Machine Learning in Real World: C4.5

Industrial-strength algorithms

- For an algorithm to be useful in a wide range of real-world applications it must:
  - Permit numeric attributes with adaptive discretization
  - 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 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

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

Example

Sort order of temperature created on startup

outlook				sunny			sunny			sunny	sunny			sunny
temperature	64	65	68	69	70	71	72	72	75	75	80	81	83	85
tuple number	7	6	5	9	4	14	8	12	10	11	2	13	3	1

We decide to split on outlook first (sunny, rainy, overcast)

9 8 11 2 1

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

X

Potential optimal breakpoints

Breakpoints between values of the same class cannot be optimal

Missing as a separate value

- Missing value denoted “?” in C4.X (Null value)
- 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: blood sugar value 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

Questions:

- How should tests on attributes with different unknown values be handled?
- How should the partitioning be done in case of examples with unknown values?
- How should an unseen case with missing values be handled?

Missing values - advanced

- Info gain with unknown values during learning
  - Let T be the training set and X a test on an attribute with unknown values and F be the fraction of examples where the value is known.
  - Rewrite the gain:
    Gain(X) = probability that A is known * (info(T) – info_X(T))+
    probability that A is unknown * 0
    = F * (info(T) – info_X(T))

Missing values - advanced

Assume splitting is done with respect to attribute X

Consider instances w/o missing values

Split w.r.t. those instances

Distribute instances with missing values proportionally

Pruning

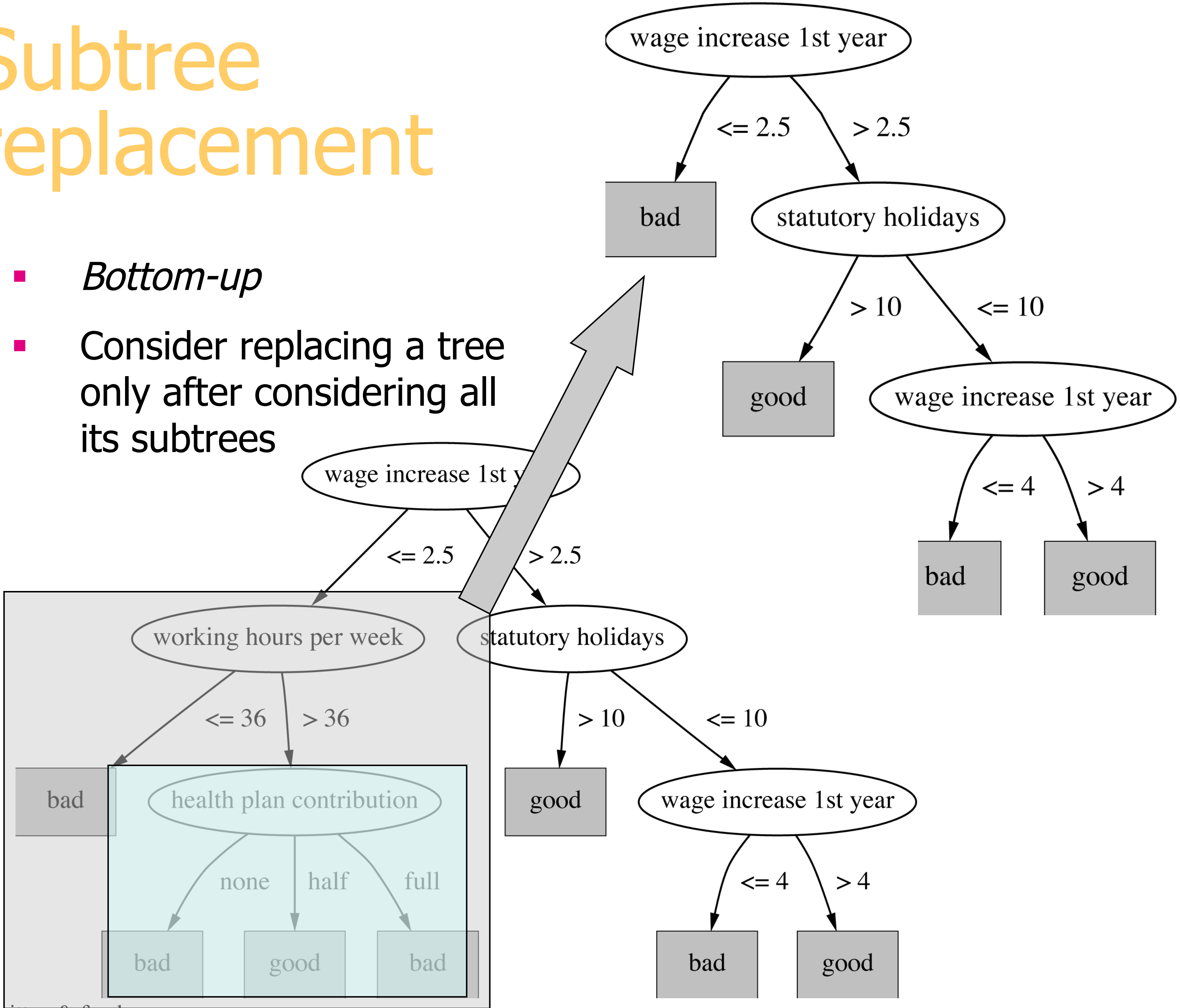
- Goal: Prevent overfitting to noise in the data
- Two strategies for “pruning” the decision tree:
  - *Postpruning* - take a fully-grown decision tree and discard unreliable parts
  - *Prepruning* - stop growing a branch when information becomes unreliable
- Postpruning preferred in practice—prepruning can “stop too early”

Post-pruning

- First, build full tree
- Then, prune it
  - Fully-grown tree shows all attribute interactions
- Two pruning operations:
  1. *Subtree replacement*
  2. *Subtree raising*

Subtree replacement

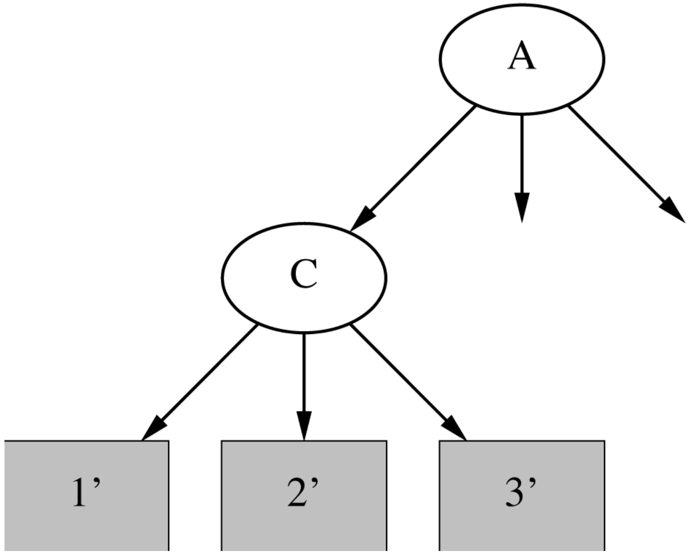
- *Bottom-up*
- Consider replacing a tree only after considering all its subtrees

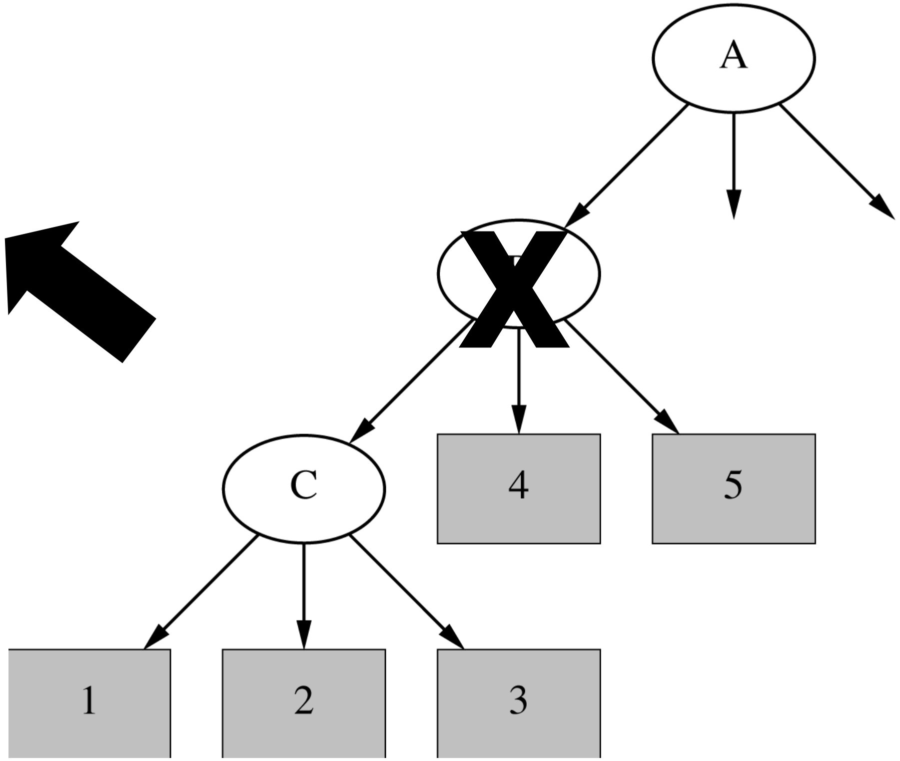


*Subtree raising

- 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
  Q: Why would it result in very little pruning?
- Use hold-out set for pruning ("reduced-error pruning")

Expected Error Pruning

- Approximate expected error assuming that we prune at a particular node.
- Approximate backed-up error from children assuming we did not prune.
- If expected error is less than backed-up error, prune.

Static Expected Error

- If we prune a node, it becomes a leaf labeled, C
- What will be the expected classification error at this leaf?

$$E(S) = \frac{N - n + k - 1}{N + k}$$

I would have used 1 here instead of k-1

S is the set of examples in a node
k is the number of classes
N examples in *S*
C the majority class in *S*
n out of *N* examples in *S* belong to *C*

This is called Laplace error estimate – it is based on the assumption that the distribution of probabilities that examples will belong to different classes is uniform.

Backed-Up Error

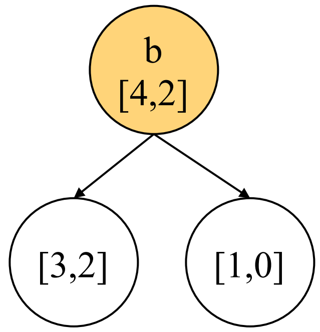
- For a non-leaf node
- Let children of Node be Node1, Node2, etc

$$BackedUpError(Node) = \sum_i P_i \times Error(Node_i)$$

Probabilities can be estimated by relative frequencies of attribute values in sets of examples that fall into child nodes

$$Error(Node) = \min(E(Node), BackedUpError(Node))$$

Example Calculation



Error Calculation for Pruning Example

- **Left child of *b* has class frequencies [3, 2]**

$$E = \frac{N - n + k - 1}{N + k} = \frac{5 - 3 + 2 - 1}{5 + 2} = 0.429$$

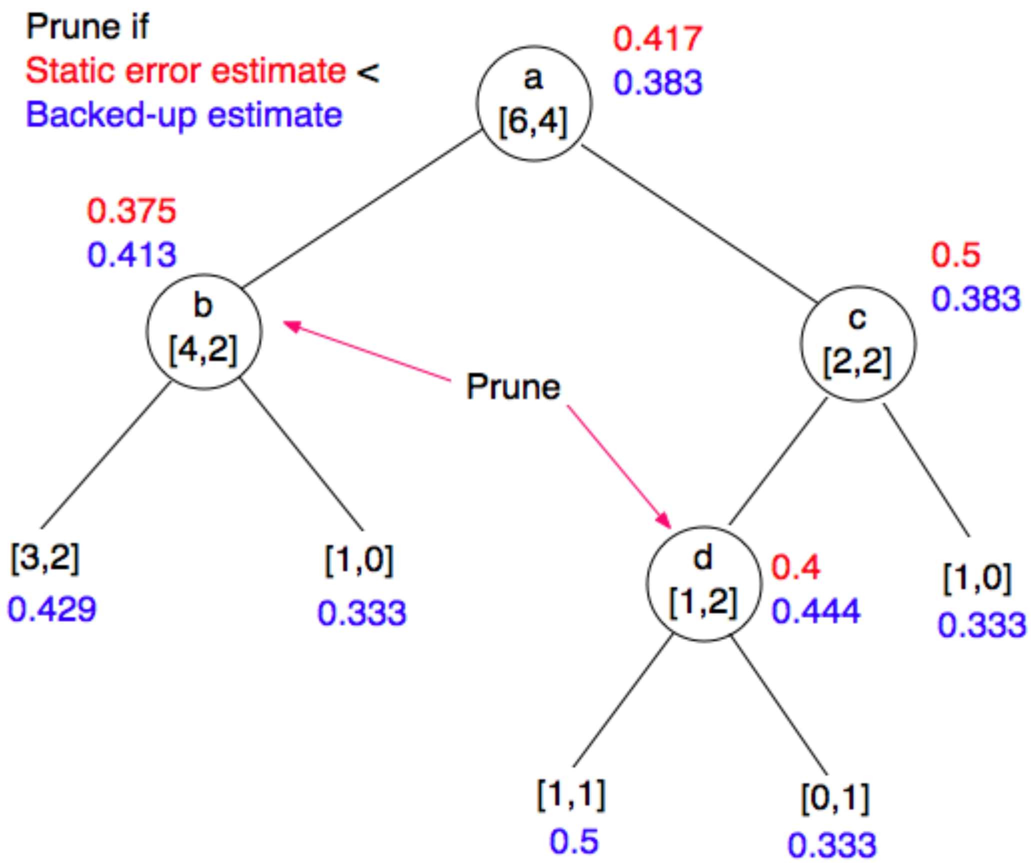
- **Right child has error of 0.333, calculated in the same way**
- **Static error estimate *E(b)* is 0.375, again calculated using the Laplace error estimate formula, with *N*=6, *n*=4, and *k*=2.**
- **Backed-up error is:**

$$BackedUpError(b) = (5/6)\times 0.429 + (1/6)\times 0.333 = 0.413$$

(5/6 and 1/6 because there are 4+2=6 examples handled by node *b*, of which 3+2=5 go to the left subtree and 1 to the right subtree.

- **Since backed-up estimate of 0.413 is greater than static estimate of 0.375, we prune the tree and use the static error of 0.375**

Example


Prune if
Static error estimate <
Backed-up estimate
a
[6,4]
0.417
0.383
0.375
0.413
b
[4,2]
Prune
0.5
0.383
c
[2,2]
[3,2]
0.429
[1,0]
0.333
d
[1,2]
0.4
0.444
[1,0]
0.333
[1,1]
0.5
[0,1]
0.333

*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: $O(n \log n)$
  - Cost for redistribution (on average): $O(\log n)$
- Total cost: $O(m\, n \log n) + O(n (\log n)^2)$

Windowing

- ID3 can deal with very large data sets by performing induction on subsets or *windows* onto the data
  1. Select a random subset of the whole set of training instances
  2. Use the induction algorithm to form a rule to explain the current window
  3. Scan through all of the training instances looking for exceptions to the rule
  4. Add the exceptions to the window
- Repeat steps 2 to 4 until there are no exceptions left

<table>
<tr><td colspan="2"></td><td colspan="2">Condition
(as determined by "Gold standard")</td><td></td></tr>
<tr><td colspan="2"></td><td>Condition Positive</td><td>Condition Negative</td><td></td></tr>
<tr><td rowspan="2">Test Outcome</td><td>Test Outcome Positive</td><td>True Positive</td><td>False Positive
(Type I error)</td><td>Positive predictive value = $\frac{\Sigma \text{ True Positive}}{\Sigma \text{ Test Outcome Positive}}$</td></tr>
<tr><td>Test Outcome Negative</td><td>False Negative
(Type II error)</td><td>True Negative</td><td>Negative predictive value = $\frac{\Sigma \text{ True Negative}}{\Sigma \text{ Test Outcome Negative}}$</td></tr>
<tr><td colspan="2"></td><td>Sensitivity = $\frac{\Sigma \text{ True Positive}}{\Sigma \text{ Condition Positive}}$</td><td>Specificity = $\frac{\Sigma \text{ True Negative}}{\Sigma \text{ Condition Negative}}$</td><td></td></tr>
</table>