  
*for any test, they fail*
  - ❑ Algorithm must either
    - use other tests to separate out positive instances
    - leave them uncovered until later in the process
- ❖ In some cases it's better to treat "missing" as a separate value
- ❖ Numeric attributes are treated just like they are in decision trees

# Pruning rules

- ❖ Two main strategies:
  - ❑ *Incremental* pruning
  - ❑ *Global* pruning
- ❖ Other difference: pruning criterion
  - ❑ Error on hold-out set (*reduced-error pruning*)
  - ❑ Statistical significance
  - ❑ MDL principle
- ❖ Also: post-pruning vs. pre-pruning

# INDUCT

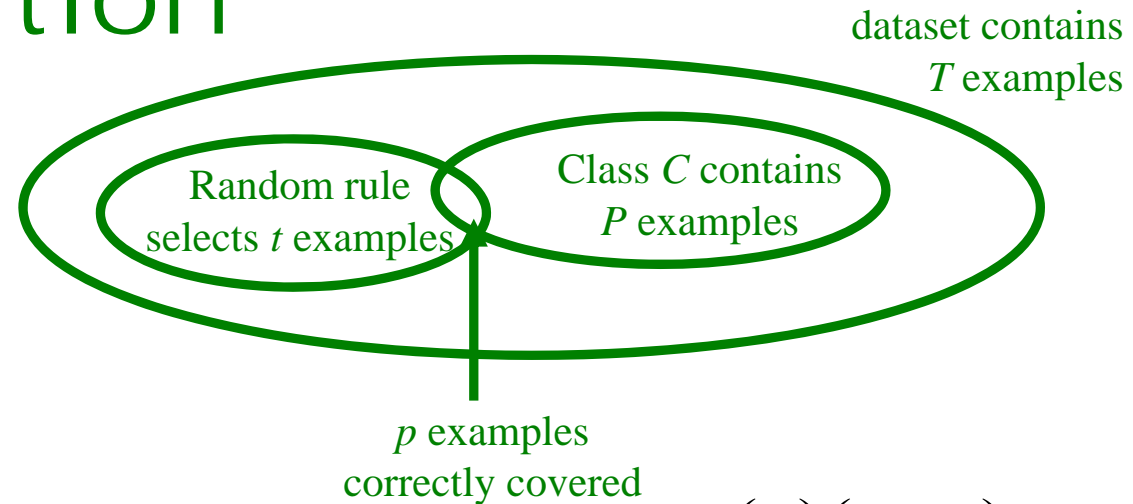
```
Initialize E to the instance set
Until E is empty do
  For each class C for which E contains an instance
    Use basic covering algorithm to create best perfect
      rule for C
    Calculate m(R):  significance for rule
                    and m(R-): significance for rule with final
                        condition omitted
    If m(R-) < m(R), prune rule and repeat previous step
  From the rules for the different classes, select the most
    significant one (i.e. with smallest m(R))
  Print the rule
  Remove the instances covered by rule from E
Continue
```

❖ Performs incremental pruning

# Computing significance

- ❖ INDUCT's significance measure for a rule:
  - ❑ Probability that a completely random rule with same coverage performs at least as well
- ❖ Random rule  $R$  selects  $t$  cases at random from the dataset
- ❖ How likely it is that  $p$  of these belong to the correct class?
- ❖ This probability is given by the *hypergeometric* distribution

# Hypergeometric distribution



Pr[ of  $t$  examples selected at random,  
exactly  $p$  are in class  $C$  ]

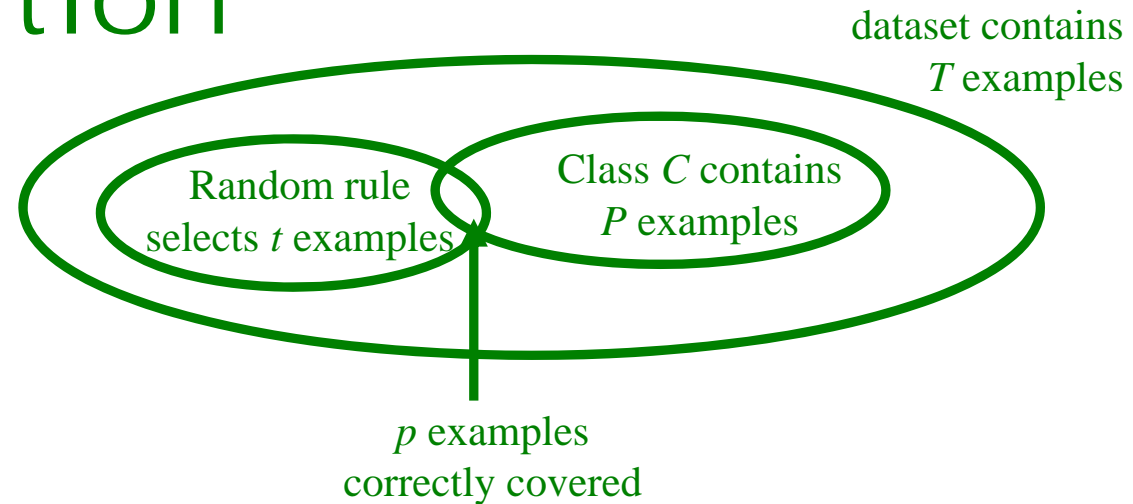
$$\frac{\binom{P}{p} \binom{T-P}{t-p}}{\binom{T}{t}}$$

What is the probability that a  
random rule does *at least as well*?

$$m(R) = \sum_{i=p}^{\min(t,P)} \frac{\binom{P}{i} \binom{T-P}{t-i}}{\binom{T}{t}}$$

*This is the statistical significance of the rule*

# Binomial distribution



- ❖ Hypergeometric is hard to compute
- ❖ Approximation: sample *with replacement* instead of *without replacement*

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

# Using a pruning set

- ❖ For statistical validity, must evaluate measure on data not used for training:
  - ❑ This requires a *growing set* and a *pruning set*
- ❖ *Reduced-error pruning* :  
build full rule set and then prune it
- ❖ *Incremental reduced-error pruning* :  
simplify each rule as soon as it is built
  - ❑ Can re-split data after rule has been pruned
- ❖ Stratification advantageous

# Incremental reduced-error pruning

```
Initialize E to the instance set
Until E is empty do
    Split E into Grow and Prune in the ratio 2:1
    For each class C for which Grow contains an instance
        Use basic covering algorithm to create best perfect rule
        for C
        Calculate w(R): worth of rule on Prune
            and w(R-): worth of rule with final condition
            omitted
        If w(R-) > w(R), prune rule and repeat previous step
    From the rules for the different classes, select the one
    that's worth most (i.e. with largest w(R))
    Print the rule
    Remove the instances covered by rule from E
Continue
```

# Measures used in incr. reduced-error pruning

- ❖  $[p + (N - n)] / T$ 
  - ❑ ( $N$  is total number of negatives)
  - ❑ Counterintuitive:
    - $p = 2000$  and  $n = 1000$  vs.  $p = 1000$  and  $n = 1$
- ❖ Success rate  $p / t$ 
  - ❑ Problem:  $p = 1$  and  $t = 1$   
vs.  $p = 1000$  and  $t = 1001$
- ❖  $(p - n) / t$ 
  - ❑ Same effect as success rate because it equals  $2p/t - 1$
- ❖ Seems hard to find a simple measure of a rule's worth that corresponds with intuition
- ❖ Use hypergeometric/binomial measure?

# Variations

- ❖ Generating rules for classes in order
  - ❑ Start with the smallest class
  - ❑ Leave the largest class covered by the default rule
- ❖ Stopping criterion
  - ❑ Stop rule production if accuracy becomes too low
- ❖ Rule learner RIPPER:
  - ❑ Uses MDL-based stopping criterion
  - ❑ Employs post-processing step to modify rules guided by MDL criterion

# PART

- ❖ Avoids global optimization step used in C4.5rules and RIPPER
- ❖ Generates an unrestricted decision list using basic separate-and-conquer procedure
- ❖ Builds a *partial* decision tree to obtain a rule
  - ❑ A rule is only pruned if all its implications are known
  - ❑ Prevents *hasty generalization*
- ❖ Uses C4.5's procedures to build a tree

# Building a partial tree

Expand-subset (S):

Choose test T and use it to split set of examples  
into subsets

Sort subsets into increasing order of average  
entropy

while

there is a subset X not yet been expanded

AND all subsets expanded so far are leaves

expand-subset(X)

if

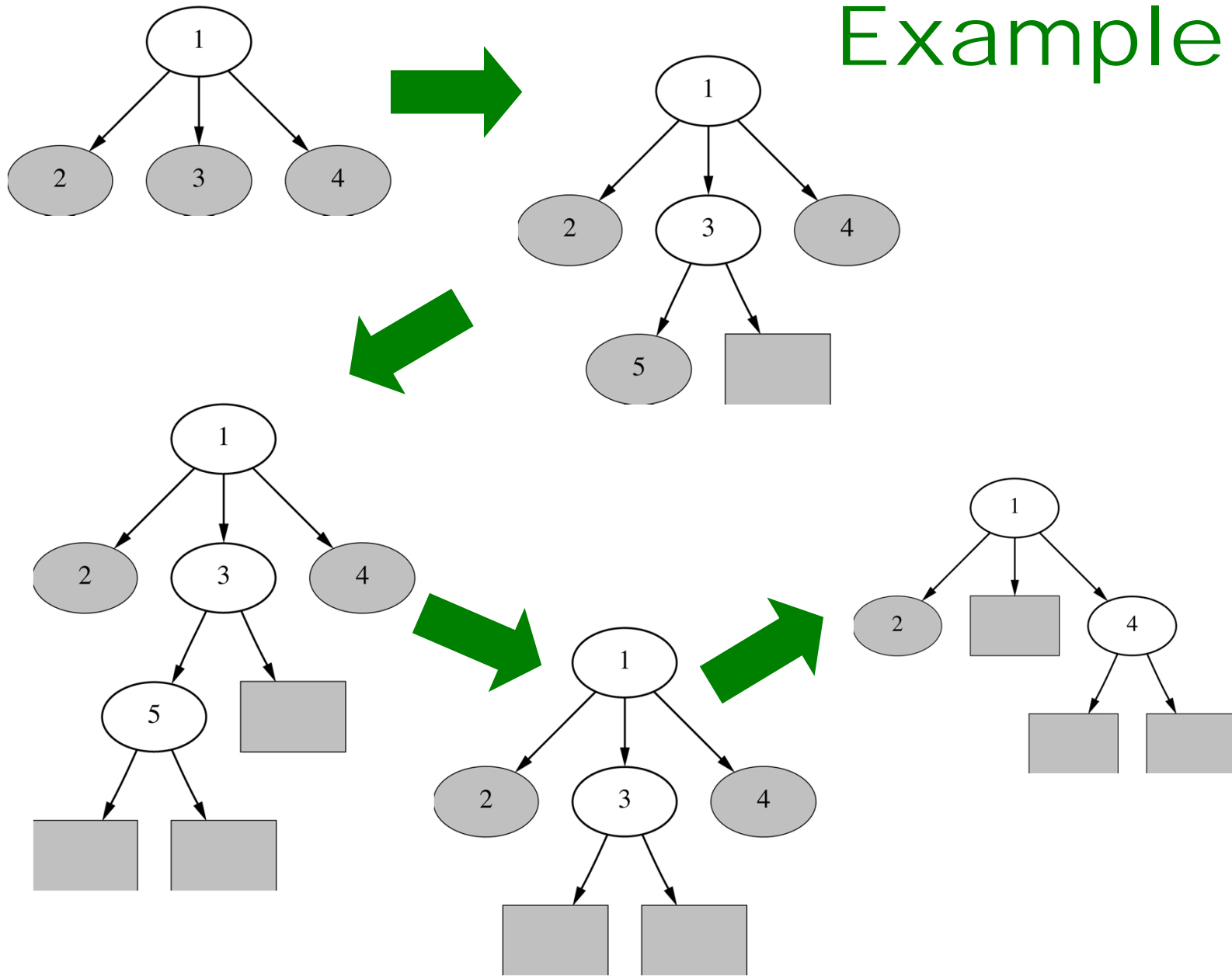
all subsets expanded are leaves

AND estimated error for subtree

$\geq$  estimated error for node

undo expansion into subsets and make node a leaf

# Example



# Notes on PART

- ❖ Make leaf with maximum coverage into a rule
- ❖ Treat missing values just as C4.5 does
  - ❑ I.e. split instance into pieces
- ❖ Time taken to generate a rule:
  - ❑ Worst case: same as for building a pruned tree
    - Occurs when data is noisy
  - ❑ Best case: same as for building a single rule
    - Occurs when data is noise free

# Rules with exceptions

❖ Idea: allow rules to have *exceptions*

❖ Example: rule for iris data

```
If petal-length  $\geq$  2.45 and petal-length < 4.45  
then Iris-versicolor
```

❖ New instance:

Sepal length	Sepal width	Petal length	Petal width	Type
5.1	3.5	2.6	0.2	Iris-setosa

❖ Modified rule:

```
If petal-length  $\geq$  2.45 and petal-length < 4.45  
then Iris-versicolor  
EXCEPT if petal-width < 1.0 then Iris-setosa
```

# A more complex example

## ❖ Exceptions to exceptions to exceptions ...

```
default: Iris-setosa
except if petal-length  $\geq$  2.45 and petal-length  $<$  5.355
      and petal-width  $<$  1.75
  then Iris-versicolor
      except if petal-length  $\geq$  4.95
            and petal-width  $<$  1.55
          then Iris-virginica
          else if sepal-length  $<$  4.95
                and sepal-width  $\geq$  2.45
              then Iris-virginica
      else if petal-length  $\geq$  3.35
            then Iris-virginica
            except if petal-length  $<$  4.85
                  and sepal-length  $<$  5.95
                then Iris-versicolor
```

# Advantages of using exceptions

- ❖ Rules can be updated incrementally
  - ❑ Easy to incorporate new data
  - ❑ Easy to incorporate domain knowledge
- ❖ People often think in terms of exceptions
- ❖ Each conclusion can be considered just in the context of rules and exceptions that lead to it
  - ❑ Locality property is important for understanding large rule sets
  - ❑ “Normal” rule sets don’t offer this advantage

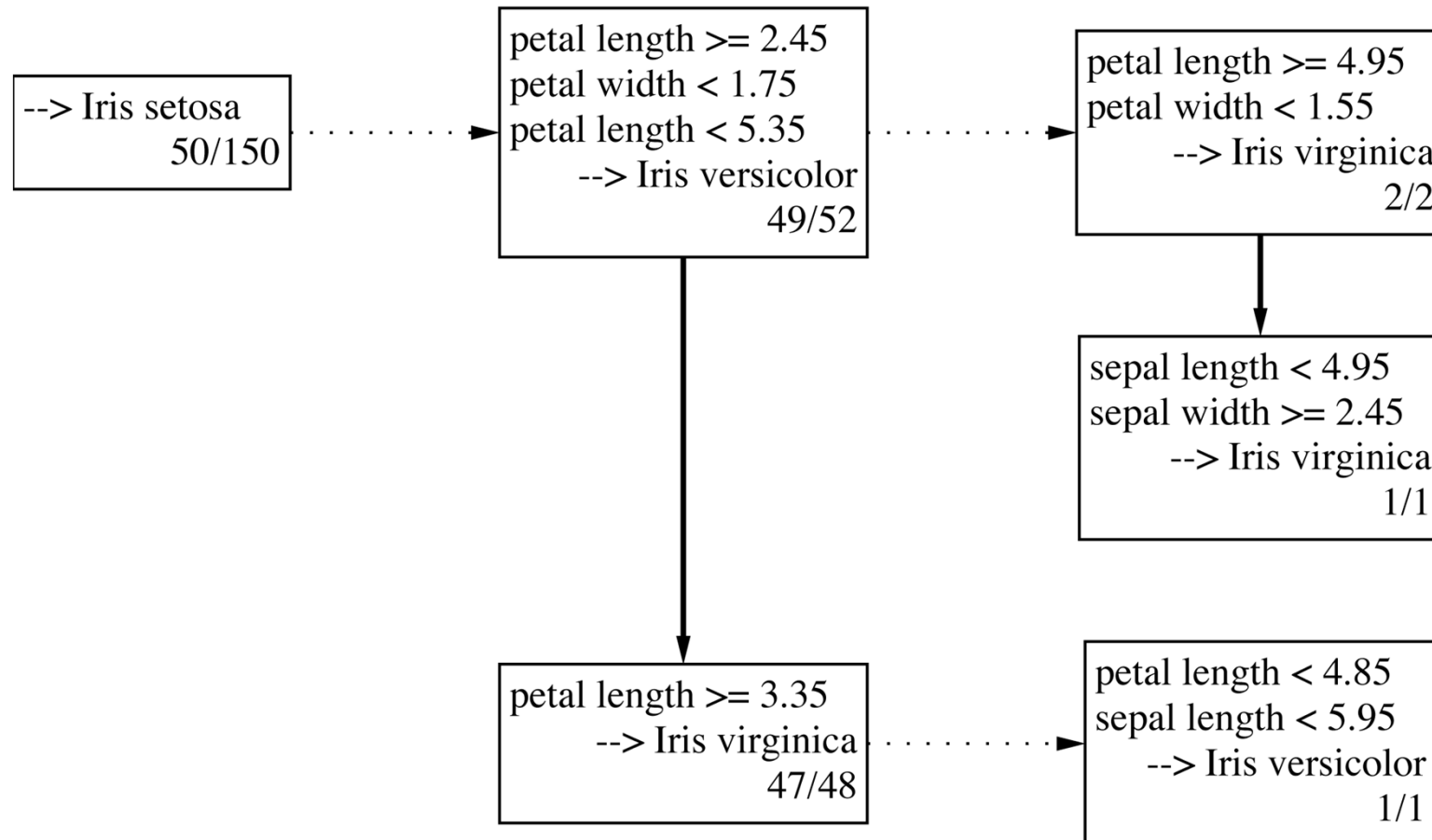
# More on exceptions

- ❖ **Default...except if...then...**  
is logically equivalent to  
**if...then...else**  
(where the else specifies what the default did)
- ❖ But: exceptions offer a psychological advantage
  - ❑ Assumption: defaults and tests early on apply more widely than exceptions further down
  - ❑ Exceptions reflect special cases

# Rules with exceptions

- ❖ Given: a way of generating a single good rule
- ❖ Then: it's easy to generate rules with exceptions
  1. Select default class for top-level rule
  2. Generate a good rule for one of the remaining classes
  3. Apply this method recursively to the two subsets produced by the rule  
(I.e. instances that are covered/not covered)

# Iris data example





# Extending linear classification

- ❖ Linear classifiers can't model nonlinear class boundaries
- ❖ Simple trick:
  - ❑ Map attributes into new space consisting of combinations of attribute values
  - ❑ E.g.: all products of  $n$  factors that can be constructed from the attributes
- ❖ Example with two attributes and  $n = 3$ :

$$x = w_1 a_1^3 + w_2 a_1^2 a_2 + w_3 a_1 a_2^2 + w_4 a_2^3$$

# Problems with this approach

## ❖ 1<sup>st</sup> problem: speed

- ❑ 10 attributes, and  $n = 5 \Rightarrow >2000$  coefficients
- ❑ Use linear regression with attribute selection
- ❑ Run time is cubic in number of attributes

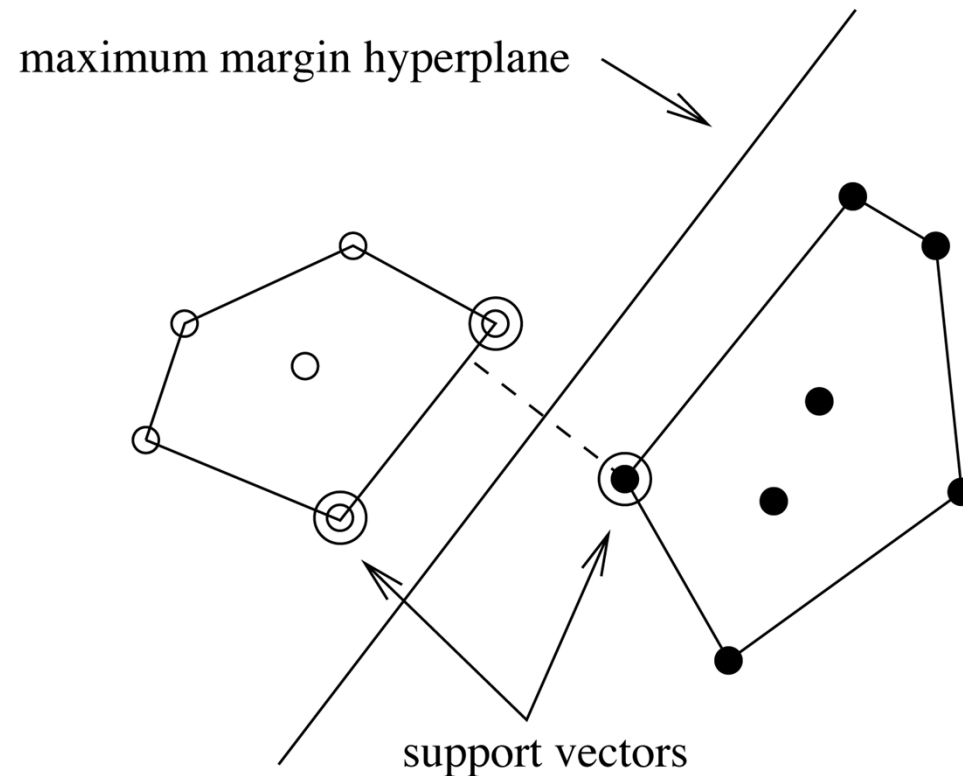
## ❖ 2<sup>nd</sup> problem: overfitting

- ❑ Number of coefficients is large relative to the number of training instances
- ❑ *Curse of dimensionality* kicks in

# Support vector machines

- ❖ *Support vector machines* are algorithms for learning linear classifiers
- ❖ Resilient to overfitting because they learn a particular linear decision boundary:
  - ❑ The *maximum margin hyperplane*
- ❖ Fast in the nonlinear case
  - ❑ Use a mathematical trick to avoid creating “pseudo-attributes”
  - ❑ The nonlinear space is created implicitly

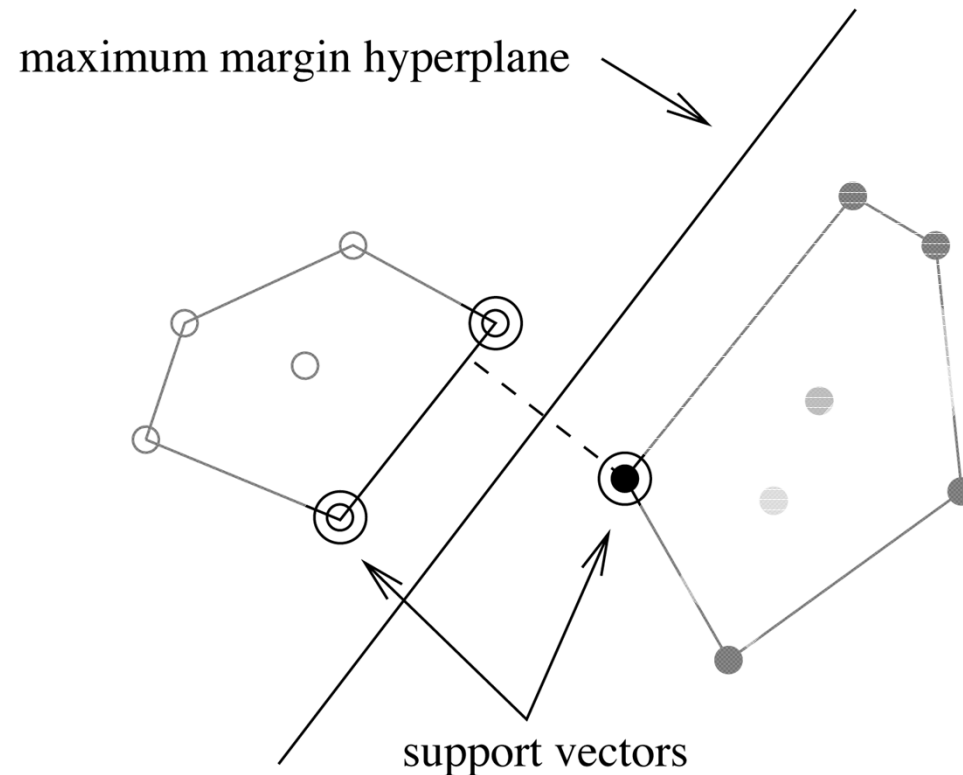
# The maximum margin hyperplane



- ❖ The instances closest to the maximum margin hyperplane are called *support vectors*

# Support vectors

- ❖ The support vectors define the maximum margin hyperplane!
  - All other instances can be deleted without changing its position and orientation



- ❖ This means the hyperplane can be written as
$$x = w_0 + w_1 a_1 + w_2 a_2$$
$$x = b + \sum_{i \text{ is supp. vector}} \alpha_i y_i \mathbf{a}(i) \bullet \mathbf{a}$$

# Finding support vectors

$$x = b + \sum_{i \text{ is supp. vector}} \alpha_i y_i \mathbf{a}(i) \bullet \mathbf{a}$$

- ❖ Support vector: training instance for which  $\alpha_i > 0$
- ❖ Determine  $\alpha_i$  and  $b$ ?—  
*A constrained quadratic optimization problem*
  - ❑ Off-the-shelf tools for solving these problems
  - ❑ However, special-purpose algorithms are faster
  - ❑ Example: Platt's *sequential minimal optimization* algorithm (implemented in WEKA)
- ❖ Note: all this assumes separable data!

# Nonlinear SVMs

- ❖ “Pseudo attributes” represent attribute combinations
- ❖ Overfitting not a problem because the maximum margin hyperplane is stable
  - ❑ There are usually few support vectors relative to the size of the training set
- ❖ Computation time still an issue
  - ❑ Each time the dot product is computed, all the “pseudo attributes” must be included

# A mathematical trick

- ❖ Avoid computing the “pseudo attributes”!
- ❖ Compute the dot product before doing the nonlinear mapping

- ❖ Example: for 
$$x = b + \sum_{i \text{ is supp. vector}} \alpha_i y_i \mathbf{a}(i) \bullet \mathbf{a}$$

compute

$$x = b + \sum_{i \text{ is supp. vector}} \alpha_i y_i (\mathbf{a}(i) \bullet \mathbf{a})^n$$

- ❖ Corresponds to a map into the instance space spanned by all products of  $n$  attributes

# Other kernel functions

❖ Mapping is called a “kernel function”

❖ Polynomial kernel  $x = b + \sum_{i \text{ is supp. vector}} \alpha_i y_i (\mathbf{a}(i) \bullet \mathbf{a})^n$

❖ We can use others:  $x = b + \sum_{i \text{ is supp. vector}} \alpha_i y_i K(\mathbf{a}(i) \bullet \mathbf{a})$

❖ Only requirement:  $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i) \bullet \phi(\mathbf{x}_j)$

❖ Examples:

$$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \bullet \mathbf{x}_j + 1)^d$$

$$K(\mathbf{x}_i, \mathbf{x}_j) = e^{\frac{-(\mathbf{x}_i - \mathbf{x}_j)^2}{2\sigma^2}}$$

$$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\beta \mathbf{x}_i \bullet \mathbf{x}_j + b)$$

# Noise

- ❖ Have assumed that the data is separable (in original or transformed space)
- ❖ Can apply SVMs to noisy data by introducing a “noise” parameter  $C$
- ❖  $C$  bounds the influence of any one training instance on the decision boundary
  - Corresponding constraint:  $0 \leq \alpha_j \leq C$
- ❖ Still a quadratic optimization problem
- ❖ Have to determine  $C$  by experimentation

# Sparse data

- ❖ SVM algorithms speed up dramatically if the data is *sparse* (i.e. many values are 0)
- ❖ Why? Because they compute lots and lots of dot products
- ❖ Sparse data  $\Rightarrow$  compute dot products very efficiently
  - Iterate only over non-zero values
- ❖ SVMs can process sparse datasets with 10,000s of attributes

# Applications

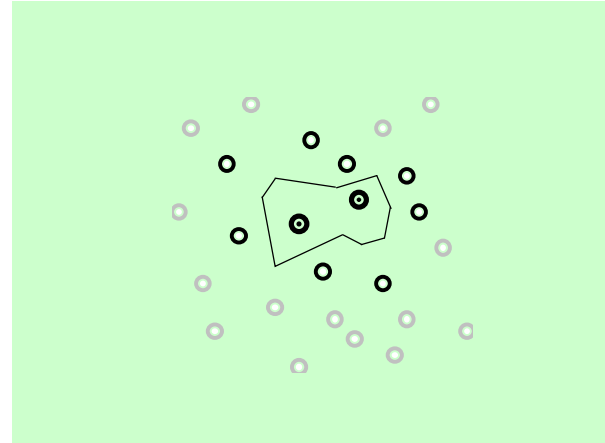
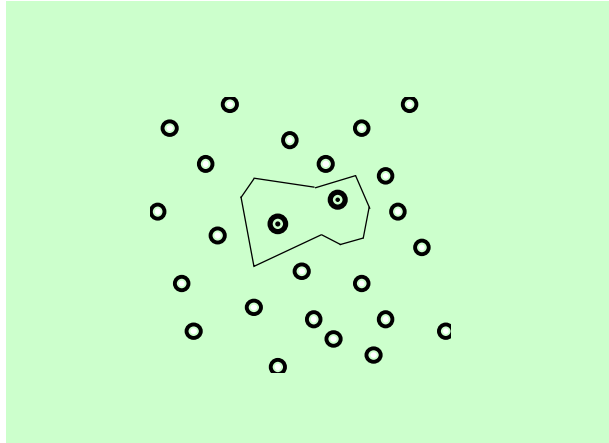
- ❖ Machine vision: e.g face identification
  - ❑ Outperforms alternative approaches (1.5% error)
- ❖ Handwritten digit recognition: USPS data
  - ❑ Comparable to best alternative (0.8% error)
- ❖ Bioinformatics: e.g. prediction of protein secondary structure
- ❖ Text classification
- ❖ Can modify SVM technique for numeric prediction problems



# Instance-based learning

- ❖ Practical problems of 1-NN scheme:
  - ❑ Slow (but: fast tree-based approaches exist)
    - Remedy: remove irrelevant data
  - ❑ Noise (but:  $k$ -NN copes quite well with noise)
    - Remedy: remove noisy instances
  - ❑ All attributes deemed equally important
    - Remedy: weight attributes (or simply select)
  - ❑ Doesn't perform explicit generalization
    - Remedy: rule-based NN approach

# Learning prototypes



- ❖ Only those instances involved in a decision need to be stored
- ❖ Noisy instances should be filtered out
- ❖ Idea: only use *prototypical* examples

# Speed up, combat noise

## ❖ IB2: save memory, speed up classification

- ❑ Work incrementally
- ❑ Only incorporate misclassified instances
- ❑ Problem: noisy data gets incorporated

## ❖ IB3: deal with noise

- ❑ Discard instances that don't perform well
- ❑ Compute confidence intervals for
  - 1. Each instance's success rate
  - 2. Default accuracy of its class
- ❑ Accept/reject instances
  - Accept if lower limit of 1 exceeds upper limit of 2
  - Reject if upper limit of 1 is below lower limit of 2

# Weight attributes

- ❖ IB5: weight each attribute

- (Weights can be class-specific)

- ❖ Weighted Euclidean distance:

$$\sqrt{w_1^2(x_1 - y_1)^2 + \dots + w_n^2(x_n - y_n)^2}$$

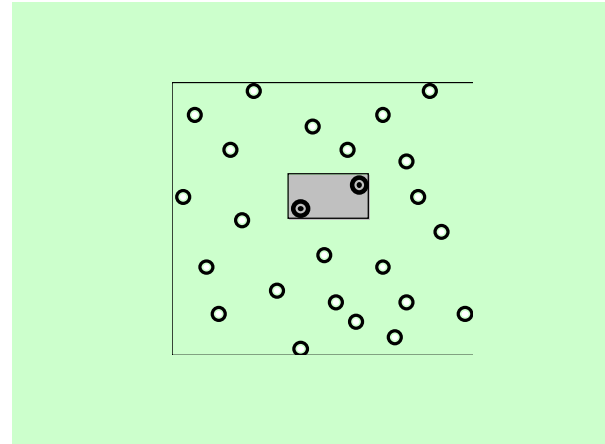
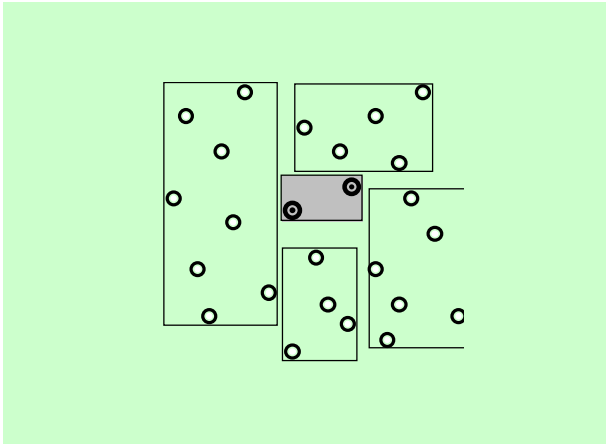
- ❖ Update weights based on nearest neighbor

- Class correct: increase weight

- Class incorrect: decrease weight

- Amount of change for  $i$ th attribute depends on  $|x_i - y_i|$

# Rectangular generalizations

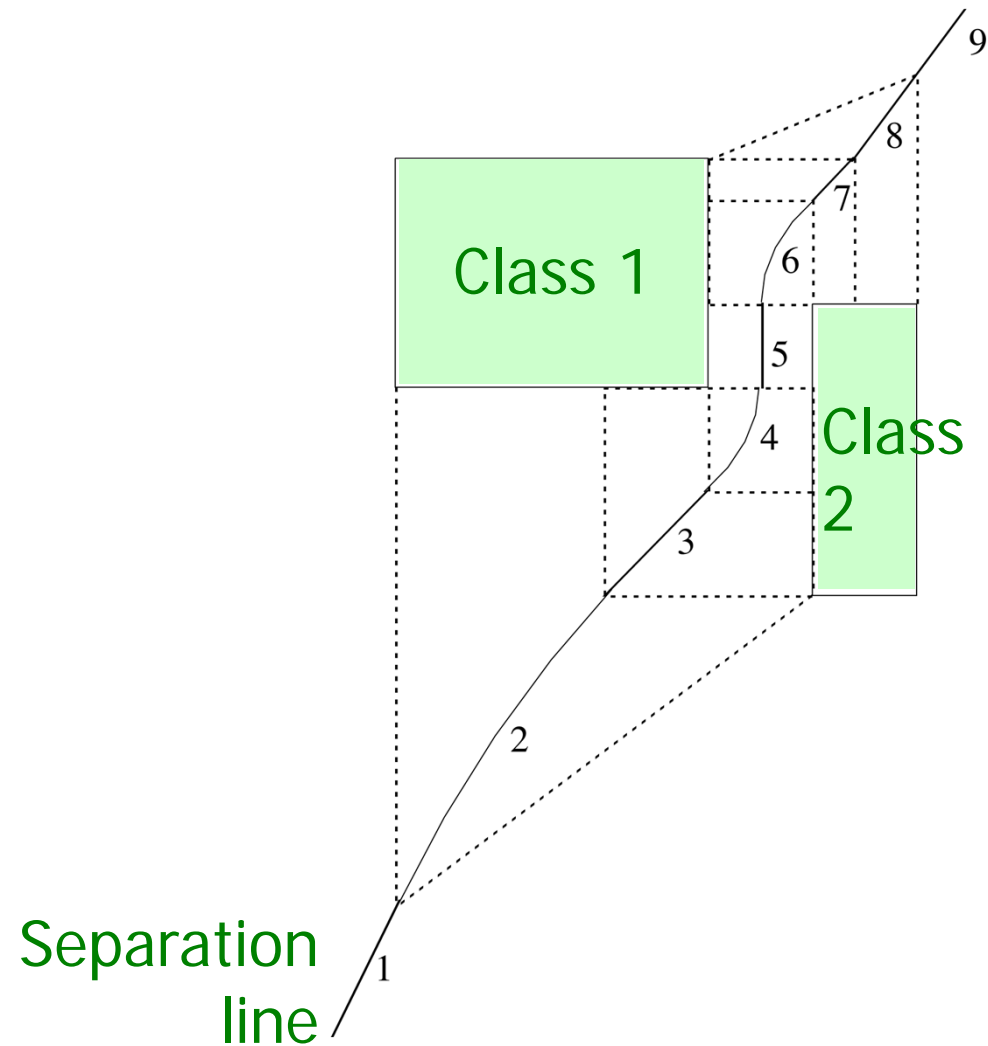


- ❖ Nearest-neighbor rule is used outside rectangles
- ❖ Rectangles are rules! (But they can be more conservative than “normal” rules.)
- ❖ Nested rectangles are rules with exceptions

# Generalized exemplars

- ❖ Generalize instances into *hyperrectangles*
  - ❑ Online: incrementally modify rectangles
  - ❑ Offline version: seek small set of rectangles that cover the instances
- ❖ Important design decisions:
  - ❑ Allow overlapping rectangles?
    - Requires conflict resolution
  - ❑ Allow nested rectangles?
  - ❑ Dealing with uncovered instances?

# Separating generalized exemplars



# Generalized distance functions

- ❖ Given: some transformation operations on attributes
- ❖  $K^*$ : distance = probability of transforming instance A into B by chance
  - ❑ Average over all transformation paths
  - ❑ Weight paths according their probability  
*(need way of measuring this)*
- ❖ Uniform way of dealing with different attribute types
- ❖ Easily generalized to give distance between *sets* of instances



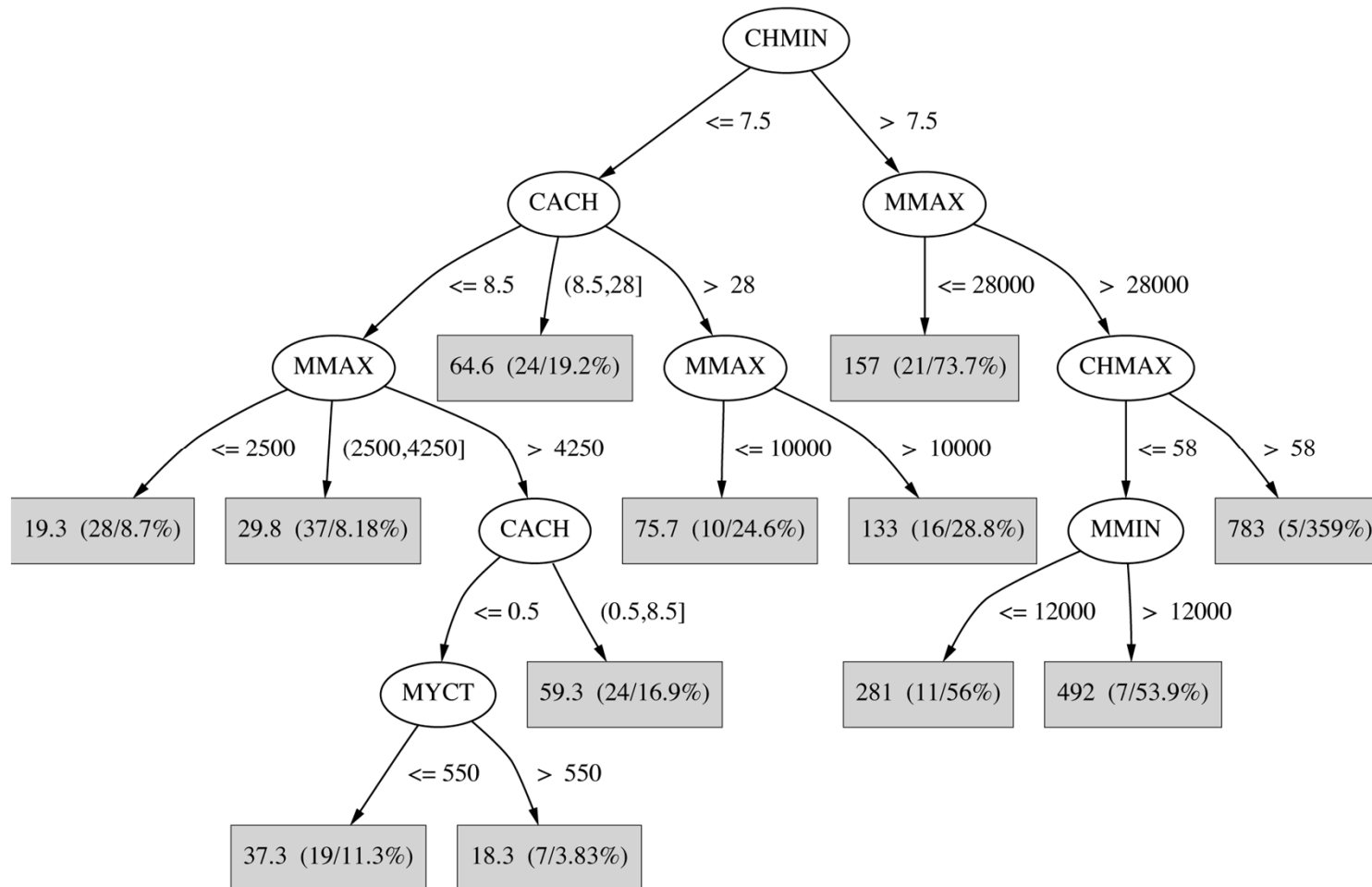
# Trees for numeric prediction

- ❖ *Regression*: the process of computing an expression that predicts a numeric quantity
- ❖ *Regression tree*: “decision tree” where each leaf predicts a numeric quantity
  - ❑ Predicted value is average value of training instances that reach the leaf
- ❖ *Model tree*: “regression tree” with linear regression models at the leaf nodes
  - ❑ Linear patches approximate continuous function

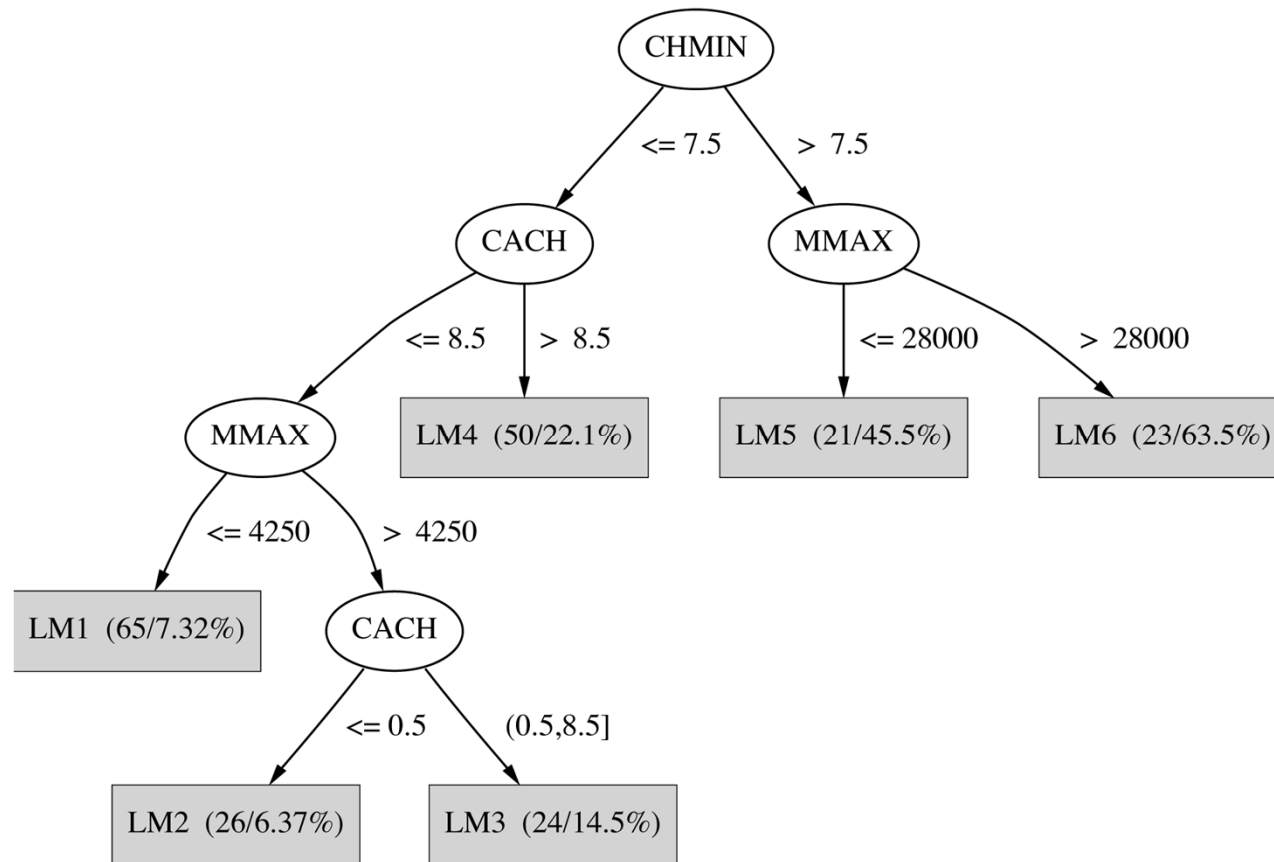
# Linear regression for the CPU data

```
PRP =  
      - 56.1  
      + 0.049 MYCT  
      + 0.015 MMIN  
      + 0.006 MMAX  
      + 0.630 CACH  
      - 0.270 CHMIN  
      + 1.460 CHMAX
```

# Regression tree for the CPU data



# Model tree for the CPU data



# Numeric prediction

- ❖ Counterparts exist for all schemes previously discussed
  - ❑ Decision trees, rule learners, SVMs, etc.
- ❖ All classification schemes can be applied to regression problems using discretization
  - ❑ Discretize the class into intervals
  - ❑ Predict weighted average of interval midpoints
  - ❑ Weight according to class probabilities

# Regression trees

- ❖ Like decision trees, but:
  - ❑ Splitting criterion: minimize intra-subset variation
  - ❑ Termination criterion: std dev becomes small
  - ❑ Pruning criterion: based on numeric error measure
  - ❑ Prediction: Leaf predicts average class values of instances
- ❖ Piecewise constant functions
- ❖ Easy to interpret
- ❖ More sophisticated version: *model trees*

# Model trees

- ❖ Build a regression tree
- ❖ Each leaf  $\Rightarrow$  linear regression function
- ❖ Smoothing: factor in ancestor's predictions
  - ❑ Smoothing formula:  $p' = \frac{np + kq}{n + k}$
  - ❑ Same effect can be achieved by incorporating ancestor models into the leaves
- ❖ Need linear regression function at each *node*
- ❖ At each node, use only a subset of attributes
  - ❑ Those occurring in subtree
  - ❑ (+maybe those occurring in path to the root)
- ❖ Fast: tree usually uses only a small subset of the attributes

# Building the tree

## ❖ Splitting: standard deviation reduction

$$SDR = sd(T) - \sum_i \frac{|T_i|}{|T|} \times sd(T_i)$$

## ❖ Termination:

- ❑ Standard deviation < 5% of its value on full training set
- ❑ Too few instances remain (e.g. < 4)

## ❖ Pruning:

- ❑ Heuristic estimate of absolute error of LR models:

$$\frac{n + v}{n - v} \times \text{average\_absolute\_error}$$

- ❑ Greedily remove terms from LR models to minimize estimated error
- ❑ Heavy pruning: single model may replace whole subtree
- ❑ Proceed bottom up: compare error of LR model at internal node to error of subtree

# Nominal attributes

- ❖ Convert nominal attributes to binary ones
  - ❑ Sort attribute by average class value
  - ❑ If attribute has  $k$  values, generate  $k - 1$  binary attributes
    - $i$ th is 0 if value lies within the first  $i$ , otherwise 1
- ❖ Treat binary attributes as numeric
- ❖ Can prove: best split on one of the new attributes is the best (binary) split on original

# Missing values

- ❖ Modify splitting criterion:

$$SDR = \frac{m}{|T|} \times \left[ sd(T) - \sum_i \frac{|T_i|}{|T|} \times sd(T_i) \right]$$

- ❖ To determine which subset an instance goes into, use *surrogate splitting*

- ☐ Split on the attribute whose correlation with original is greatest
- ☐ Problem: complex and time-consuming
- ☐ Simple solution: always use the class

- ❖ Test set: replace missing value with average

# Surrogate splitting based on class

- ❖ Choose split point based on instances with known values
- ❖ Split point divides instances into 2 subsets
  - $L$  (smaller class average)
  - $R$  (larger)
- ❖  $m$  is the average of the two averages
- ❖ For an instance with a missing value:
  - Choose  $L$  if class value  $< m$
  - Otherwise  $R$
- ❖ Once full tree is built, replace missing values with averages of corresponding leaf nodes

# Pseudo-code for M5'

- ❖ Four methods:
  - ❑ Main method: *MakeModelTree*
  - ❑ Method for splitting: *split*
  - ❑ Method for pruning: *prune*
  - ❑ Method that computes error: *subtreeError*
- ❖ We'll briefly look at each method in turn
- ❖ Assume that linear regression method performs attribute subset selection based on error

# *MakeModelTree*

```
MakeModelTree (instances)
{
    SD = sd(instances)
    for each k-valued nominal attribute
        convert into k-1 synthetic binary attributes
    root = newNode
    root.instances = instances
    split(root)
    prune(root)
    printTree(root)
}
```

# *split*

```
split(node)
{
    if sizeof(node.instances) < 4 or
        sd(node.instances) < 0.05*SD
        node.type = LEAF
    else
        node.type = INTERIOR
        for each attribute
            for all possible split positions of attribute
                calculate the attribute's SDR
        node.attribute = attribute with maximum SDR
        split(node.left)
        split(node.right)
}
```

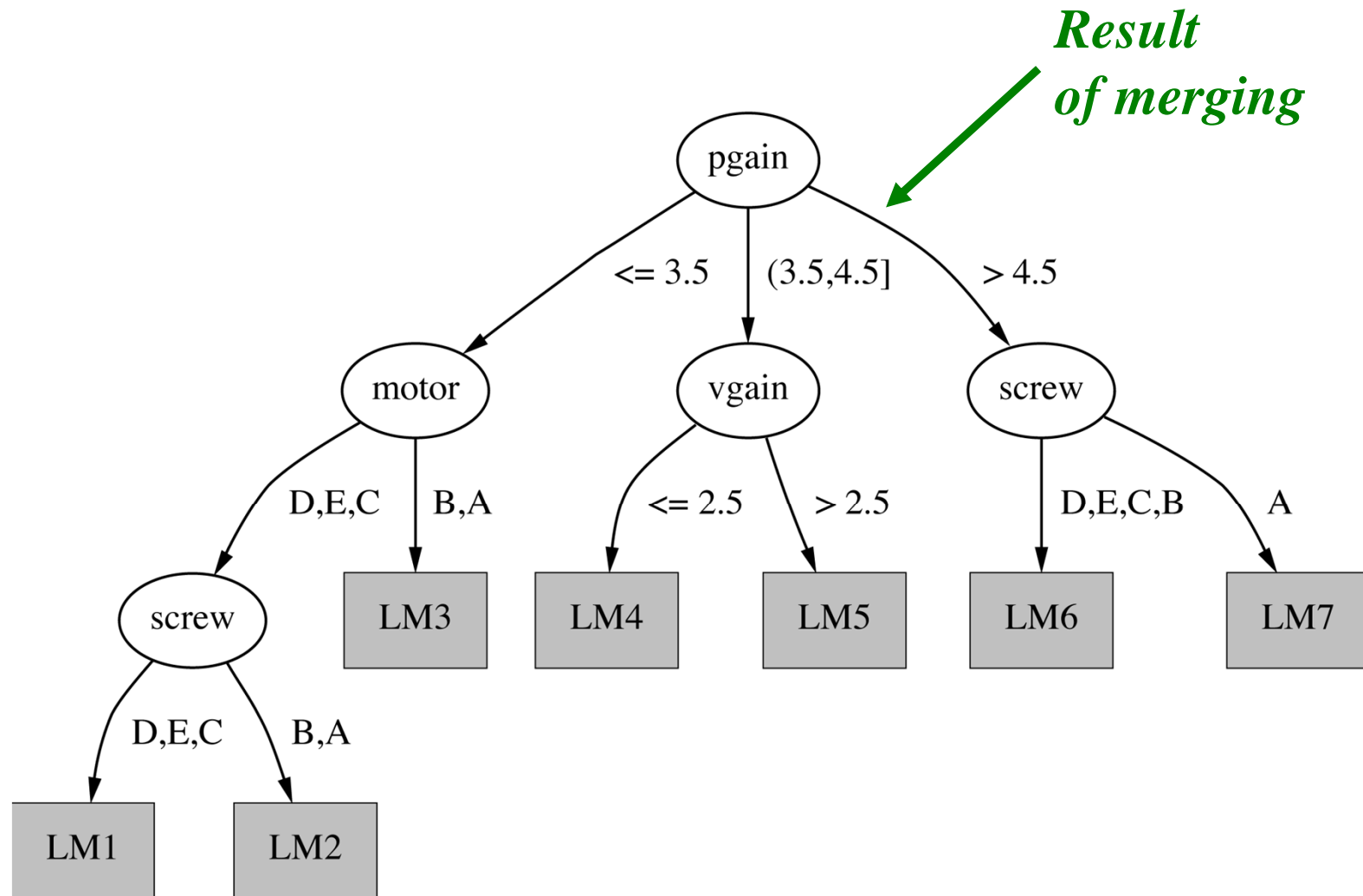
# *prune*

```
prune(node)
{
    if node = INTERIOR then
        prune(node.leftChild)
        prune(node.rightChild)
        node.model = linearRegression(node)
        if subtreeError(node) > error(node) then
            node.type = LEAF
}
```

# *subtreeError*

```
subtreeError(node)
{
    l = node.left; r = node.right
    if node = INTERIOR then
        return (sizeof(l.instances)*subtreeError(l)
                + sizeof(r.instances)*subtreeError(r))
                /sizeof(node.instances)
    else return error(node)
}
```

# Model tree for servo data



# Locally weighted regression

- ❖ Numeric prediction that combines
  - ❑ instance-based learning
  - ❑ linear regression
- ❖ “Lazy”:
  - ❑ computes regression function at prediction time
  - ❑ works incrementally
- ❖ Weight training instances
  - ❑ according to distance to test instance
  - ❑ needs weighted version of linear regression
- ❖ Advantage: nonlinear approximation
- ❖ But: slow

# Design decisions

- ❖ Weighting function:
  - ❑ Inverse Euclidean distance
  - ❑ Gaussian kernel applied to Euclidean distance
  - ❑ Triangular kernel used the same way
  - ❑ etc.
- ❖ *Smoothing parameter* is used to scale the distance function
  - ❑ Multiply distance by inverse of this parameter
  - ❑ Possible choice: distance of  $k$  th nearest training instance (makes it data dependent)

# Discussion

- ❖ Regression trees were introduced in CART
- ❖ Quinlan proposed model tree method (M5)
- ❖ M5': slightly improved, publicly available
- ❖ Quinlan also investigated combining instance-based learning with M5
- ❖ CUBIST: Quinlan's commercial rule learner for numeric prediction
- ❖ Interesting comparison: Neural nets vs. M5



# Clustering

- ❖ *Unsupervised*: no target value to predict
- ❖ Differences between models/algorithms:
  - ❑ Exclusive vs. overlapping
  - ❑ Deterministic vs. probabilistic
  - ❑ Hierarchical vs. flat
  - ❑ Incremental vs. batch learning
- ❖ Problem:
  - Evaluation?—usually by inspection
- ❖ But:
  - If treated as density estimation problem, clusters can be evaluated on test data!

# Hierarchical clustering

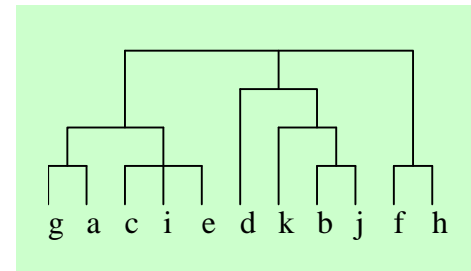
## ❖ Bottom up

- ❑ Start with single-instance clusters
- ❑ At each step, join the two closest clusters
- ❑ Design decision: distance between clusters
  - E.g. two closest instances in clusters  
vs. distance between means

## ❖ Top down

- ❑ Start with one universal cluster
- ❑ Find two clusters
- ❑ Proceed recursively on each subset
- ❑ Can be very fast

## ❖ Both methods produce a *dendrogram*



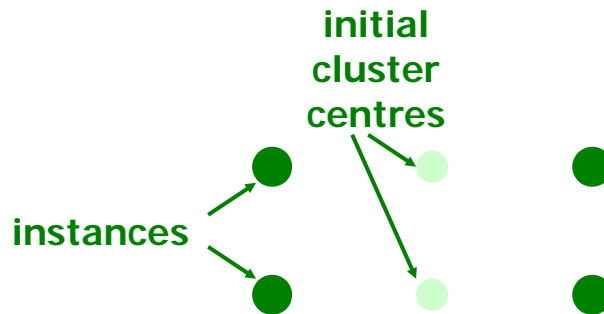
# The $k$ -means algorithm

To cluster data into  $k$  groups:  
( $k$  is predefined)

1. Choose  $k$  cluster centers
  - e.g. at random
2. Assign instances to clusters
  - based on distance to cluster centers
3. Compute *centroids* of clusters
4. Go to step 1
  - until convergence

# Discussion

- ❖ Result can vary significantly
  - ❑ based on initial choice of seeds
- ❖ Can get trapped in local minimum
  - ❑ Example:



- ❖ To increase chance of finding global optimum:  
restart with different random seeds

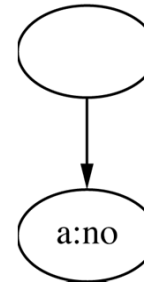
# Incremental clustering

- ❖ Heuristic approach (COBWEB/CLASSIT)
- ❖ Form a hierarchy of clusters incrementally
- ❖ Start:
  - ❑ tree consists of empty root node
- ❖ Then:
  - ❑ add instances one by one
  - ❑ update tree appropriately at each stage
  - ❑ to update, find the right leaf for an instance
  - ❑ May involve restructuring the tree
- ❖ Base update decisions on *category utility*

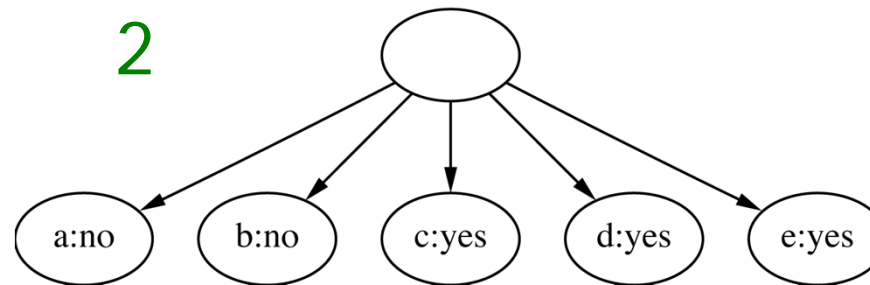
# Clustering weather data

ID	Outlook	Temp.	Humidity	Windy
A	Sunny	Hot	High	False
B	Sunny	Hot	High	True
C	Overcast	Hot	High	False
D	Rainy	Mild	High	False
E	Rainy	Cool	Normal	False
F	Rainy	Cool	Normal	True
G	Overcast	Cool	Normal	True
H	Sunny	Mild	High	False
I	Sunny	Cool	Normal	False
J	Rainy	Mild	Normal	False
K	Sunny	Mild	Normal	True
L	Overcast	Mild	High	True
M	Overcast	Hot	Normal	False
N	Rainy	Mild	High	True

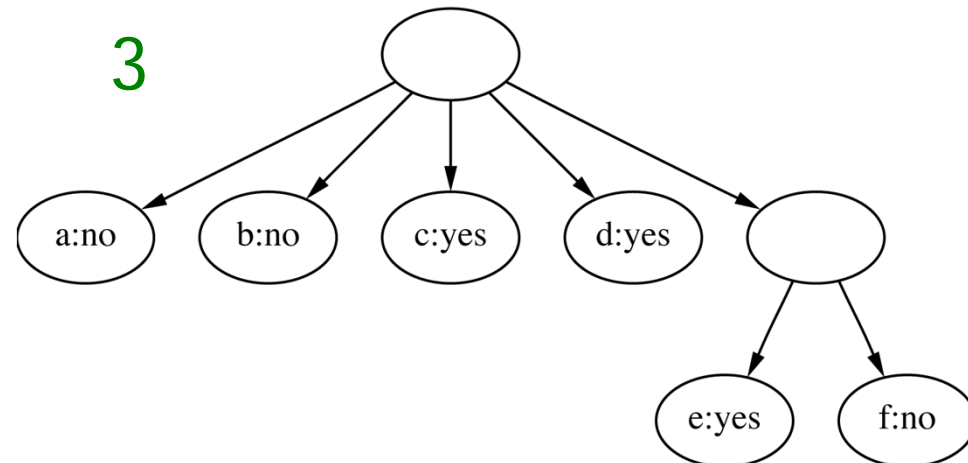
1



2



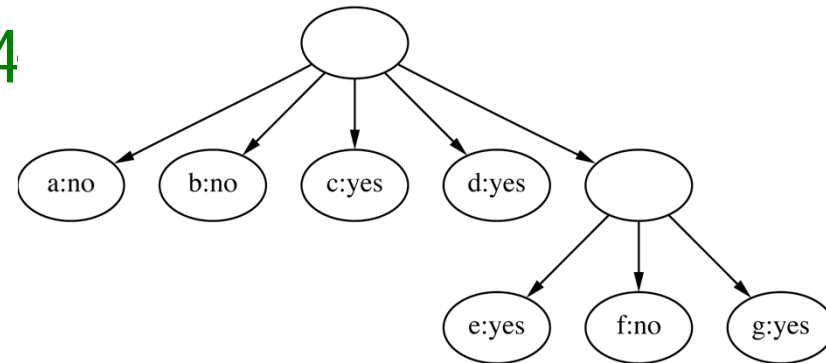
3



# Clustering weather data

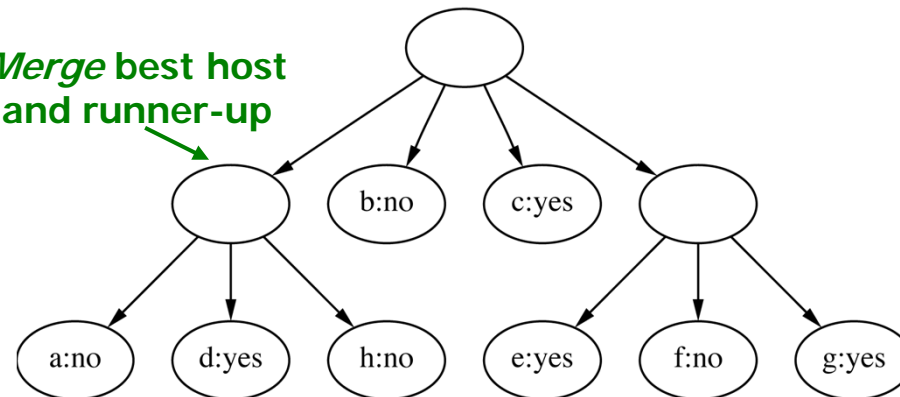
ID	Outlook	Temp.	Humidity	Windy
A	Sunny	Hot	High	False
B	Sunny	Hot	High	True
C	Overcast	Hot	High	False
D	Rainy	Mild	High	False
E	Rainy	Cool	Normal	False
F	Rainy	Cool	Normal	True
G	Overcast	Cool	Normal	True
H	Sunny	Mild	High	False
I	Sunny	Cool	Normal	False
J	Rainy	Mild	Normal	False
K	Sunny	Mild	Normal	True
L	Overcast	Mild	High	True
M	Overcast	Hot	Normal	False
N	Rainy	Mild	High	True

4



5

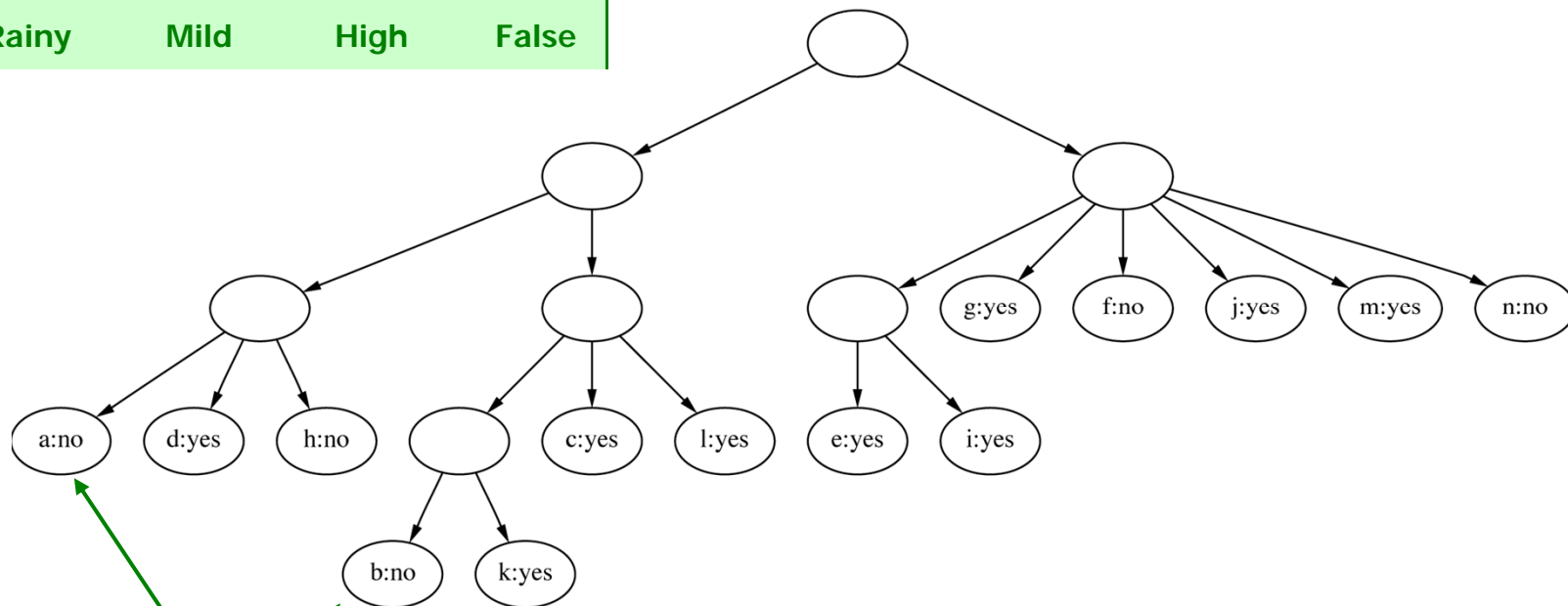
*Merge best host  
and runner-up*



Consider *splitting* the best  
host if merging doesn't help

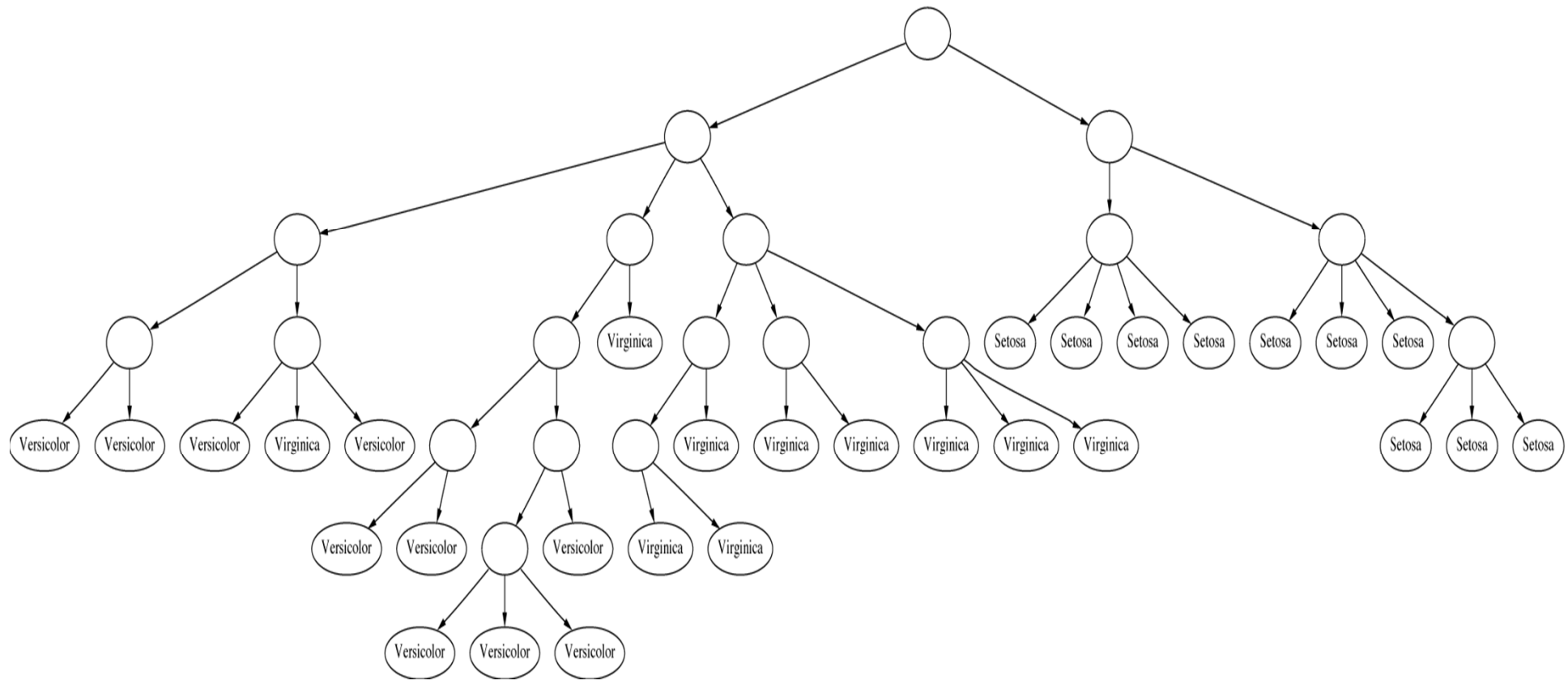
# Final hierarchy

ID	Outlook	Temp.	Humidity	Windy
A	Sunny	Hot	High	False
B	Sunny	Hot	High	True
C	Overcast	Hot	High	False
D	Rainy	Mild	High	False

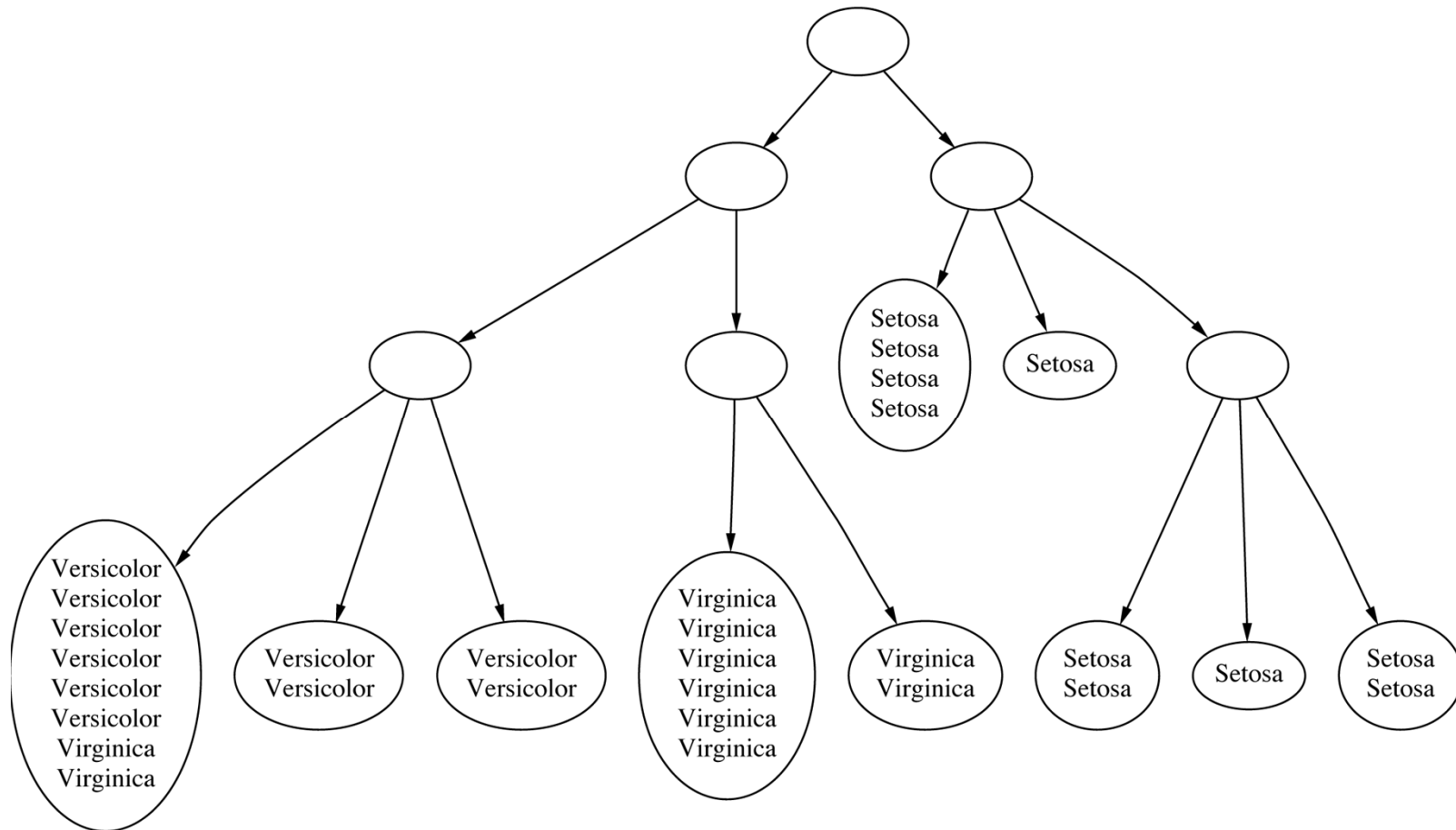


Oops! *a* and *b* are  
actually very similar

# Example: the iris data (subset)




# Clustering with cutoff



# Category utility

- ❖ Category utility: quadratic loss function defined on conditional probabilities:

$$CU(C_1, C_2, \dots, C_k) = \frac{\sum_l \Pr[C_l] \sum_i \sum_j (\Pr[a_i = v_{ij} | C_l]^2 - \Pr[a_i = v_{ij}]^2)}{k}$$


- ❖ Every instance in different category  $\Rightarrow$  numerator becomes

$$\begin{array}{c} m - \Pr[a_i = v_{ij}]^2 \\ \uparrow \\ \text{number of attributes} \end{array} \quad \longleftarrow \text{maximum}$$

# Numeric attributes

❖ Assume normal distribution:  $f(a) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(a-\mu)^2}{2\sigma^2}}$

❖ Then:  $\sum_j \Pr[a_i = v_{ij}]^2 \Leftrightarrow \int f(a_i)^2 da_i = \frac{1}{2\sqrt{\pi}\sigma_i}$

❖ Thus  $CU = \frac{\sum_l \Pr[C_l] \sum_i \sum_j (\Pr[a_i = v_{ij} | C_l]^2 - \Pr[a_i = v_{ij}]^2)}{k}$

becomes  $CU = \frac{\sum_l \Pr[C_l] \frac{1}{2\sqrt{\pi}} \sum_i \left( \frac{1}{\sigma_{il}} - \frac{1}{\sigma_i} \right)}{k}$

❖ Prespecified minimum variance

□ *acuity* parameter

# Probability-based clustering

- ❖ Problems with heuristic approach:
  - ❑ Division by  $k$ ?
  - ❑ Order of examples?
  - ❑ Are restructuring operations sufficient?
  - ❑ Is result at least *local* minimum of category utility?
- ❖ Probabilistic perspective  $\Rightarrow$   
seek the *most likely* clusters given the data
- ❖ Also: instance belongs to a particular cluster  
*with a certain probability*

# Finite mixtures

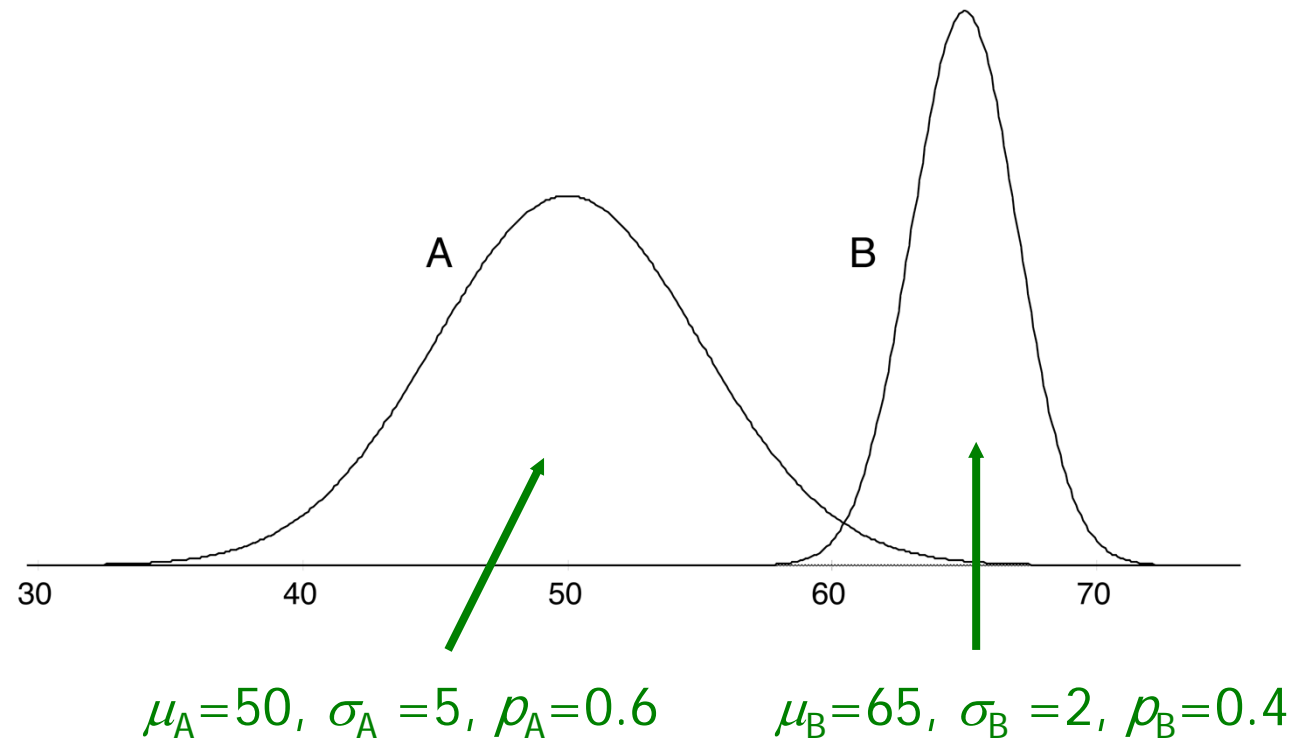
- ❖ Model data using a *mixture* of distributions
- ❖ One cluster, one distribution
  - governs probabilities of attribute values in that cluster
- ❖ *Finite mixtures* : finite number of clusters
- ❖ Individual distributions are normal (usually)
- ❖ Combine distributions using cluster weights

# Two-class mixture model

data

A	51	B	62	B	64	A	48	A	39	A	51
A	43	A	47	A	51	B	64	B	62	A	48
B	62	A	52	A	52	A	51	B	64	B	64
B	64	B	64	B	62	B	63	A	52	A	42
A	45	A	51	A	49	A	43	B	63	A	48
A	42	B	65	A	48	B	65	B	64	A	41
A	46	A	48	B	62	B	66	A	48		
A	45	A	49	A	43	B	65	B	64		
A	45	A	46	A	40	A	46	A	48		

model



# Using the mixture model

- ❖ Probability that instance  $x$  belongs to cluster  $A$ :

$$\Pr[A | x] = \frac{\Pr[x | A] \Pr[A]}{\Pr[x]} = \frac{f(x; \mu_A, \sigma_A) p_A}{\Pr[x]}$$

with

$$f(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

- ❖ *Likelihood* of an instance given the clusters:

$$\Pr[x | \text{the distributions}] = \sum_i \Pr[x | \text{cluster}_i] \Pr[\text{cluster}_i]$$

# Learning the clusters

- ❖ Assume:
  - we know there are  $k$  clusters
- ❖ Learn the clusters  $\Rightarrow$ 
  - determine their parameters
  - I.e. means and standard deviations
- ❖ Performance criterion:
  - *likelihood of training data given the clusters*
- ❖ EM algorithm
  - finds a local maximum of the likelihood

# EM algorithm

- ❖ EM = Expectation-Maximization
  - ❑ Generalize  $k$ -means to probabilistic setting
- ❖ Iterative procedure:
  - ❑ E “expectation” step:  
Calculate cluster probability for each instance
  - ❑ M “maximization” step:  
Estimate distribution parameters from cluster probabilities
- ❖ Store cluster probabilities as instance weights
- ❖ Stop when improvement is negligible

# More on EM

- ❖ Estimate parameters from weighted instances

$$\mu_A = \frac{w_1 x_1 + w_2 x_2 + \dots + w_n x_n}{w_1 + w_2 + \dots + w_n}$$

$$\sigma_A^2 = \frac{w_1 (x_1 - \mu)^2 + w_2 (x_2 - \mu)^2 + \dots + w_n (x_n - \mu)^2}{w_1 + w_2 + \dots + w_n}$$

- ❖ Stop when log-likelihood saturates

- ❖ Log-likelihood:  $\sum_i \log(p_A \Pr[x_i | A] + p_B \Pr[x_i | B])$

# Extending the mixture model

- ❖ More than two distributions: easy
- ❖ Several attributes: easy—assuming independence!
- ❖ Correlated attributes: difficult
  - ❑ Joint model: bivariate normal distribution with a (symmetric) covariance matrix
  - ❑  $n$  attributes: need to estimate  $n + n(n+1)/2$  parameters

# More mixture model extensions

- ❖ Nominal attributes: easy if independent
- ❖ Correlated nominal attributes: difficult
  - ❑ Two correlated attributes  $\Rightarrow \nu_1 \nu_2$  parameters
- ❖ Missing values: easy
- ❖ Can use other distributions than normal:
  - ❑ “log-normal” if predetermined minimum is given
  - ❑ “log-odds” if bounded from above and below
  - ❑ Poisson for attributes that are integer counts
- ❖ Use cross-validation to estimate  $k$  !

# Bayesian clustering

- ❖ Problem: many parameters  $\Rightarrow$  EM overfits
- ❖ *Bayesian approach* : give every parameter a prior probability distribution
  - ❑ Incorporate prior into overall likelihood figure
  - ❑ Penalizes introduction of parameters
- ❖ Eg: Laplace estimator for nominal attributes
- ❖ Can also have prior on number of clusters!
- ❖ Implementation: NASA's AUTOCLASS

# Discussion

- ❖ Can interpret clusters by using supervised learning
  - ❑ post-processing step
- ❖ Decrease dependence between attributes?
  - ❑ pre-processing step
  - ❑ E.g. use *principal component analysis*
- ❖ Can be used to fill in missing values
- ❖ Key advantage of probabilistic clustering:
  - ❑ Can estimate likelihood of data
  - ❑ Use it to compare different models objectively