Machine Learning in Real World: C4.5

Outline

- Handling Numeric Attributes
  - Finding Best Split(s)
- Dealing with Missing Values
- Pruning
  - Pre-pruning, Post-pruning, Error Estimates
- From Trees to Rules

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

C4.5 History

- ID3, CHAID – 1960s
- C4.5 innovations (Quinlan):
  - permit numeric attributes
  - deal sensibly with missing values
  - pruning to deal with for noisy data
- C4.5 - one of best-known and most widely-used learning algorithms
  - Last research version: C4.8, implemented in Weka as J4.8 (Java)
  - Commercial successor: C5.0 (available from Rulequest)

Numeric attributes

- Standard method: binary splits
  - E.g. 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 – nominal values

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
…	…	…	…	…

```
If outlook = sunny and humidity = high 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
```

Weather data - numeric

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 < 85 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 $\geq$ 71.5: yes/5, no/3
  - Info([4,2],[5,3])
    = 6/14 info([4,2]) + 8/14 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: $O(n \log n)$
- ***Q. Does this have to be repeated at each node of the tree?***
- A: 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

More speeding up

- Entropy only needs to be evaluated between points of different classes (Fayyad & Irani, 1992)

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

Potential optimal breakpoints

Breakpoints between values of the same class cannot be optimal

Binary *vs.* multi-way 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

Missing as a separate value

- Missing value denoted “?” in C4.X
- Simple idea: treat missing as a separate value
- Q: When this is not appropriate?
- A: When values are missing due to different reasons
  - Example 1: gene expression could be missing when it is very high or very low
  - Example 2: field **IsPregnant**=missing for a male patient should be treated differently (no) than for a female patient of age 25 (unknown)

Missing values - advanced

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

- Goal: Prevent overfitting to noise in the data
- Two strategies for "pruning" the decision tree:
  - *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 too 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

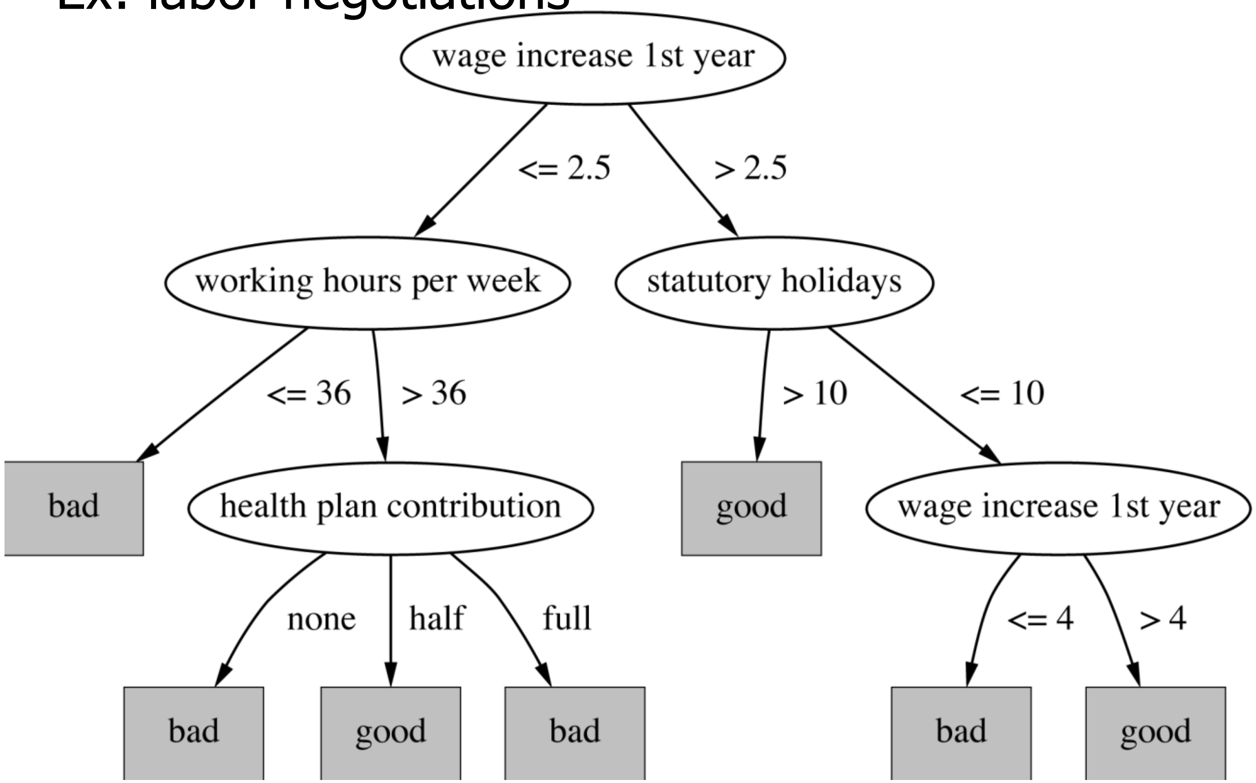
- Pre-pruning may stop the growth process prematurely: *early stopping*
- Classic example: XOR/Parity-problem
  - No *individual* attribute exhibits any significant association to the class
  - Structure is only visible in fully expanded tree
  - Pre-pruning won't expand the root node
- But: XOR-type problems rare in practice
- And: pre-pruning faster than post-pruning

Post-pruning

- 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:
  1. *Subtree replacement*
  2. *Subtree raising*
- Possible strategies:
  - error estimation
  - significance testing
  - MDL principle

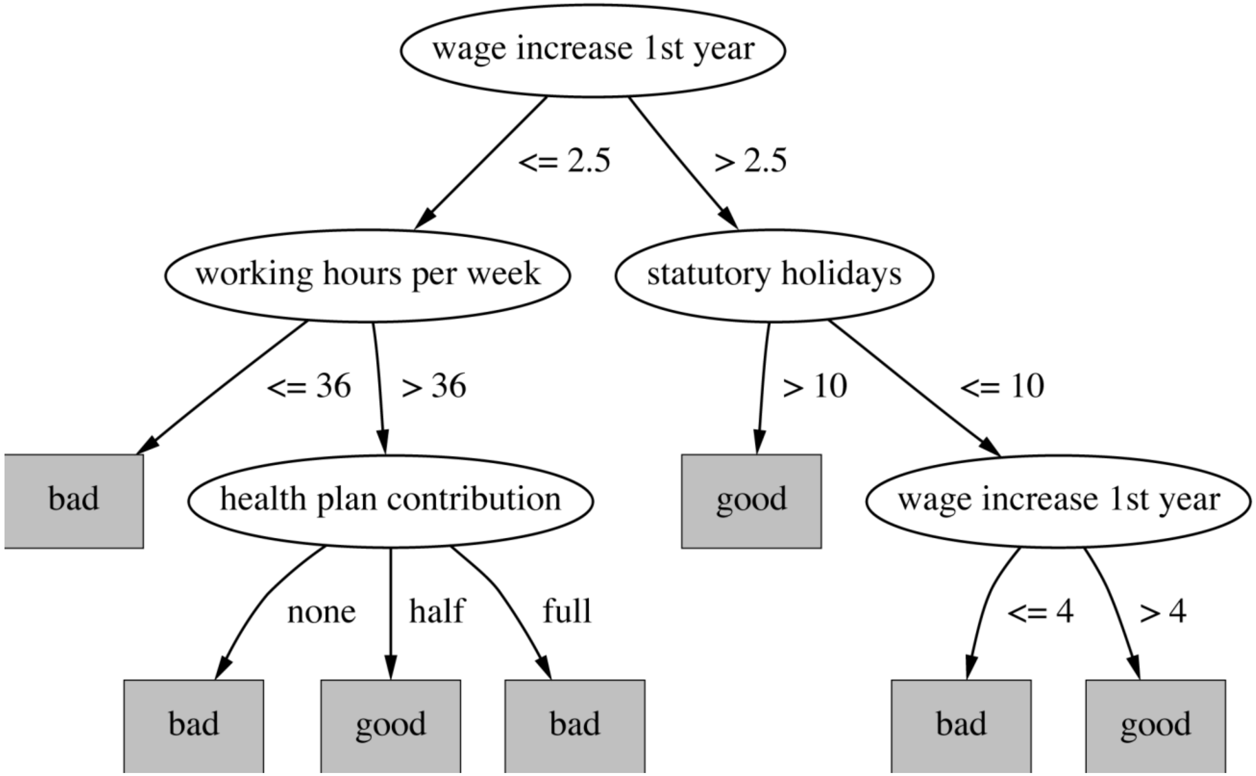
Subtree replacement, 1

- *Bottom-up*
- Consider replacing a tree only after considering all its subtrees
- Ex: labor negotiations



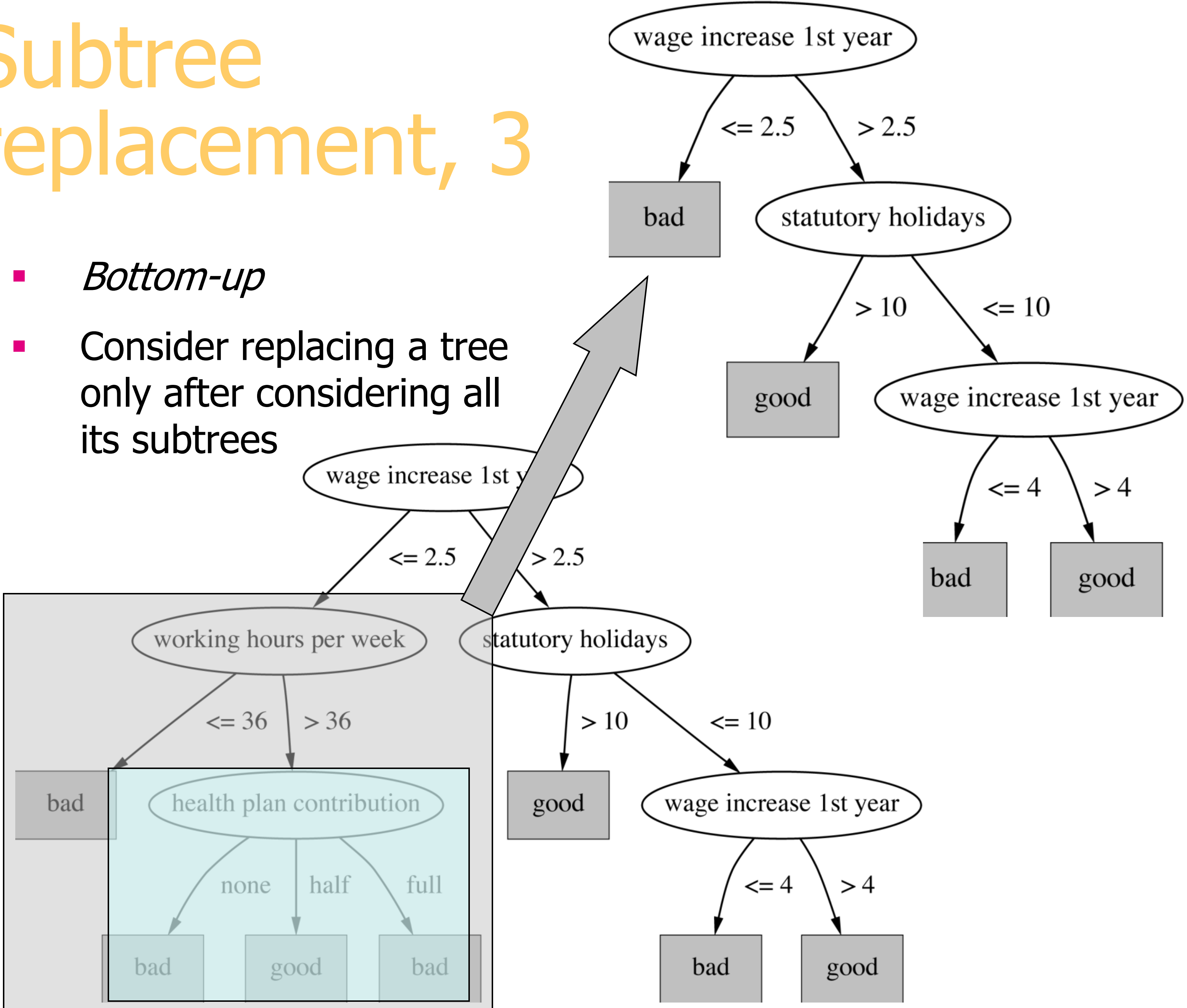
Subtree replacement, 2

What subtree can we replace?

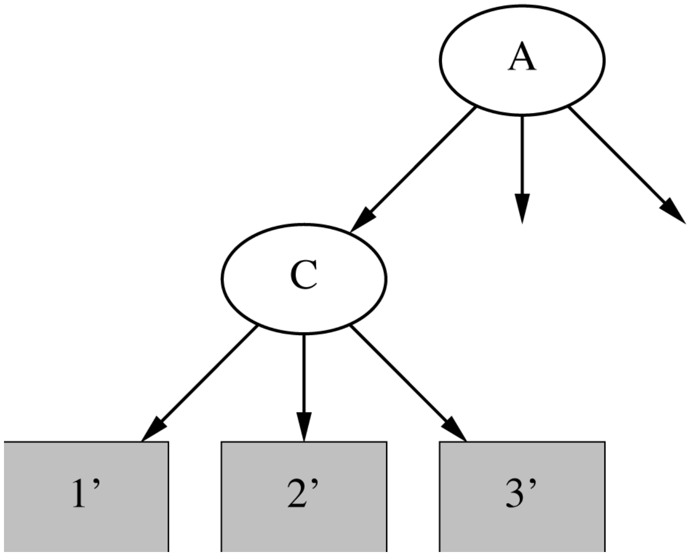


Subtree replacement, 3

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

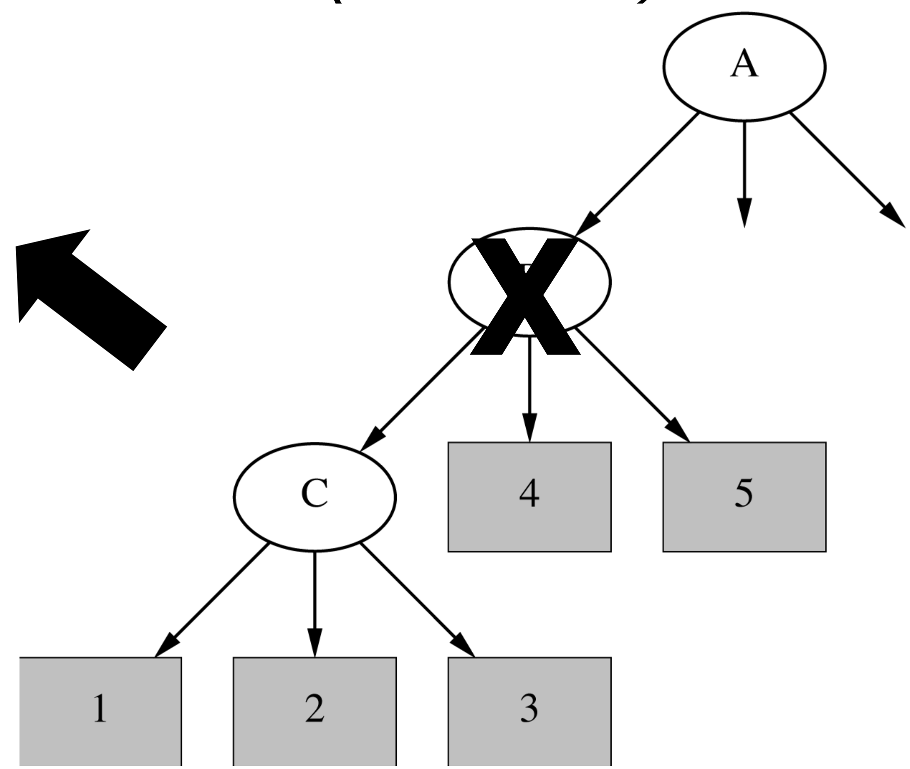


*Subtree raising



- 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
  Q: Why it would result in very little 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)

*Mean and variance

- Mean and variance for a Bernoulli trial: p, $p(1-p)$
- Expected success rate $f=S/N$
- Mean and variance for f: p, $p(1-p)/N$
- For large enough N, f follows a Normal distribution
- c% confidence interval $[-z \le X \le z]$ for random variable with 0 mean is given by:

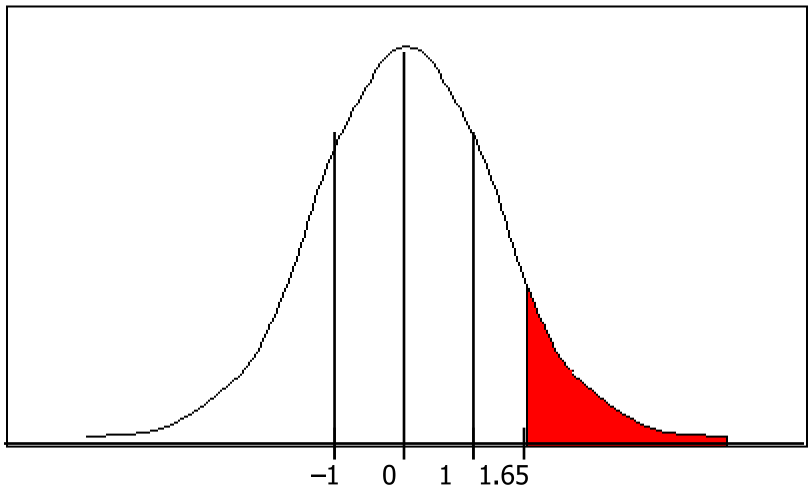
$$\Pr[-z \le X \le z] = c$$

- With a symmetric distribution:

$$\Pr[-z \le X \le z] = 1 - 2 \times \Pr[X \ge z]$$

*Confidence limits

- Confidence limits for the normal distribution with 0 mean and a variance of 1:



$\Pr[X \geq z]$	z
0.1%	3.09
0.5%	2.58
1%	2.33
5%	1.65
10%	1.28
20%	0.84
25%	0.69
40%	0.25

- Thus:

$$\Pr[-1.65 \leq X \leq 1.65] = 90\%$$

- To use this we have to reduce our random variable f to have 0 mean and unit variance

*Transforming *f*

- Transformed value for *f* : $$\frac{f-p}{\sqrt{p(1-p)/N}}$$

  (i.e. subtract the mean and divide by the *standard deviation*)

- Resulting equation: $$\Pr\left[-z \le \frac{f-p}{\sqrt{p(1-p)/N}} \le z\right] = c$$

- Solving for *p*:

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

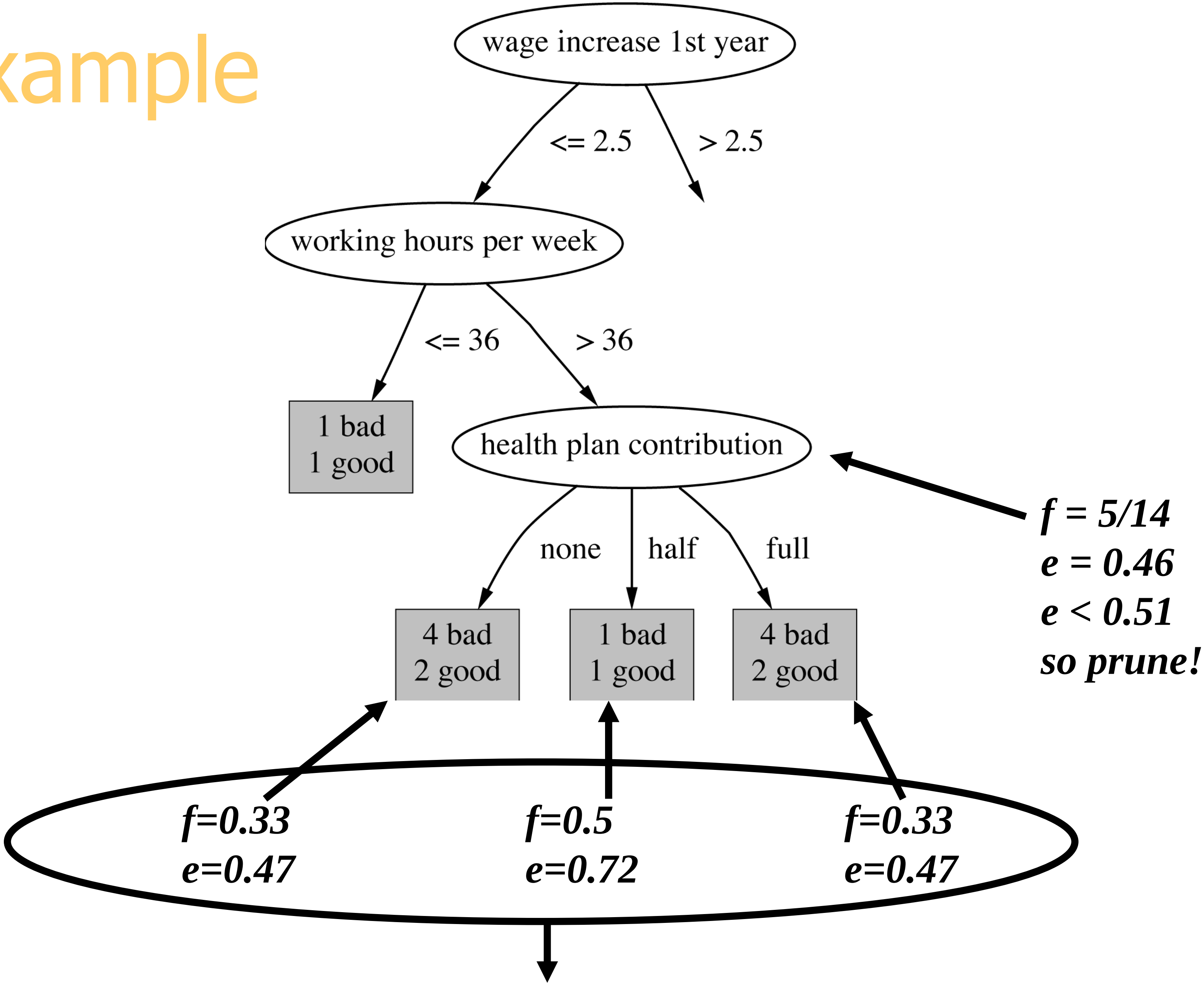
C4.5's method

- Error estimate for subtree is weighted sum of error estimates for all its leaves
- Error estimate for a node (upper bound):

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

- 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



Combined using ratios 6:2:6 gives 0.51

*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 – how?

How can we produce a set of rules from a decision tree?

From trees to rules – simple

- 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

C4.5rules: 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)

*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

Summary

- Decision Trees
  - splits – binary, multi-way
  - split criteria – entropy, gini, …
  - missing value treatment
  - pruning
  - rule extraction from trees
- Both C4.5 and CART are robust tools
- No method is always superior – experiment!