Decision Trees

Nguyen Hung Son

Lecture plan

- Decision tree approach
- Decision rules
- Mining decision rules from large data


Outline

- Top-Down Decision Tree Construction
- Choosing the Splitting Attribute
- Information Gain and Gain Ratio


DECISION TREE

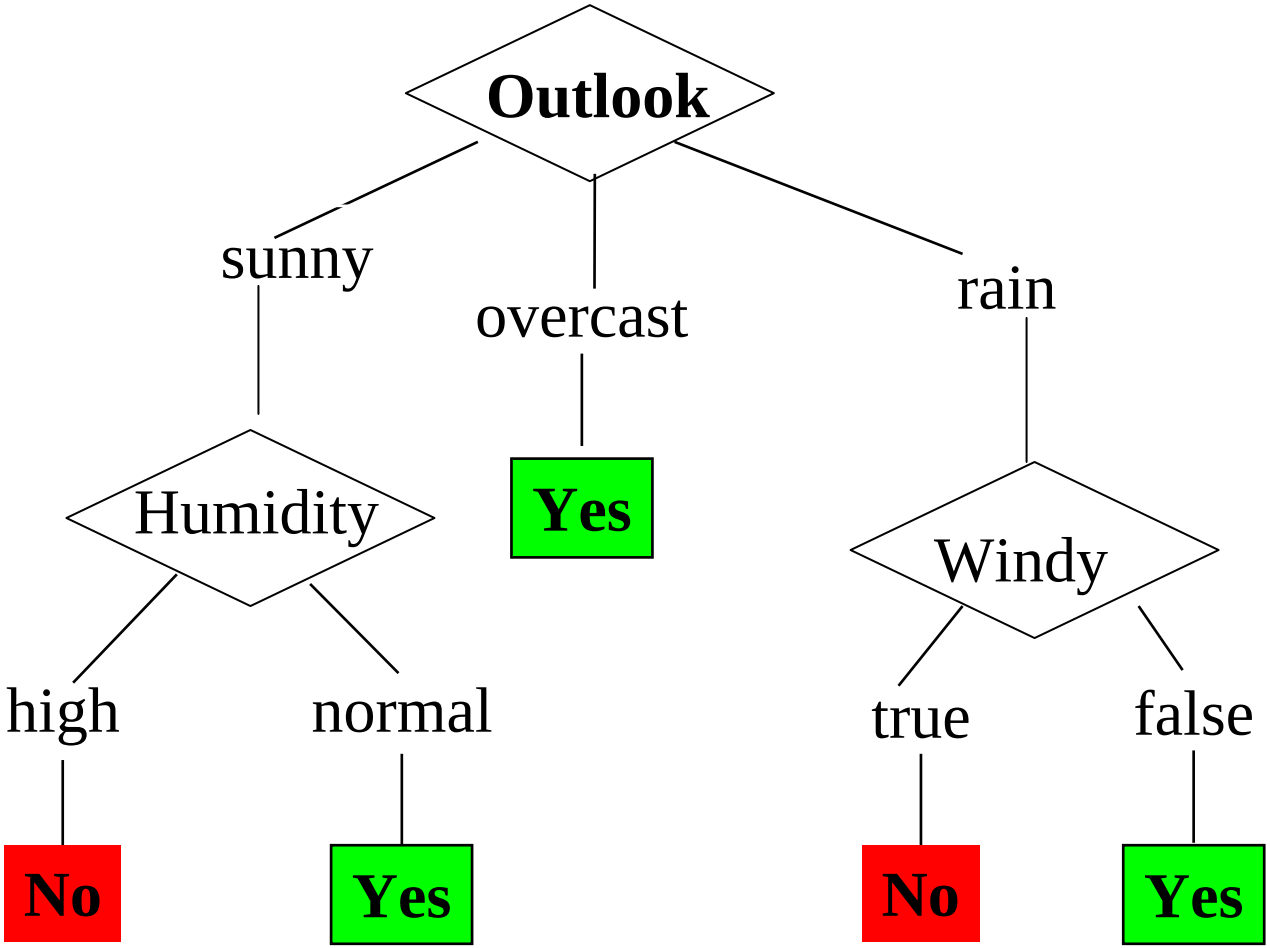
- An internal node is a test on an attribute.
- A branch represents an outcome of the test, e.g., Color=red.
- A leaf node represents a class label or class label distribution.
- At each node, one attribute is chosen to split training examples into distinct classes as much as possible
- A new case is classified by following a matching path to a leaf node.

Weather Data: Play or not Play?

Outlook	Temperature	Humidity	Windy	Play?
sunny	hot	high	false	No
sunny	hot	high	true	No
overcast	hot	high	false	Yes
rain	mild	high	false	Yes
rain	cool	normal	false	Yes
rain	cool	normal	true	No
overcast	cool	normal	true	Yes
sunny	mild	high	false	No
sunny	cool	normal	false	Yes
rain	mild	normal	false	Yes
sunny	mild	normal	true	Yes
overcast	mild	high	true	Yes
overcast	hot	normal	false	Yes
rain	mild	high	true	No

Note:
Outlook is the Forecast,
no relation to Microsoft email program

Example Tree for "Play?"



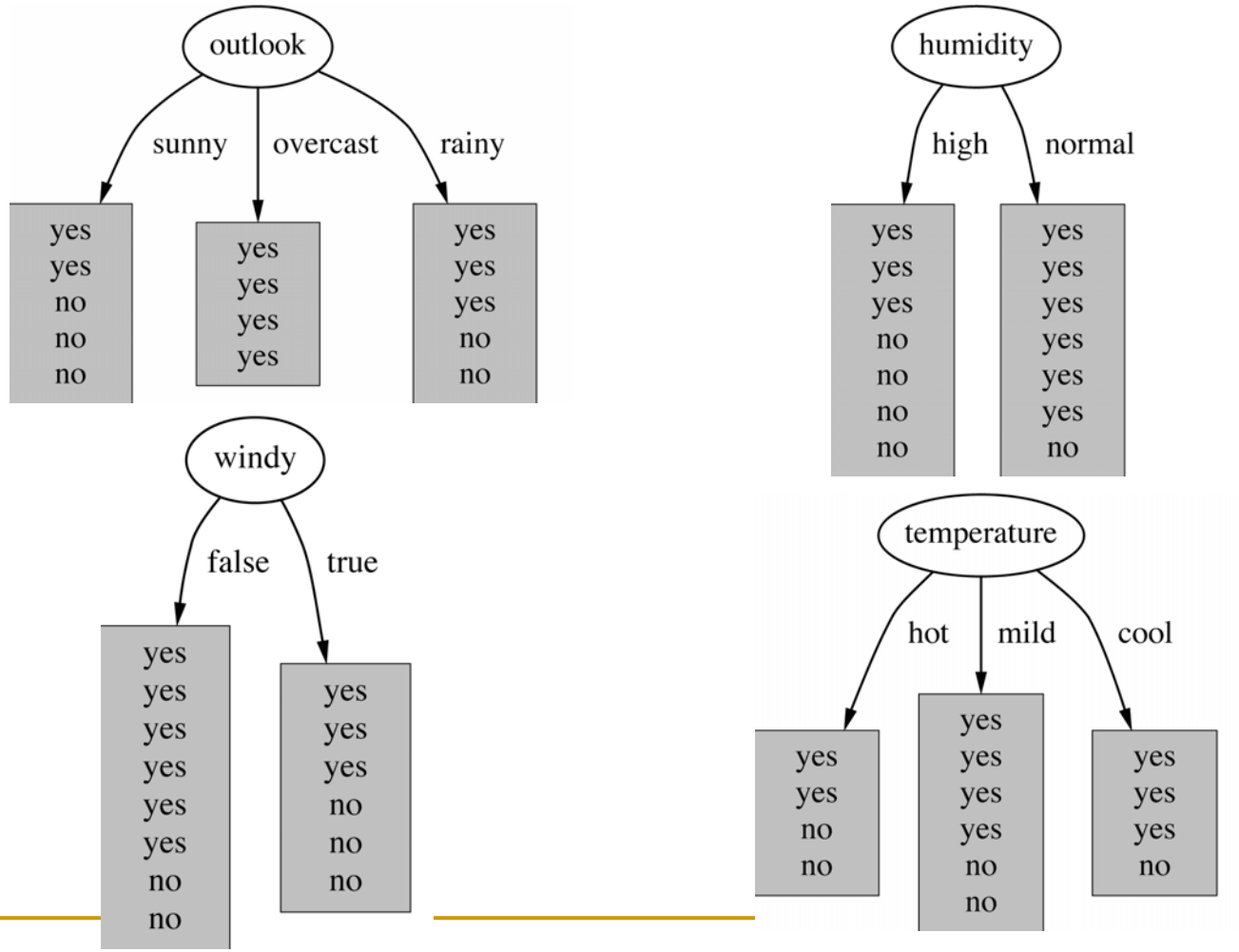
Building Decision Tree [Q93]

- Top-down tree construction
  - At start, all training examples are at the root.
  - Partition the examples recursively by choosing one attribute each time.
- Bottom-up tree pruning
  - Remove subtrees or branches, in a bottom-up manner, to improve the estimated accuracy on new cases.


Choosing the Splitting Attribute

- At each node, available attributes are evaluated on the basis of separating the classes of the training examples. A Goodness function is used for this purpose.
- Typical goodness functions:
  - information gain (ID3/C4.5)
  - information gain ratio
  - gini index

Which attribute to select?



A criterion for attribute selection

- Which is the best attribute?
  - The one which will result in the smallest tree
  - Heuristic: choose the attribute that produces the "purest" nodes
- Popular *impurity criterion*: *information gain*
  - Information gain increases with the average purity of the subsets that an attribute produces
- Strategy: choose attribute that results in greatest information gain

Computing information

- Information is measured in *bits*
  - Given a probability distribution, the info required to predict an event is the distribution's *entropy*
  - Entropy gives the information required in bits (this can involve fractions of bits!)
- Formula for computing the entropy:

$$\text{entropy}(p_1, p_2, \ldots, p_n) = -p_1 \log p_1 - p_2 \log p_2 \ldots - p_n \log p_n$$

*Claude Shannon

Born: 30 April 1916

Died: 23 February 2001

"Father of information theory"

Claude Shannon, who has died aged 84, perhaps more than anyone laid the groundwork for today's digital revolution. His exposition of information theory, stating that all information could be represented mathematically as a succession of noughts and ones, facilitated the digital manipulation of data without which today's information society would be unthinkable.

Shannon's master's thesis, obtained in 1940 at MIT, demonstrated that problem solving could be achieved by manipulating the symbols 0 and 1 in a process that could be carried out automatically with electrical circuitry. That dissertation has been hailed as one of the most significant master's theses of the 20th century. Eight years later, Shannon published another landmark paper, *A Mathematical Theory of Communication*, generally taken as his most important scientific contribution.

Shannon applied the same radical approach to cryptography research, in which he later became a consultant to the US government.

Many of Shannon's pioneering insights were developed before they could be applied in practical form. He was truly a remarkable man, yet unknown to most of the world.

Example: attribute "Outlook"

- "Outlook" = "Sunny":

$$\text{info}([2,3]) = \text{entropy}(2/5,3/5) = -2/5\log(2/5) - 3/5\log(3/5) = 0.971\,\text{bits}$$

- "Outlook" = "Overcast":

$$\text{info}([4,0]) = \text{entropy}(1,0) = -1\log(1) - 0\log(0) = 0\,\text{bits}$$

Note: log(0) is not defined, but we evaluate 0*log(0) as zero

- "Outlook" = "Rainy":

$$\text{info}([3,2]) = \text{entropy}(3/5,2/5) = -3/5\log(3/5) - 2/5\log(2/5) = 0.971\,\text{bits}$$

- Expected information for attribute:

$$\text{info}([3,2],[4,0],[3,2]) = (5/14)\times 0.971 + (4/14)\times 0 + (5/14)\times 0.971$$

$$= 0.693\,\text{bits}$$

Computing the information gain

- Information gain:

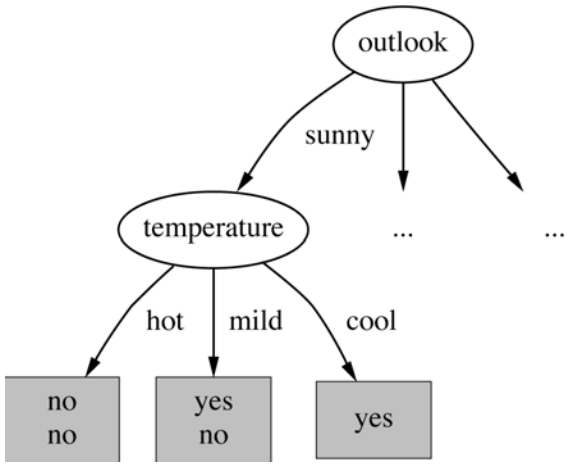
(information before split) – (information after split)

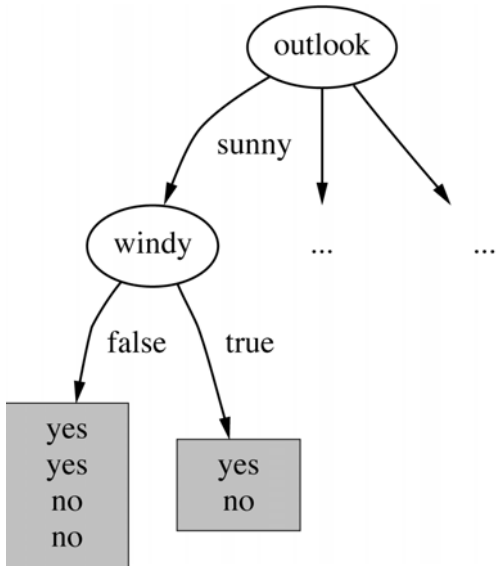
$$\text{gain("Outlook")} = \text{info}([9,5]) - \text{info}([2,3],[4,0],[3,2]) = 0.940 - 0.693 = 0.247 \text{ bits}$$

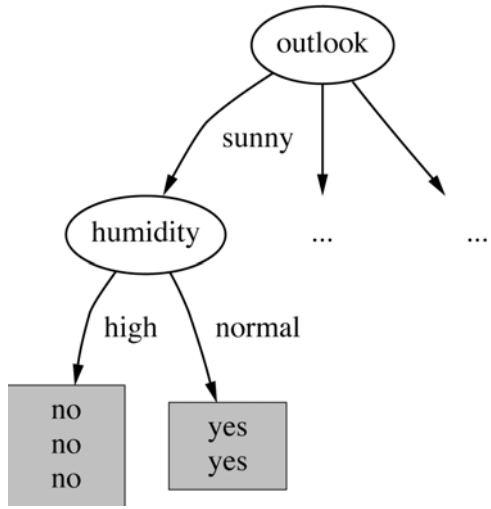
- Information gain for attributes from weather data:

$$\text{gain("Outlook")} = 0.247 \text{ bits}$$
$$\text{gain("Temperature")} = 0.029 \text{ bits}$$
$$\text{gain("Humidity")} = 0.152 \text{ bits}$$
$$\text{gain("Windy")} = 0.048 \text{ bits}$$

Continuing to split





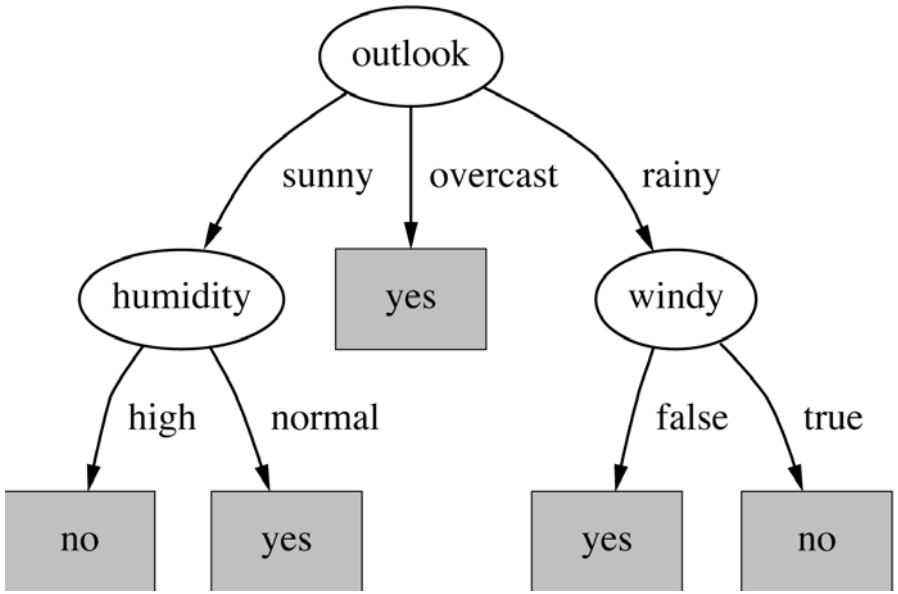


gain("Humidity") = 0.971 bits

gain("Temperature") = 0.571 bits

gain("Windy") = 0.020 bits

The final decision tree



- Note: not all leaves need to be pure; sometimes identical instances have different classes

  ⇒ Splitting stops when data can't be split any further

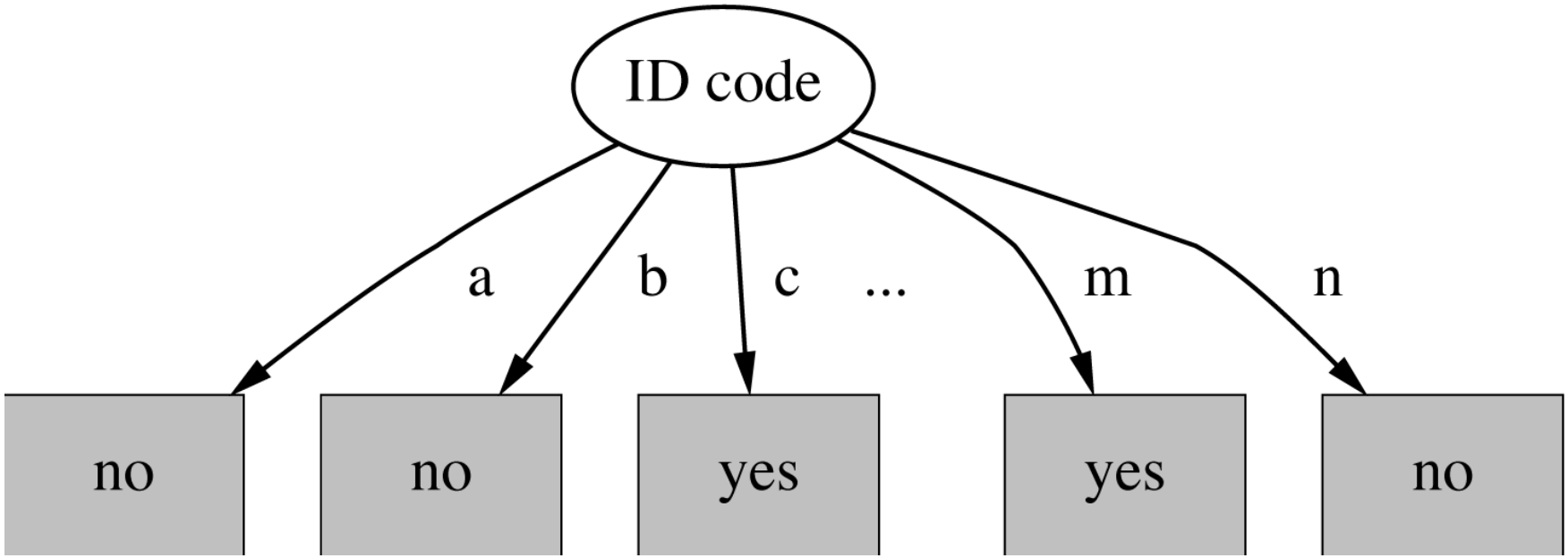
Highly-branching attributes

- Problematic: attributes with a large number of values (extreme case: ID code)
- Subsets are more likely to be pure if there is a large number of values
  - ⇒ Information gain is biased towards choosing attributes with a large number of values
  - ⇒ This may result in *overfitting* (selection of an attribute that is non-optimal for prediction)

Weather Data with ID code

ID	Outlook	Temperature	Humidity	Windy	Play?
A	sunny	hot	high	false	No
B	sunny	hot	high	true	No
C	overcast	hot	high	false	Yes
D	rain	mild	high	false	Yes
E	rain	cool	normal	false	Yes
F	rain	cool	normal	true	No
G	overcast	cool	normal	true	Yes
H	sunny	mild	high	false	No
I	sunny	cool	normal	false	Yes
J	rain	mild	normal	false	Yes
K	sunny	mild	normal	true	Yes
L	overcast	mild	high	true	Yes
M	overcast	hot	normal	false	Yes
N	rain	mild	high	true	No

Split for ID Code Attribute



Entropy of split = 0 (since each leaf node is “pure”, having only one case.

Information gain is maximal for ID code

Gain ratio

- *Gain ratio*: a modification of the information gain that reduces its bias on high-branch attributes
- Gain ratio should be
  - Large when data is evenly spread
  - Small when all data belong to one branch
- Gain ratio takes number and size of branches into account when choosing an attribute
  - It corrects the information gain by taking the *intrinsic information* of a split into account (i.e. how much info do we need to tell which branch an instance belongs to)

Gain Ratio and Intrinsic Info.

- Intrinsic information: entropy of distribution of instances into branches

$$IntrinsicInfo(S,A) \equiv -\sum \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}.$$

- *Gain ratio* (Quinlan'86) normalizes info gain by:

$$GainRatio(S,A) = \frac{Gain(S,A)}{IntrinsicInfo(S,A)}.$$

Computing the gain ratio

- Example: intrinsic information for ID code

$$info([1,1,\ldots,1) = 14 \times (-1/14 \times \log 1/14) = 3.807\,\text{bits}$$

- **Importance of attribute decreases as intrinsic information gets larger**
- Example of gain ratio:

$$\text{gain\_ratio}(\text{"Attribute"}) = \frac{\text{gain}(\text{"Attribute"})}{\text{intrinsic\_info}(\text{"Attribute"})}$$

- Example:

$$\text{gain\_ratio}(\text{"ID\_code"}) = \frac{0.940\,\text{bits}}{3.807\,\text{bits}} = 0.246$$

Gain ratios for weather data

Outlook		**Temperature**	
Info:	0.693	Info:	0.911
Gain: 0.940-0.693	0.247	Gain: 0.940-0.911	0.029
Split info: info([5,4,5])	1.577	Split info: info([4,6,4])	1.362
Gain ratio: 0.247/1.577	0.156	Gain ratio: 0.029/1.362	0.021

Humidity		**Windy**	
Info:	0.788	Info:	0.892
Gain: 0.940-0.788	0.152	Gain: 0.940-0.892	0.048
Split info: info([7,7])	1.000	Split info: info([8,6])	0.985
Gain ratio: 0.152/1	0.152	Gain ratio: 0.048/0.985	0.049

More on the gain ratio

- "Outlook" still comes out top
- However: "ID code" has greater gain ratio
  - Standard fix: *ad hoc* test to prevent splitting on that type of attribute
- Problem with gain ratio: it may overcompensate
  - May choose an attribute just because its intrinsic information is very low
  - Standard fix:
    - First, only consider attributes with greater than average information gain
    - Then, compare them on gain ratio

*CART Splitting Criteria: Gini Index

- If a data set T contains examples from n classes, gini index, gini(T) is defined as

$$gini(T) = 1 - \sum_{j=1}^{n} p_j^2$$

where p_j is the relative frequency of class j in T.

gini(T) is minimized if the classes in T are skewed.

*Gini Index

After splitting T into two subsets T1 and T2 with sizes N1 and N2, the gini index of the split data is defined as

$$gini_{split}(T) = \frac{N_1}{N} gini(T_1) + \frac{N_2}{N} gini(T_2)$$

- The attribute providing smallest $gini_{split}(T)$ is chosen to split the node.

Discussion

- Algorithm for top-down induction of decision trees ("ID3") was developed by Ross Quinlan
  - Gain ratio just one modification of this basic algorithm
  - Led to development of C4.5, which can deal with numeric attributes, missing values, and noisy data
- Similar approach: CART (to be covered later)
- There are many other attribute selection criteria! (But almost no difference in accuracy of result.)


Summary

- Top-Down Decision Tree Construction
- Choosing the Splitting Attribute
- Information Gain biased towards attributes with a large number of values
- Gain Ratio takes number and size of branches into account when choosing an attribute


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 ≥ 71.5: yes/5, no/3

  - Info([4,2],[5,3])
    = 6/14 info([4,2]) + 8/14 info([5,3])
    = 0.939 bits

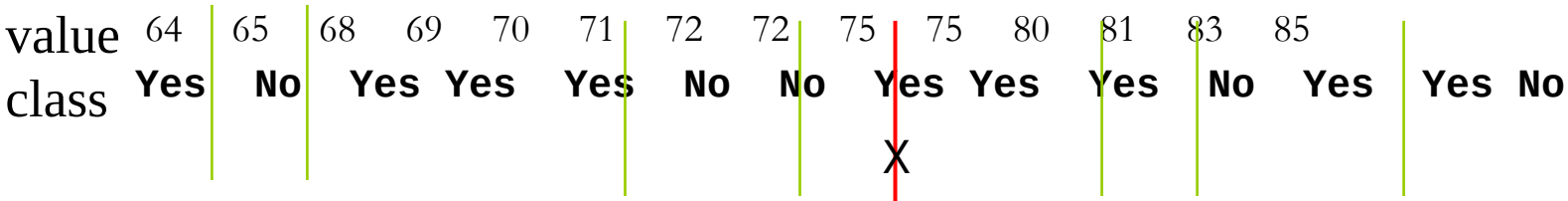
- Place split points halfway between values
- Can evaluate all split points in one pass!

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)



Potential optimal breakpoints

Breakpoints between values of the same class cannot be optimal

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

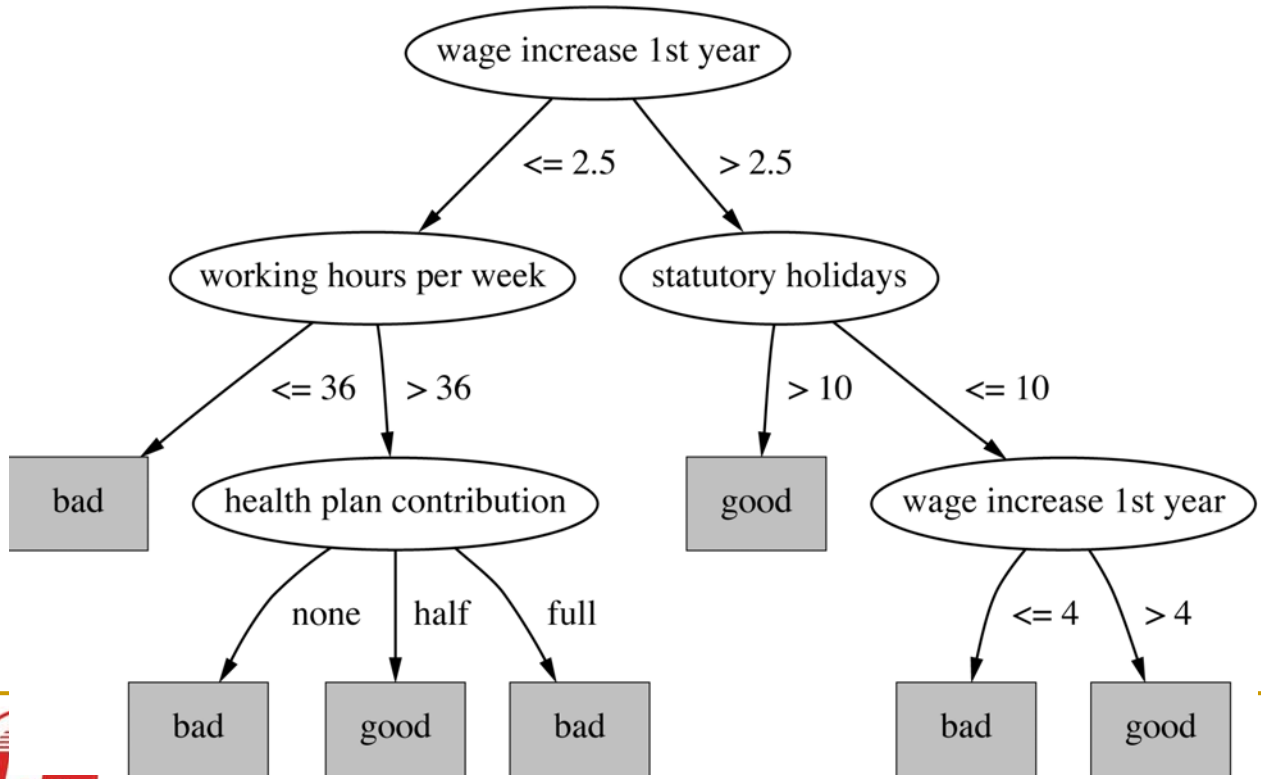
- Pre-pruning may stop the growth process prematurely: *early stopping*
- Classic example: XOR/Parity-problem
  - No *individual* attribute exhibits any significant association to the class
  - Structure is only visible in fully expanded tree
  - Pre-pruning won't expand the root node
- But: XOR-type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

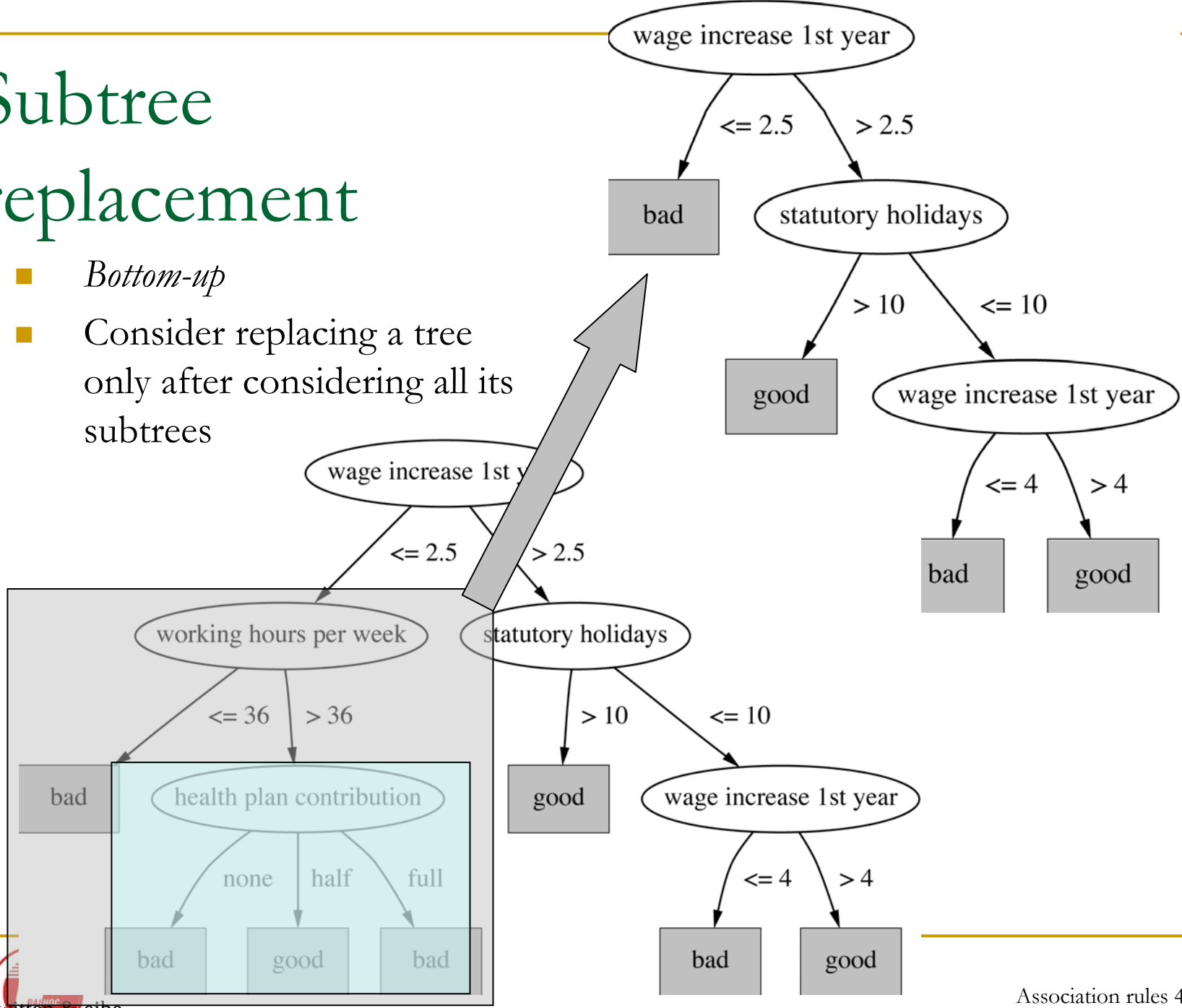
Subtree replacement

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



Subtree replacement

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



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:

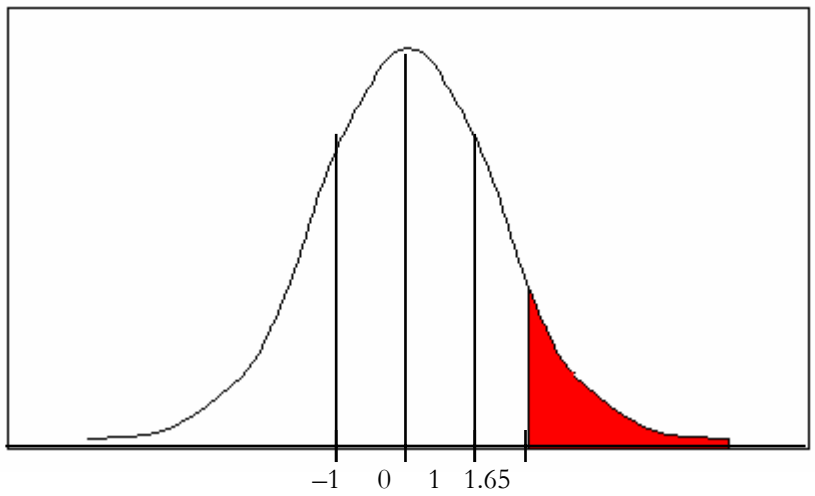
- With a symmetric distribution:

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

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

*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

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

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

*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*:

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

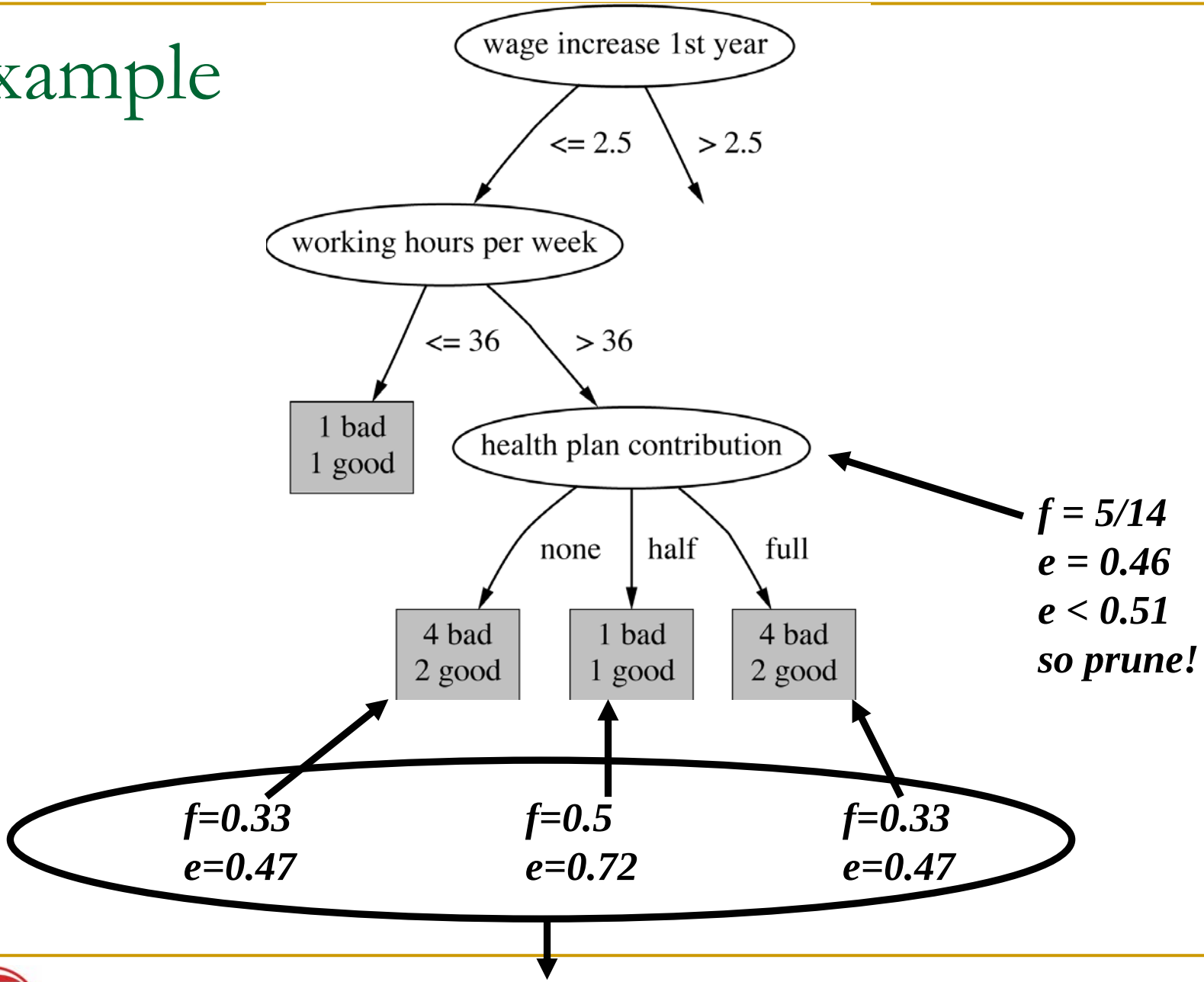
C4.5's method

- 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


wage increase 1st year
<= 2.5
> 2.5
working hours per week
<= 36
> 36
1 bad
1 good
health plan contribution
none
half
full
4 bad
2 good
1 bad
1 good
4 bad
2 good
f = 5/14
e = 0.46
e < 0.51
so prune!
f=0.33
e=0.47
f=0.5
e=0.72
f=0.33
e=0.47
Combined using ratios 6:2:6 gives 0.51

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

C4.5rules: 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)


Rule based classifiers

- Each *classification rule* is of form

$$r : (Condition) \rightarrow y$$

  - LHS of the rule (*Condition*), called *rule antecedent* or *pre-condition*, is a conjunction of attribute tests
  - RHS, also called the *rule consequent*, is the class label

- *Rule set:*

$$R = \{r_1, r_2, \ldots, r_n\}$$


Classifying Instances with Rules

- A rule *r* ***covers*** an instance x if the attributes of the instance satisfy the condition of the rule
- Example
  - Rule:

    r : (Age < 35) $\wedge$ (Status = Married) $\rightarrow$ Cheat=No
  - Instances:

    x_1 : (Age=29, Status=Married, Refund=No)

    x_2 : (Age=28, Status=Single, Refund=Yes)

    x_3 : (Age=38, Status=Divorced, Refund=No)
  - Only x_1 is covered by the rule r

Classifying Instances with Rules

- **Rules may not be mutually exclusive**
  - More than one rule may cover the same instance
- **Strategies:**
  - Strict enforcement of mutual exclusiveness
    - Avoid generating rules that have overlapping coverage with previously selected rules
  - Ordered rules
    - Rules are rank ordered according to their priority
  - Voting
    - Allow an instance to trigger multiple rules, and consider the consequent of each triggered rule as a vote for that particular class

- **Rules may not be exhaustive**
- **Strategy:**
  - A *default rule*

    $r_d : ( ) \rightarrow y_d$

    can be added
  - The default rule has an empty antecedent and is applicable when all other rules have failed
  - y_d is known as *default class* and is often assigned to the majority class

Advantages of Rule Based Classifiers

- As highly expressive as decision trees
- Easy to interpret
- Easy to generate
- Can classify new instances rapidly
- Performance comparable to decision trees

Definition

- Coverage of a rule:
  - Number (or fraction) of instances that satisfy the antecedent of a rule
- Accuracy of a rule:
  - Fraction of instances that satisfy both the antecedent and consequent of a rule
- Length:
  - Number of descriptors

Example

(Marital Status=Married) → No

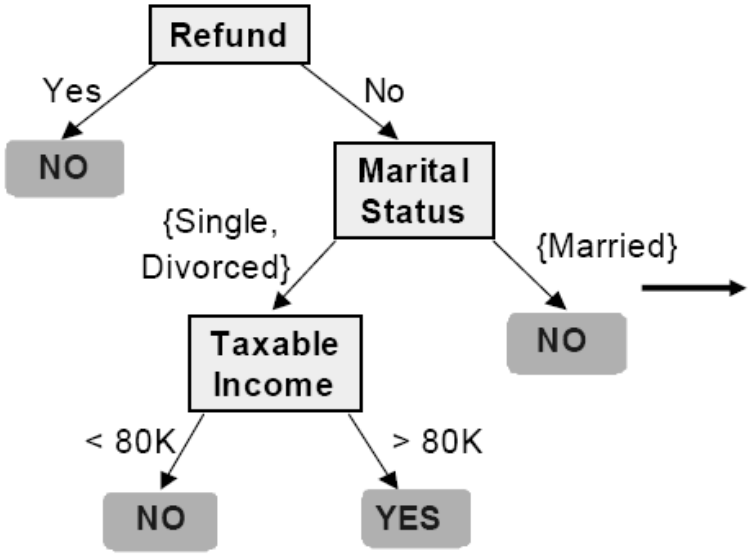
- Coverage = 40%,
- Accuracy = 100%
- Length = 1

ID	Refund	Marital Status	Income	Cheat
1	Yes	Single	125K	No
2	No	**Married**	100K	No
3	No	Single	70K	No
4	Yes	**Married**	120K	No
5	No	Divorced	95K	Yes
6	No	**Married**	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	**Married**	75K	No
10	**No**	**Single**	**90K**	Yes

Construction of a Rule Based Classifier from data

- Generate an initial set of rules
  - Direct Method:
    - Extract rules directly from data
    - Examples: RIPPER, CN2, Holte's 1R, Boolean reasoning
  - Indirect Method:
    - Extract rules from other classification methods (e.g. decision trees)
    - Example: C4.5 rules
- Rules are pruned and simplified
- Rules can be order to obtain a rule set *R*
- Rule set *R* can be further optimized

Indirect Method: Conversion from Decision Trees



Classification Rules

(Refund=Yes) ==> No

(Refund=No, Marital Status={Single,Divorced}, Taxable Income<80K) ==> No

(Refund=No, Marital Status={Single,Divorced}, Taxable Income>80K) ==> Yes

(Refund=No, Marital Status={Married}) ==> No

- Rules are mutually exclusive and exhaustive
- Rule set contains as much information as the tree
- Rules can be simplified

$(\text{Refund}=\text{No}) \wedge (\text{Status}=\text{Married}) \rightarrow \text{No} \quad ==> \quad (\text{Status}=\text{Married}) \rightarrow \text{No}$

Indirect Method: C4.5 rules

- Creating an initial set of rules
  - Extract rules from an un-pruned decision tree
  - For each rule, $r : A \rightarrow y$
    - Consider alternative rules r': $A' \rightarrow y$, where A' is obtained by removing one of the conjuncts in A
    - Replace r by r' if it has a lower pessimistic error
    - Repeat until we can no longer improve the generalization error
- Ordering the rules
  - Instead of ordering the rules, order subsets of rules
    - Each subset is a collection of rules with the same consequent (class)
    - The subsets are then ordered in the increasing order of

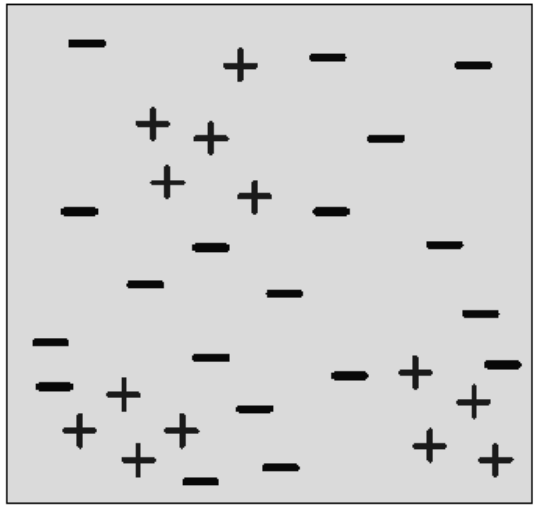
      Description length = L(exceptions) + $g \cdot L$(model)
    - where g is a parameter that takes in to account the presence of redundant attributes in a rule set. Default value is 0.5

Direct method: Sequential covering

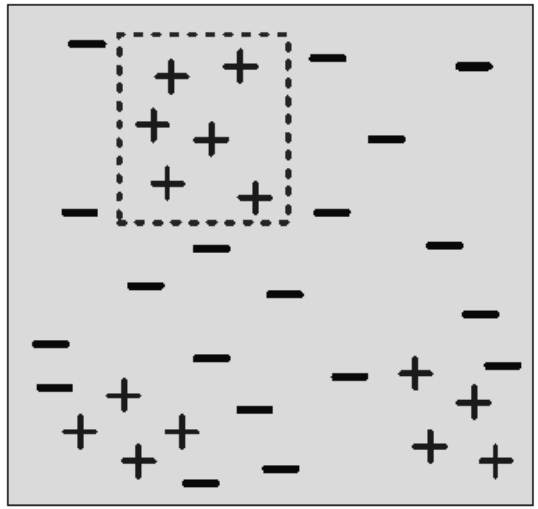
(E: training examples, A: set of attributes)

1. Let $R = \{\ \}$ be the initial rule set
2. While stopping criteria is not met
   1. r := Learn-One-Rule (E, A);
   2. Remove instances from E that are covered by r;
   3. Add r to rule set: $R = R + \{r\}$;

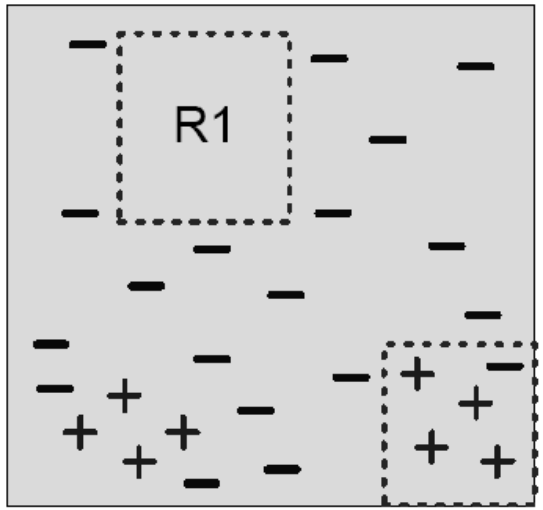
- Ex. Stopping criteria = „E is empty”



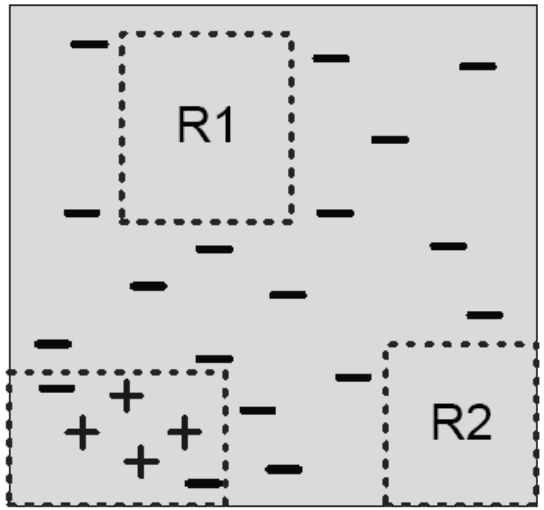
(i) Original Data



(ii) Step 1



(iii) Step 2



(iv) Step 3

Learn one rule (1R)

- The objective of this function is to extract the best rule that covers the current set of training instances
  - What is the strategy used for rule growing
  - What is the evaluation criteria used for rule growing
  - What is the stopping criteria for rule growing
  - What is the pruning criteria for generalizing the rule

Learn One Rule: Rule Growing Strategy

- General-to-specific approach
  - It is initially assumed that the best rule is the empty rule, $r: \{\} \rightarrow y$, where y is the majority class of the instances
  - Iteratively add new conjuncts to the LHS of the rule until the stopping criterion is met
- Specific-to-general approach
  - A positive instance is chosen as the initial seed for a rule
  - The function keeps refining this rule by generalizing the conjuncts until the stopping criterion is met

Rule Evaluation and Stopping Criteria

- Evaluate rules using rule evaluation metric
  - Accuracy
  - Coverage
  - Entropy
  - Laplace
  - M-estimate
- A typical condition for terminating the rule growing process is to compare the evaluation metric of the previous candidate rule to the newly grown rule

Learn 1R

- Rule Pruning
  - – Each extracted rule can be pruned to improve their ability to generalize beyond the training instances
  - Pruning can be done by removing one of the conjuncts of the rule and then testing it against a validation set
- Instance Elimination
  - – Instance elimination prevents the same rule from being generated again
  - Positive instances must be removed after each rule is extracted
  - Some rule based classifiers keep negative instances, while some remove them prior to generating next rule

RIPPER

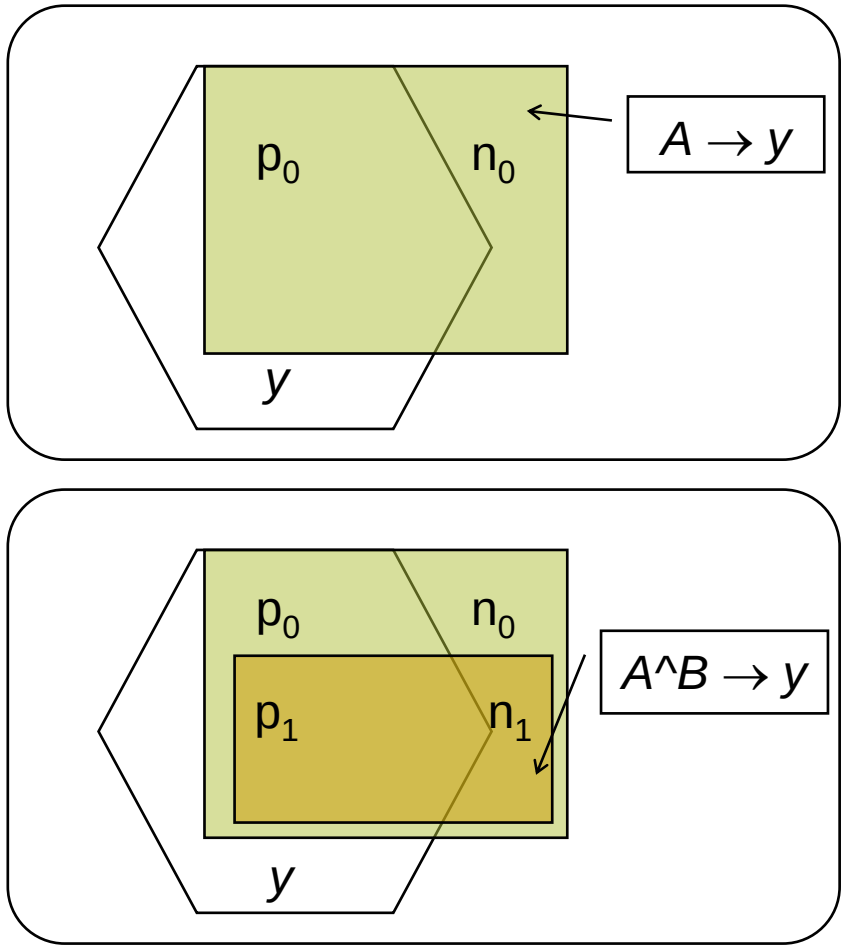
- For 2-class problem, choose one of the classes as positive class, and the other as negative class
  - Learn rules for positive class
  - Negative class will be default class
- For multi-class problem
  - Order the classes according to increasing class prevalence (fraction of instances that belong to a particular class)
  - Learn the rule set for smallest class first, treat the rest as negative class
  - Repeat with next smallest class as positive class

Foil's Information Gain

- Compares the performance of a rule before and after adding a new conjunct
- Foil's information gain is defined as

  $= t \cdot [\, log_2(p_1/(p_1 + n_1)) - log_2(p_0/(p_0 + n_0)) \,]$

  where t is the number of positive instances covered by both r and r'



Direct Method: RIPPER

- Growing a rule:
  - Start from empty rule
  - Add conjuncts as long as they improve Foil's information gain
  - Stop when rule no longer covers negative examples
  - Prune the rule immediately using incremental reduced error pruning
  - Measure for pruning: $v = (p - n) / (p + n)$
    - p: number of positive examples covered by the rule in the validation set
    - n: number of negative examples covered by the rule in the validation set
  - Pruning method: delete any final sequence of conditions that maximizes v

RIPPER: Building a Rule Set

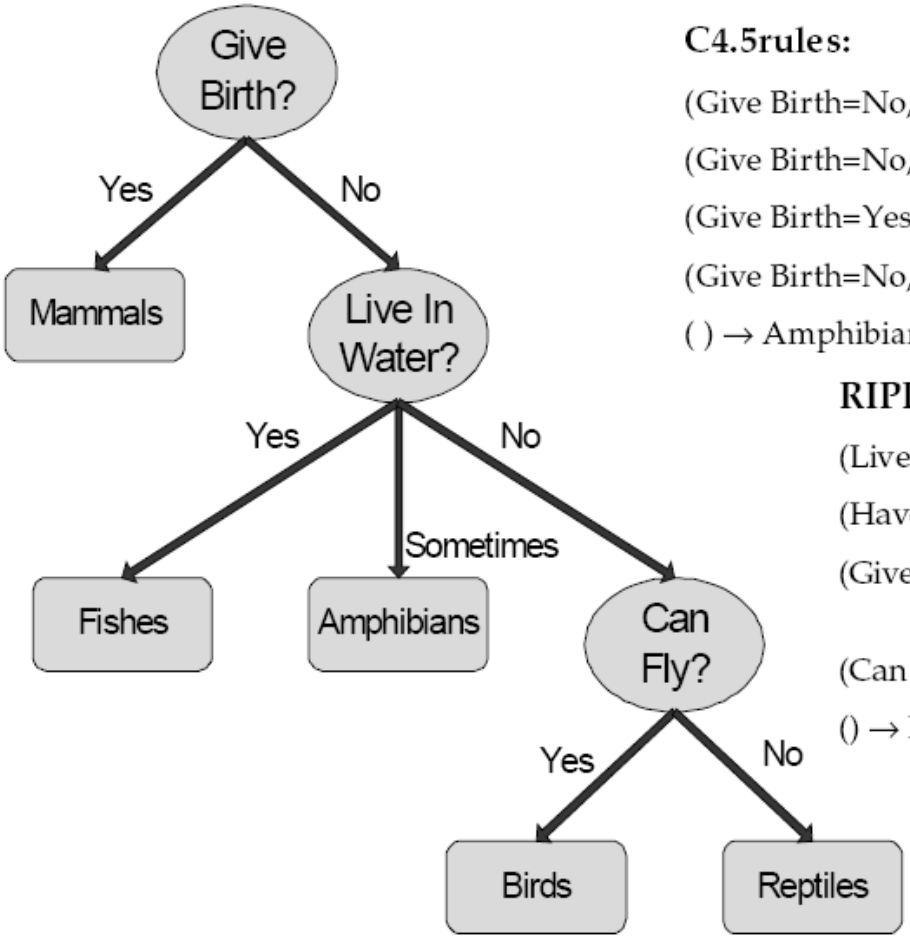
- Use sequential covering algorithm
  - Finds the best rule that covers the current set of positive examples
  - Eliminate both positive and negative examples covered by the rule
- Each time a rule is added to the rule set, compute the description length
  - Stop adding new rules when the new description length is *d* bits longer than the smallest description length obtained so far. *d* is often chosen as 64 bits


RIPPER: Optimize the rule set:

- For each rule *r* in the rule set ***R***
  - Consider 2 alternative rules:
    - Replacement rule (*r**): grow new rule from scratch
    - Revised rule (*r'*): add conjuncts to extend the rule *r*
  - Compare the rule set for *r* against the rule set for *r** and *r'*
  - Choose rule set that minimizes MDL principle
- Repeat rule generation and rule optimization for the remaining positive examples

Name	Give Birth	Lay Eggs	Can Fly	Live in Water	Have Legs	Class
human	yes	no	no	no	yes	mammals
python	no	yes	no	no	no	reptiles
salmon	no	yes	no	yes	no	fishes
whale	yes	no	no	yes	no	mammals
frog	no	yes	no	sometimes	yes	amphibians
komodo	no	yes	no	no	yes	reptiles
bat	yes	no	yes	no	yes	mammals
pigeon	no	yes	yes	no	yes	birds
cat	yes	no	no	no	yes	mammals
leopard shark	yes	no	no	yes	no	fishes
turtle	no	yes	no	sometimes	yes	reptiles
penguin	no	yes	no	sometimes	yes	birds
porcupine	yes	no	no	no	yes	mammals
eel	no	yes	no	yes	no	fishes
salamander	no	yes	no	sometimes	yes	amphibians
gila monster	no	yes	no	no	yes	reptiles
platypus	no	yes	no	no	yes	mammals
owl	no	yes	yes	no	yes	birds
dolphin	yes	no	no	yes	no	mammals
eagle	no	yes	yes	no	yes	birds

C 4.5 rules vs. RIPPER



C4.5rules:

(Give Birth=No, Can Fly=Yes) → Birds

(Give Birth=No, Live in Water=Yes) → Fishes

(Give Birth=Yes) → Mammals

(Give Birth=No, Can Fly=No, Live in Water=No) → Reptiles

( ) → Amphibians

RIPPER:

(Live in Water=Yes) → Fishes

(Have Legs=No) → Reptiles

(Give Birth=No, Can Fly=No, Live In Water=No) → Reptiles

(Can Fly=Yes,Give Birth=No) → Birds

() → Mammals