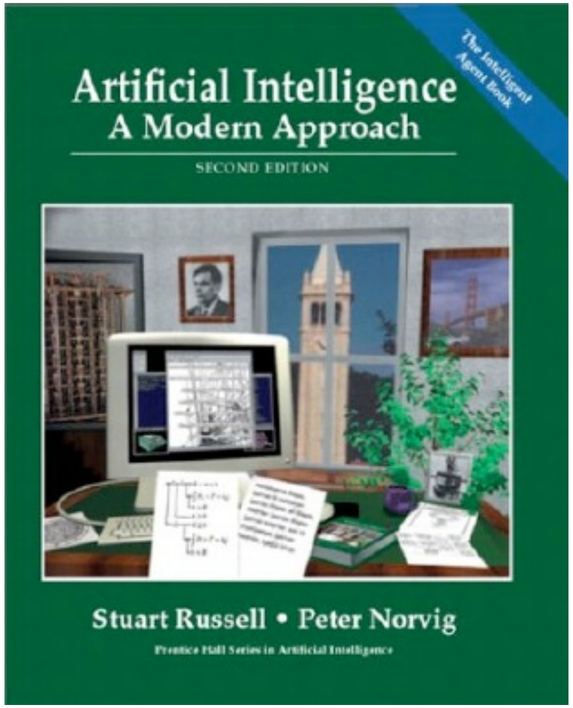
Einführung in Web- und Data-Science

Prof. Dr. Ralf Möller
Universität zu Lübeck
Institut für Informationssysteme

Tanya Braun (Übungen)

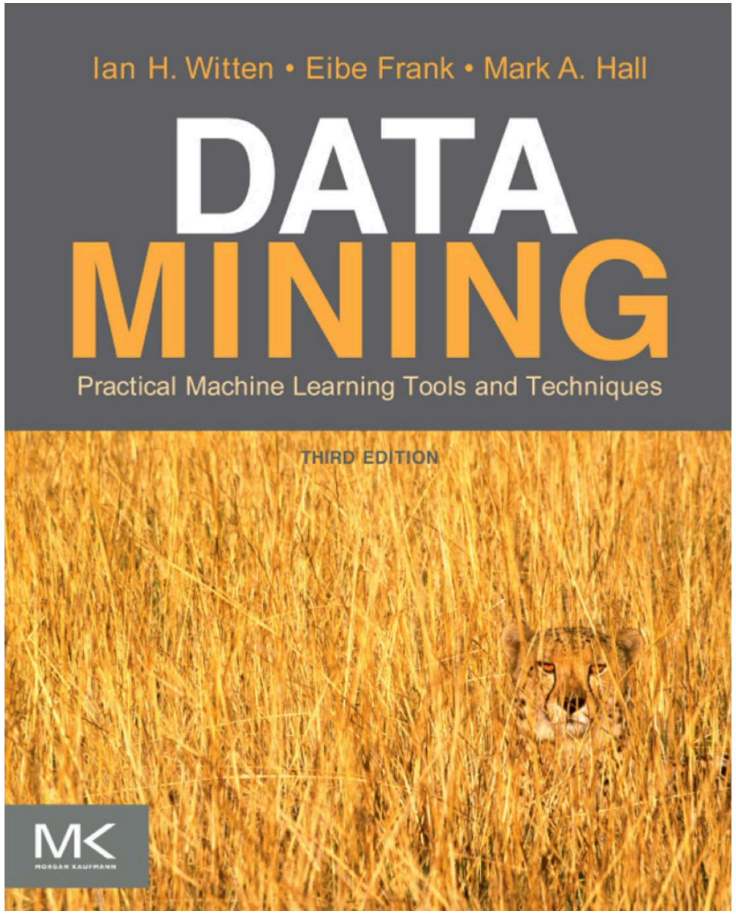
Inductive Learning

Chapter 18/19



Material adopted from
Yun Peng, Chuck Dyer,
Gregory Piatetsky-Shapiro & Gary Parker

Chapters 3 and 4



Card Example: Guess a Concept

- Given a set of examples
  - Positive: e.g., 4♣ 7♣ 2♠
  - Negative: e.g., 5♥ j♠
- What cards are accepted?
  - What concept lays behind it?

Card Example: Guess a Concept

(r=1) v … v (r=10) v (r=J) v (r=Q) v (r=K) ⇔ ANY-RANK(r)

(r=1) v … v (r=10) ⇔ NUM(r)

(r=J) v (r=Q) v (r=K) ⇔ FACE(r)

(s=♠) v (s=♣) v (s=♦) v (s=♥) ⇔ ANY-SUIT(s)

(s=♠) v (s=♣) ⇔ BLACK(s)

(s=♦) v (s=♥) ⇔ RED(s)

A hypothesis is any sentence of the form:

$$R(r) \wedge S(s)$$

where:

- R(r) is ANY-RANK(r), NUM(r), FACE(r), or (r=x)
- S(s) is ANY-SUIT(s), BLACK(s), RED(s), or (s=y)

Simplified Representation

For simplicity, we represent a concept by rs, with:

- r ∈ {a, n, f, 1, …, 10, j, q, k}
- s ∈ {a, b, r, ♣, ♠, ♦, ♥}

For example:

- n♠ represents:
  NUM(r) ∧ (s=♠)
- aa represents:
  ANY-RANK(r) ∧ ANY-SUIT(s)

Extension of a Hypothesis

The extension of a hypothesis h is the set of objects that satisfies h

Examples:

- The extension of f♠ is: {j♠, q♠, k♠}
- The extension of aa is the set of all cards

More General/Specific Relation

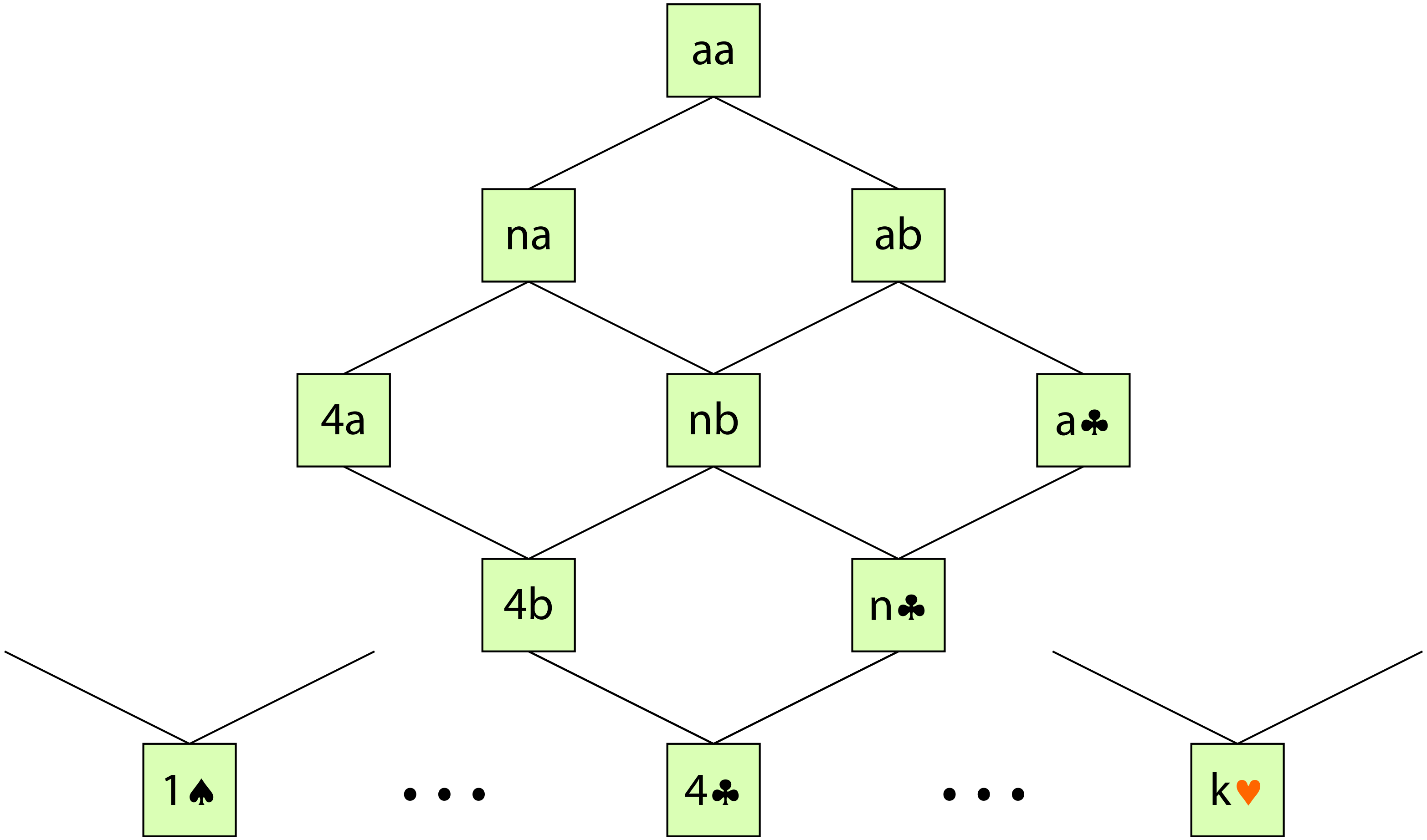
- Let h_1 and h_2 be two hypotheses in H
- h_1 is more general than h_2 iff the extension of h_1 is a proper superset of the extension of h_2

Examples:
- aa is more general than f♦
- f♥ is more general than q♥
- fr and nr are not comparable

More General/Specific Relation

- Let h_1 and h_2 be two hypotheses in H
- h_1 is more general than h_2 iff the extension of h_1 is a proper superset of the extension of h_2
- The inverse of the "more general" relation is the "more specific" relation
- The "more general" relation defines a partial ordering on the hypotheses in H

Example: Subset of Partial Order



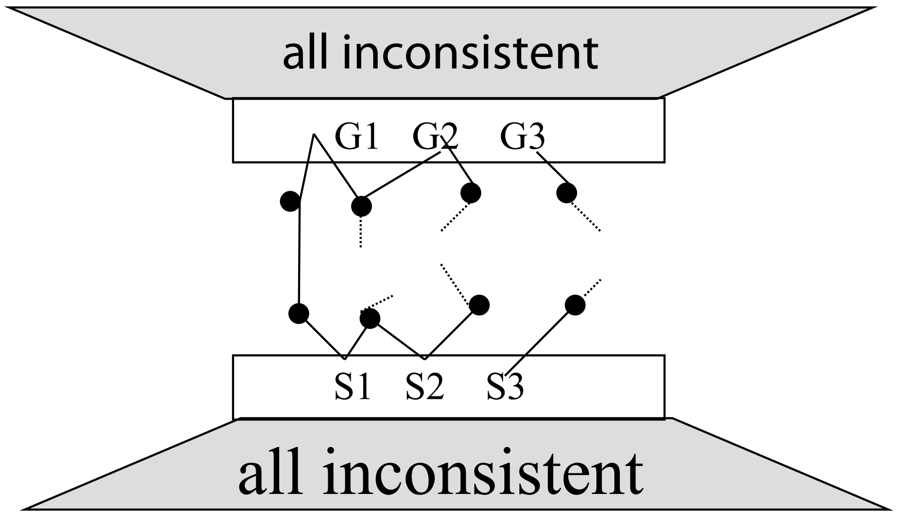
G-Boundary / S-Boundary of V

- A hypothesis in V is most general
  iff no hypothesis in V is more general

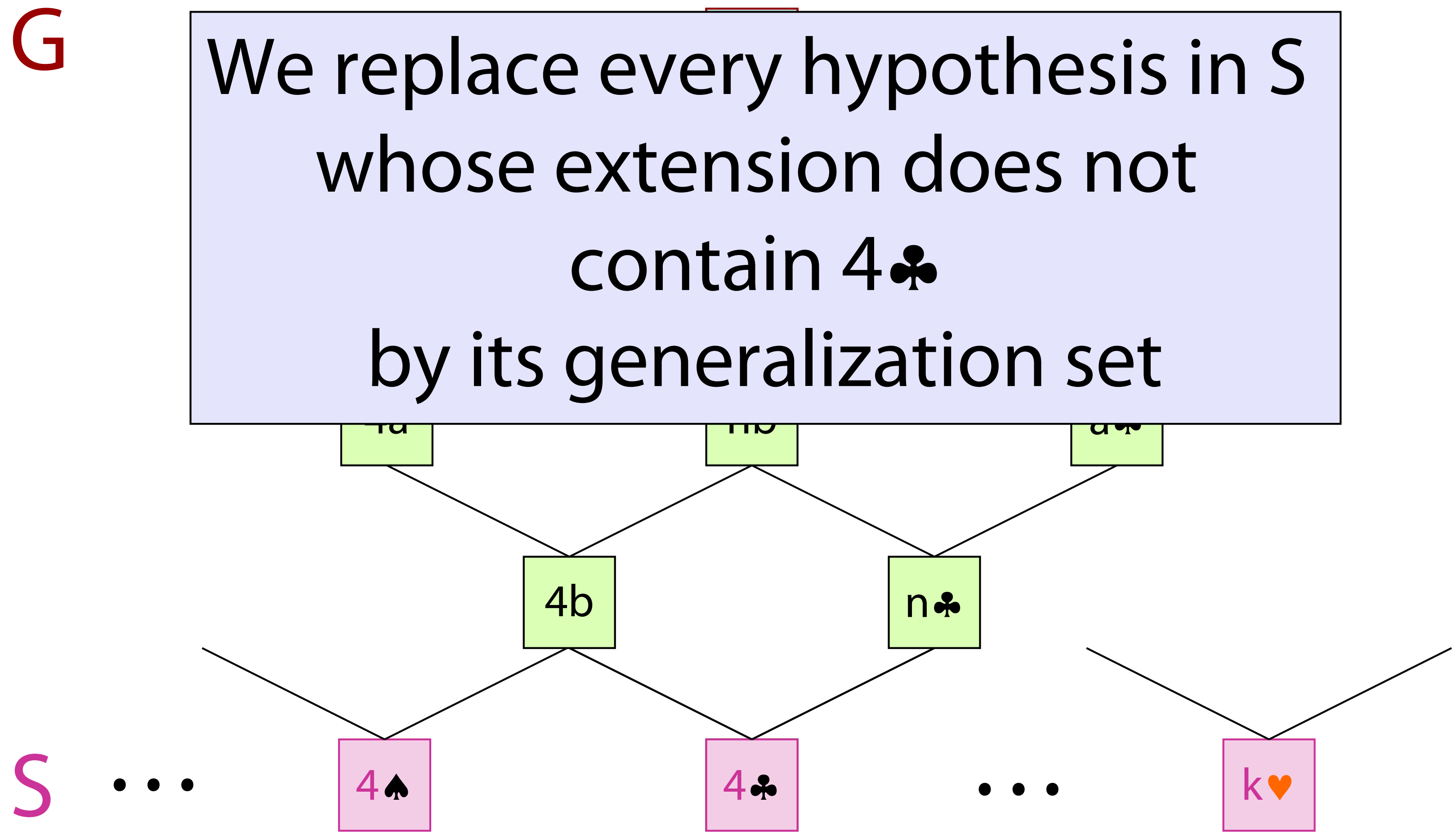
- G-boundary G of V: Set of most general hypotheses in V

G-Boundary / S-Boundary of V

- A hypothesis in V is most general iff no hypothesis in V is more general
- G-boundary G of V: Set of most general hypotheses in V
- A hypothesis in V is most specific iff no hypothesis in V is more specific
- S-boundary S of V: Set of most specific hypotheses in V



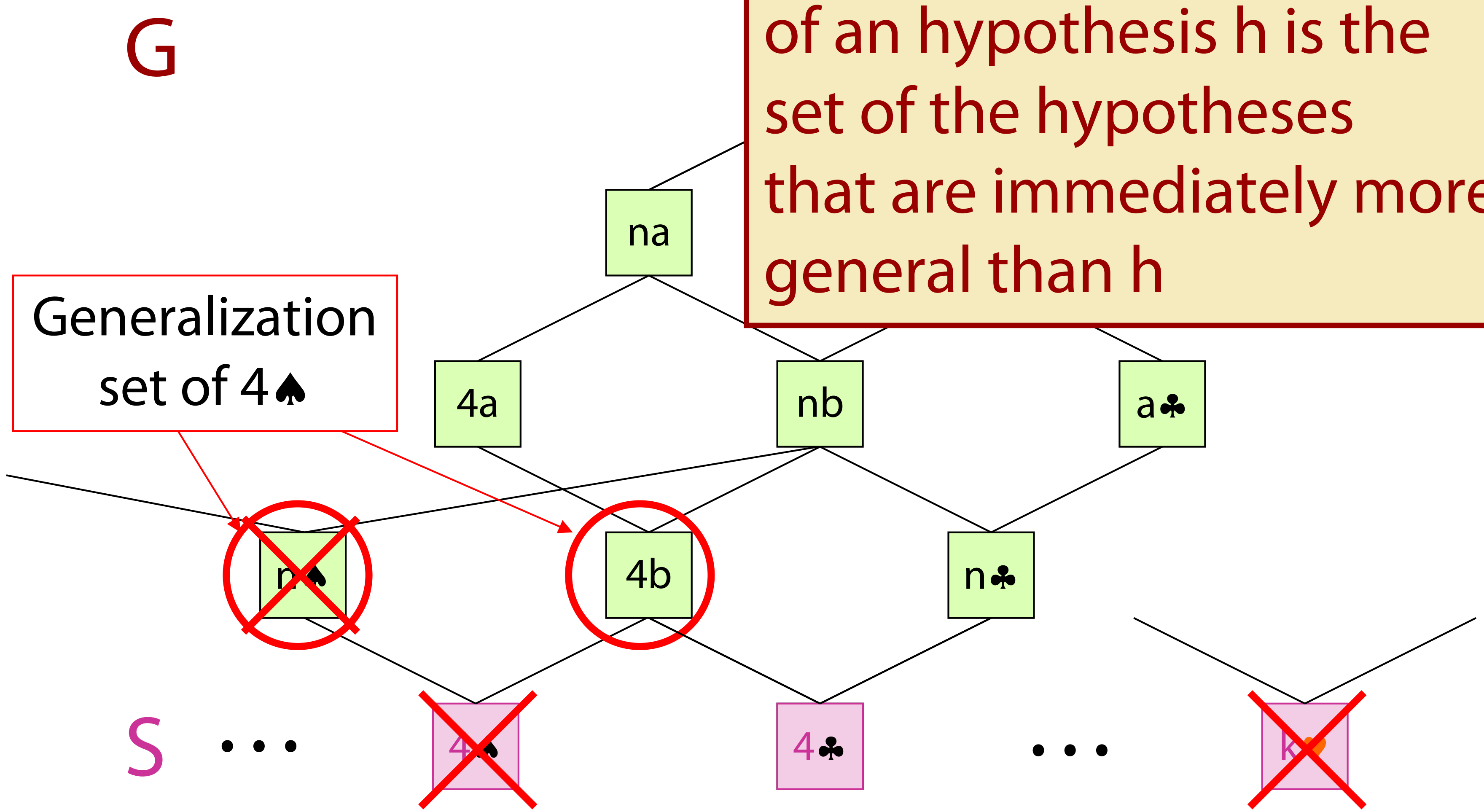
Example: G-/S-Boundaries of V



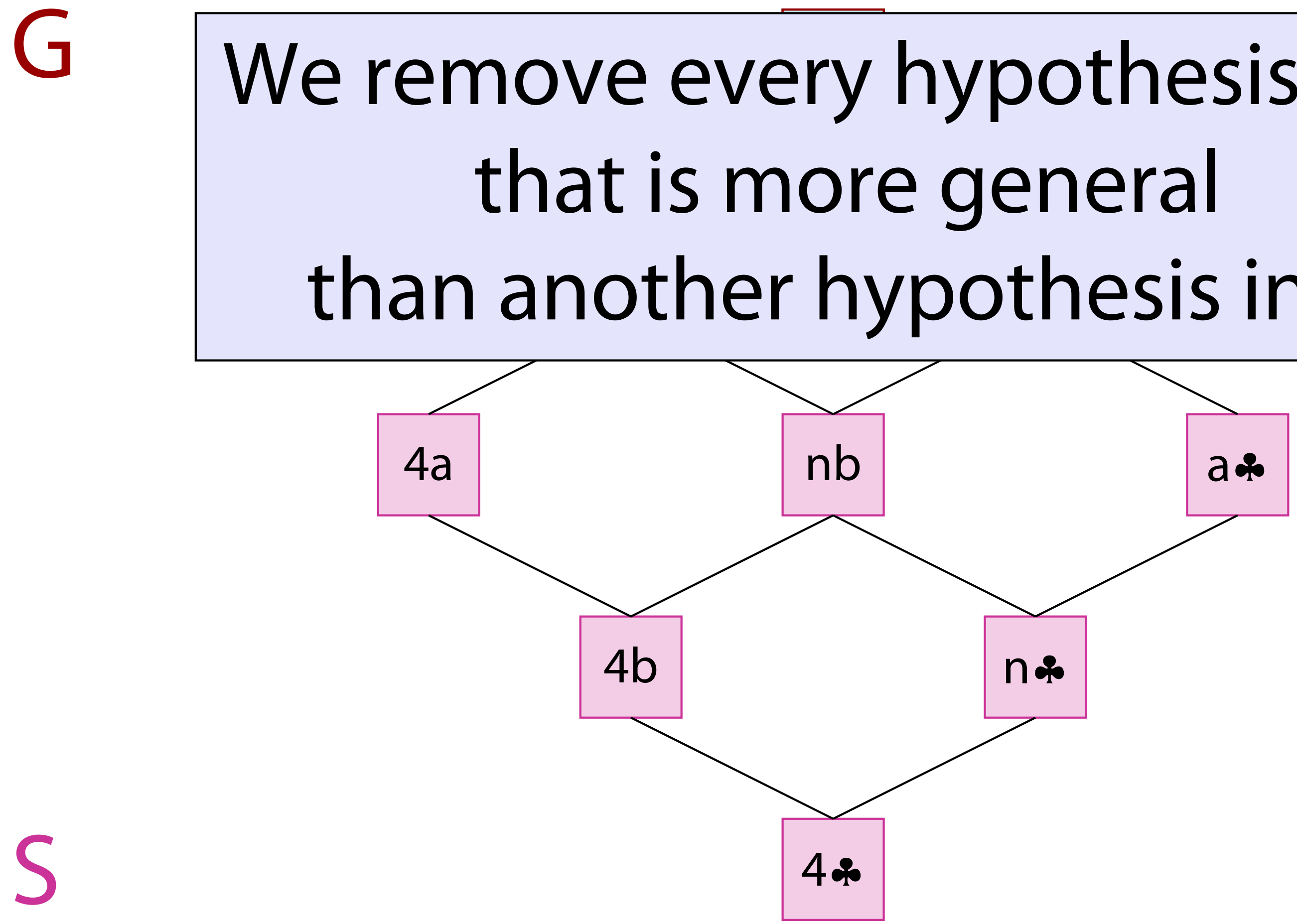
We replace every hypothesis in S whose extension does not contain 4♣ by its generalization set

Example: G-/S-Boundaries of V

The **generalization set** of an hypothesis h is the set of the hypotheses that are immediately more general than h



Example: G-/S-Boundaries of V

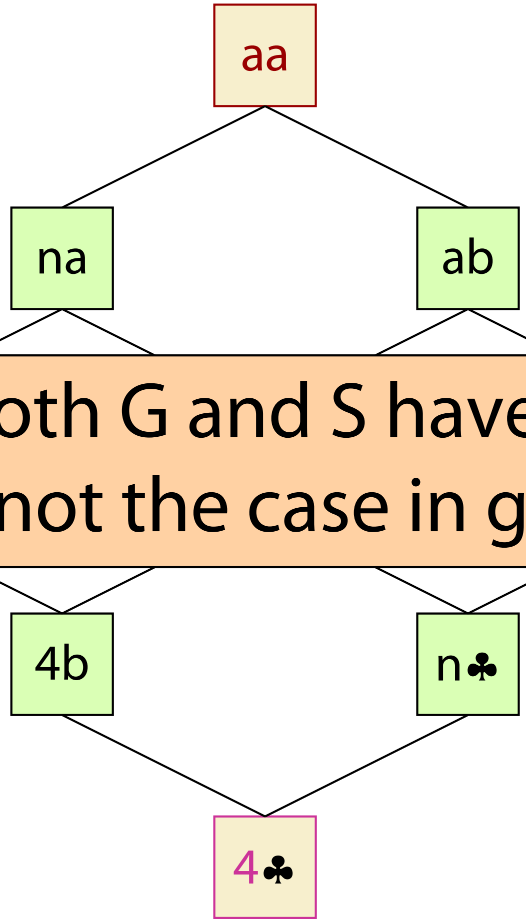


We remove every hypothesis in S
that is more general
than another hypothesis in G

Example: G-/S-Boundaries of V

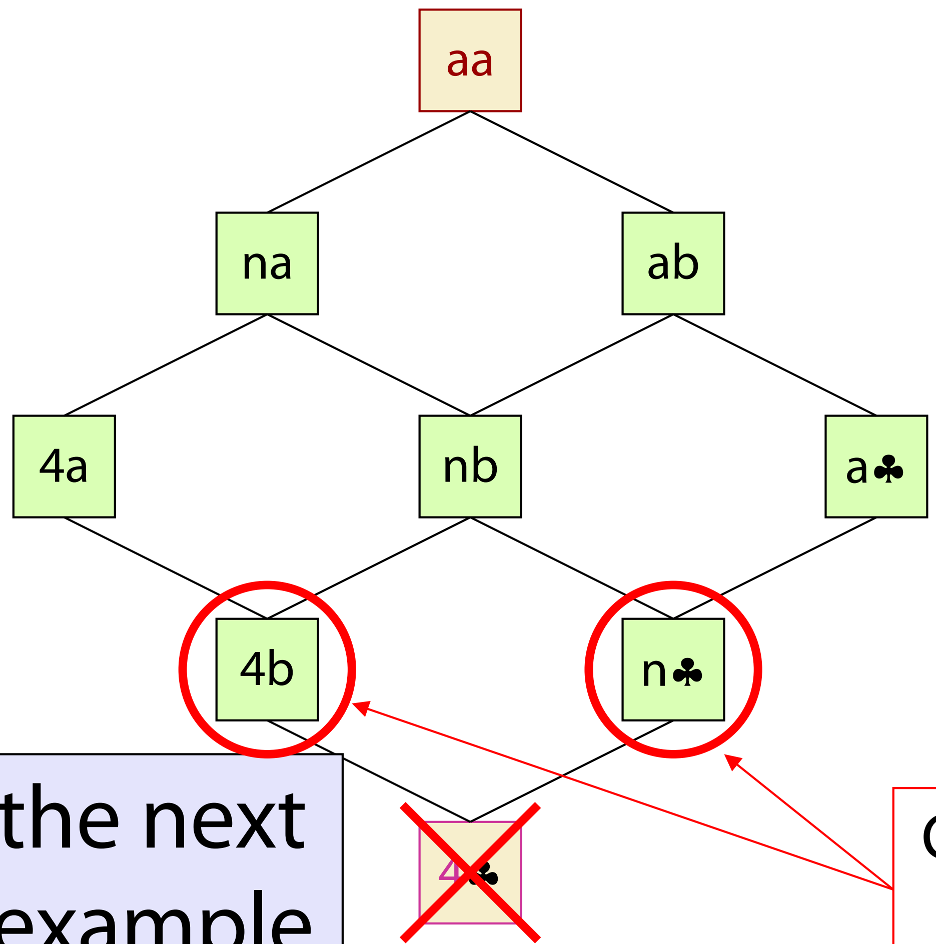
G

S



Here, both G and S have size 1.
This is not the case in general!

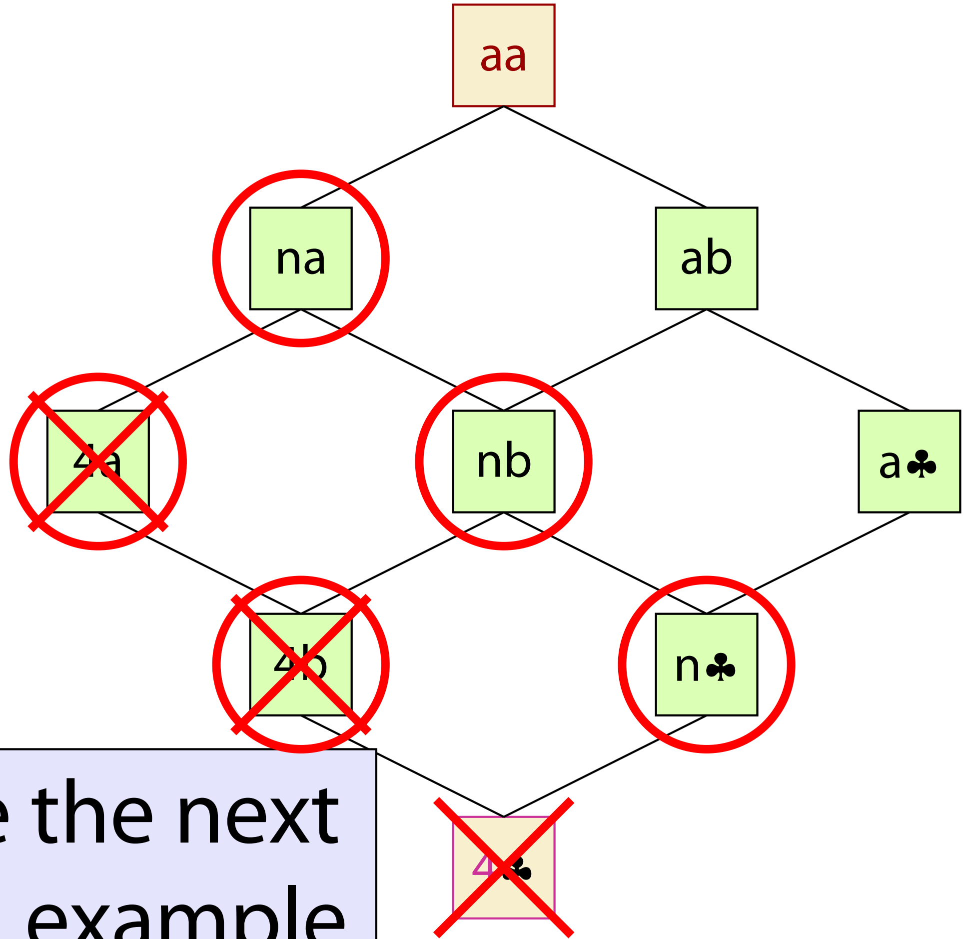
Example: G-/S-Boundaries of V



Let 7♣ be the next (positive) example

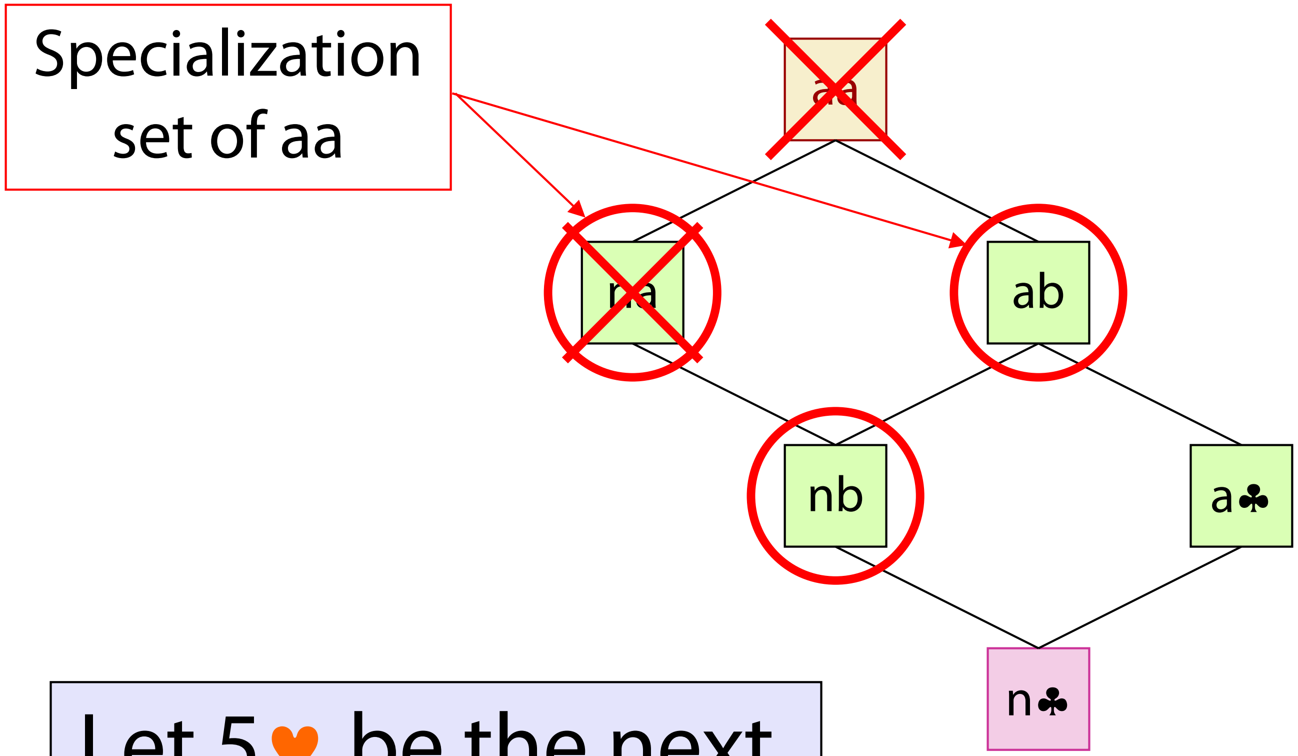
Generalization set of 4♣

Example: G-/S-Boundaries of V



Let 7♣ be the next (positive) example

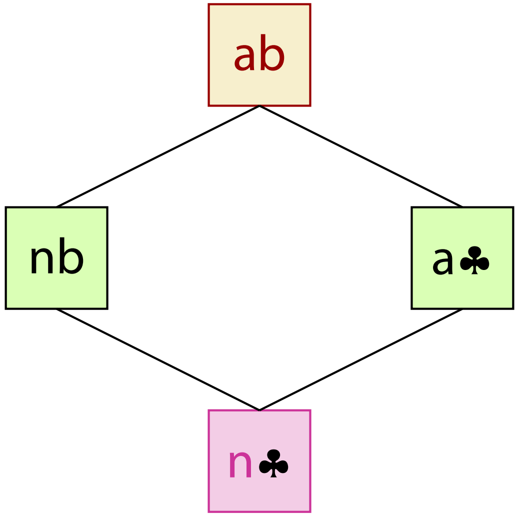
Example: G-/S-Boundaries of V



Let 5♥ be the next (negative) example

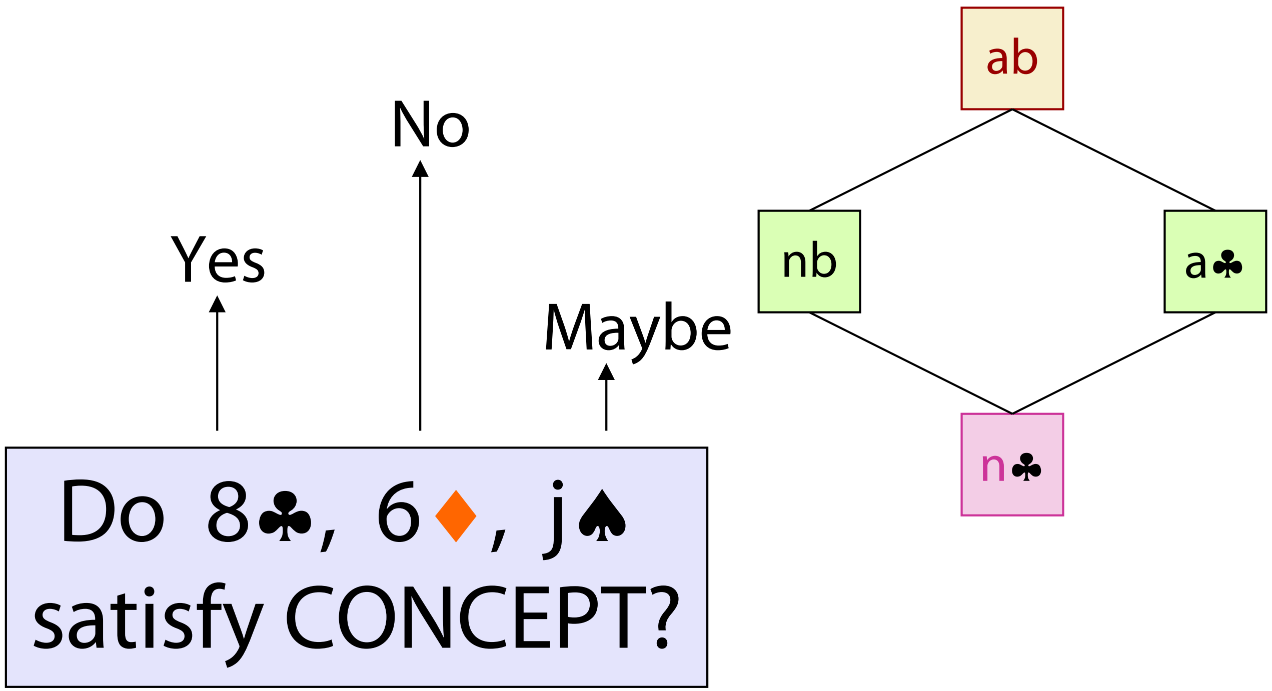
Example: G-/S-Boundaries of V

G and S, and all hypotheses in between form exactly the version space

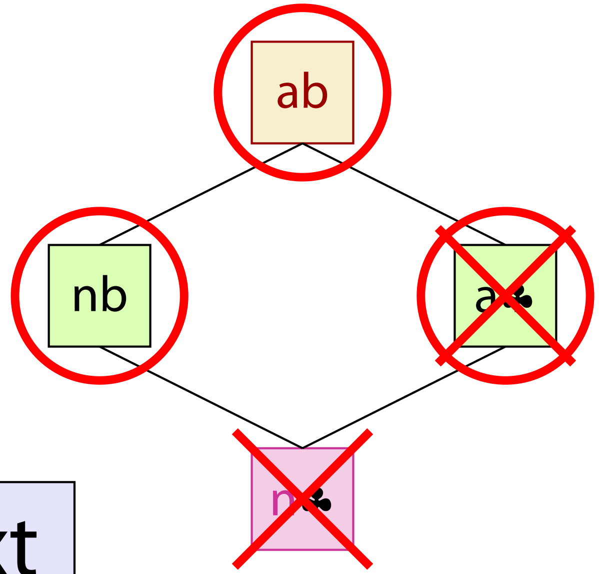


Example: G-/S-Boundaries of V

At this stage …

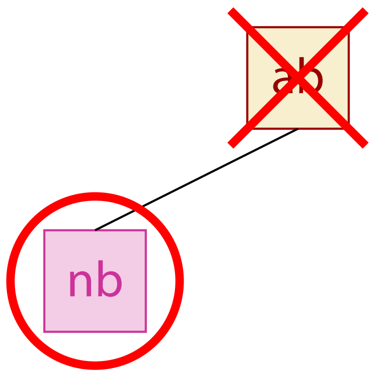


Example: G-/S-Boundaries of V



Let 2♠ be the next (positive) example

Example: G-/S-Boundaries of V



Let j♠ be the next (negative) example

Example: G-/S-Boundaries of V

+ 4♣ 7♣ 2♠

– 5♥ j♠

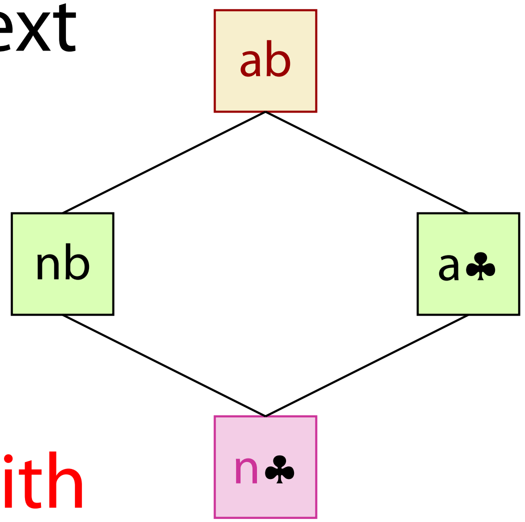
nb

NUM(r) ∧ BLACK(s)

Example: G-/S-Boundaries of V

Let us return to the version space …

… and let 8♣ be the next (negative) example

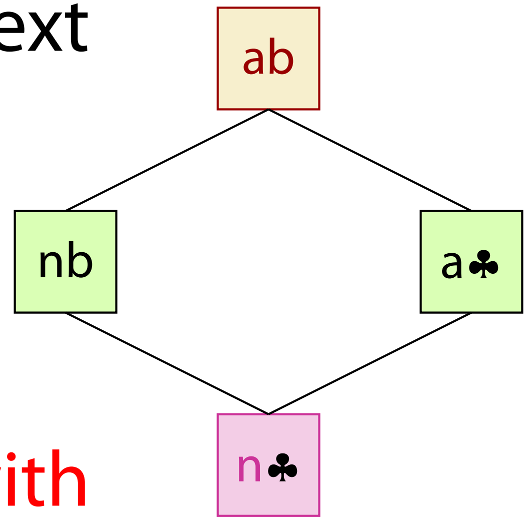


The only most specific hypothesis disagrees with this example, so no hypothesis in H agrees with all examples

Example: G-/S-Boundaries of V

Let us return to the version space …

… and let j♥ be the next (positive) example



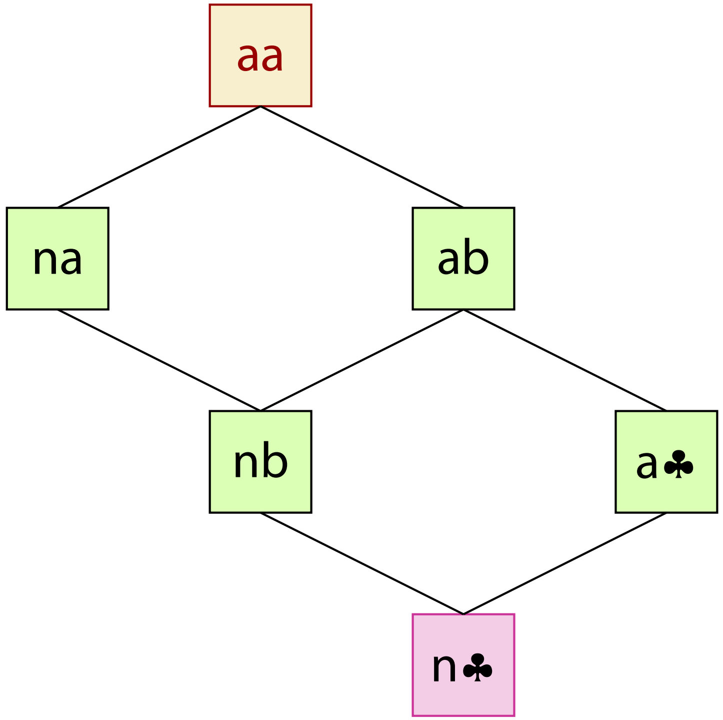
The only most general hypothesis disagrees with this example, so no hypothesis in H agrees with all examples

Example-Selection Strategy

- Suppose that at each step the learning procedure has the possibility to select the object (card) of the next example
- Let it pick the object such that, whether the example is positive or not, it will eliminate one-half of the remaining hypotheses
- Then a single hypothesis will be isolated in $O(\log |H|)$ steps

Example

- 9♣?
- j♥?
- j♣?



Example-Selection Strategy

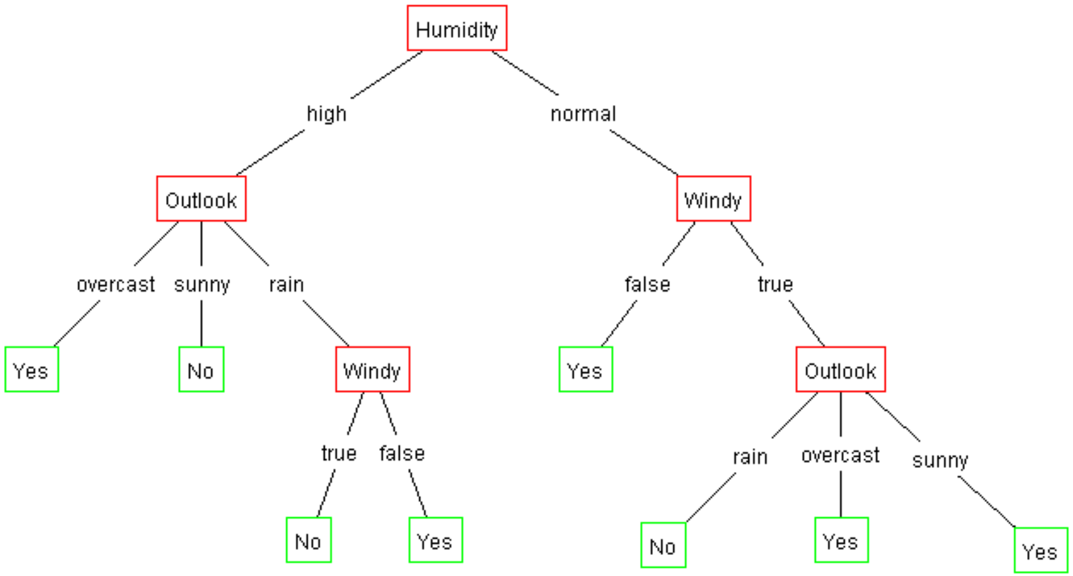
- Suppose that at each step the learning procedure has the possibility to select the object (card) of the next example
- Let it pick the object such that, whether the example is positive or not, it will eliminate one-half of the remaining hypotheses
- Then a single hypothesis will be isolated in O(log |H|) steps
- But picking the object that eliminates half the version space may be expensive

Noise

- If some examples are misclassified, the version space may collapse
- Possible solution:
  Maintain several G- and S-boundaries, e.g., consistent with all examples, all examples but one, etc…

Decision Trees

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



Decision trees

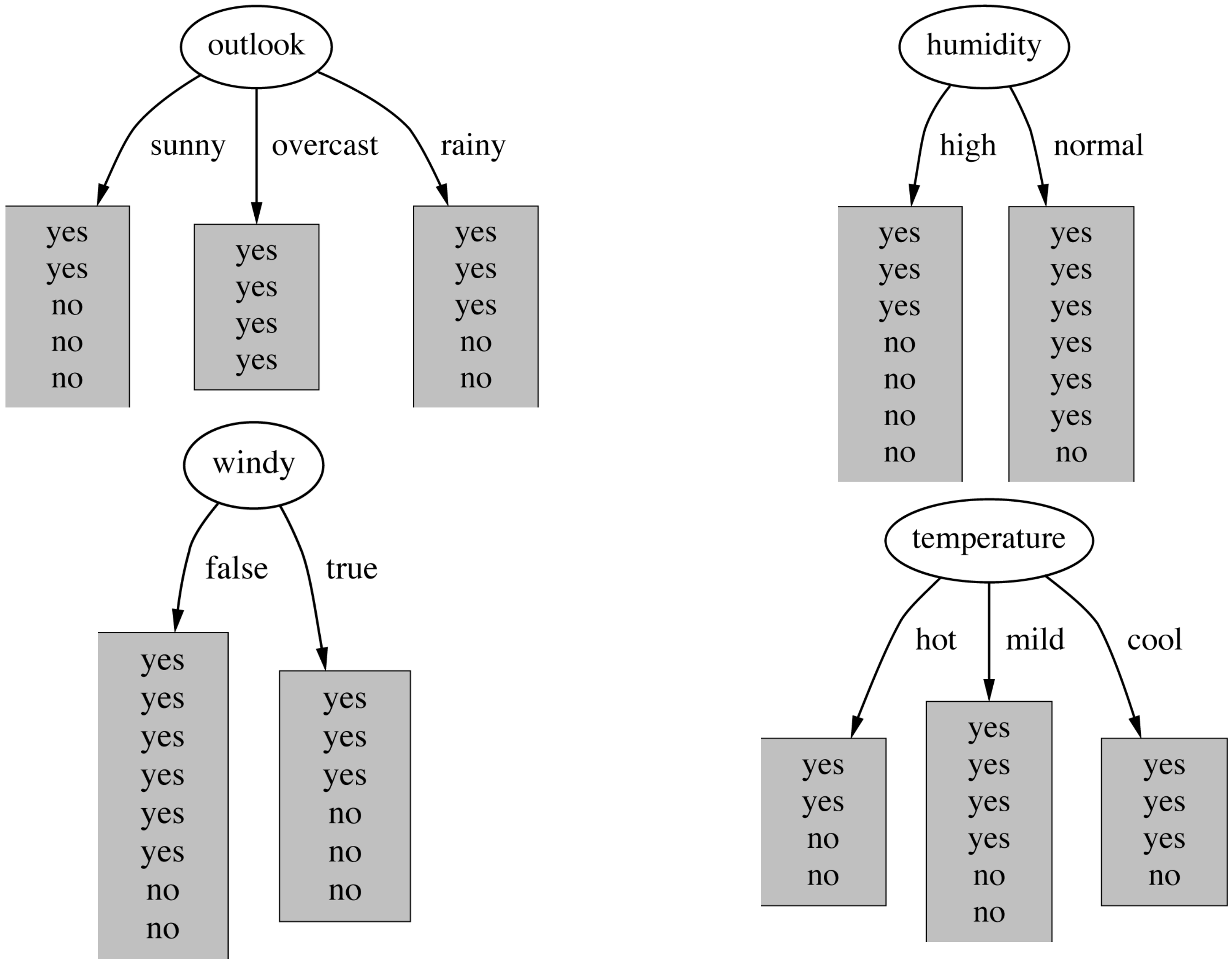
- An internal node is a test on an attribute.
- A branch represents an outcome of the test, e.g., Color=red.
- A leaf node represents a class label or class label distribution.
- At each node, one attribute is chosen to split training examples into distinct classes as much as possible
- A new case is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

Building Decision Trees

- 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.

R. Quinlan, Learning efficient classification procedures, Machine Learning: an artificial intelligence approach, Michalski, Carbonell & Mitchell (eds.), Morgan Kaufmann, p. 463-482., **1983**

Which attribute to select?



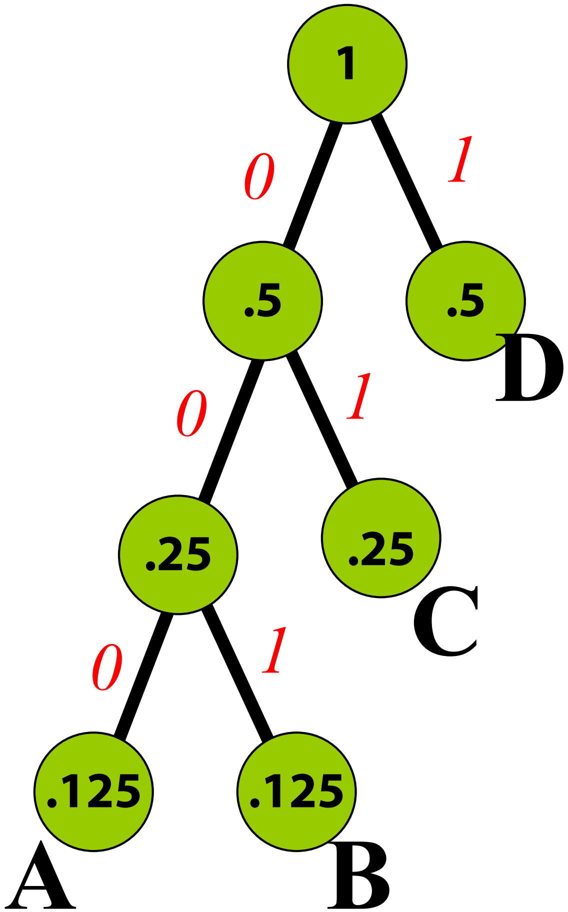
Choosing the Best Attribute

- The key problem is choosing which attribute to split a given set of examples.
- Some possibilities are:
  - Random: Select any attribute at random
  - Least-Values: Choose the attribute with the smallest number of possible values
  - Most-Values: Choose the attribute with the largest number of possible values
  - Information gain: Choose the attribute that has the largest expected information gain, i.e. select attribute that will result in the smallest expected size of the subtrees rooted at its children.

Information Theory

- Assume you can bet 1$ for a coin flip (10000 bets), if your bet is right, you get back 2$ otherwise you get nothing
- You know that the coin used is rigged and comes up heads with probability 0.99, so you bet heads - obviously (but find somebody arranging this bet :)
- The expected value for the bet is 1.98$
- How much will you be willing to pay for the advance information about the actual outcome of the flip? What the value of the advance information?
- Less than 0.02$!
- If the coin were fair, your expected value would 1$ and you would be willing to pay up to 1$
- The less you know, the more valuable the information
- Information theory does not measure the value of information in $ but the information content of a message in bits.

Huffman code example



M	code	length	prob	
A	000	3	0,125	0,375
B	001	3	0,125	0,375
C	01	2	0,250	0,500
D	1	1	0,500	0,500
average message length				1,750

If we need to send many messages (A,B,C or D) and they have this probability distribution and we use this code, then over time, the average bits/message should approach 1.75

Information Theory Background

- If there are n equally probable possible messages, then the probability p of each is 1/n
- Information conveyed by a message is log(n) = -log(p)
- Eg, if there are 16 messages, then log(16) = 4 and we need 4 bits to identify/send each message.
- In general, if we are given a probability distribution

  P = (p1, p2, .., pn)
- the information conveyed by distribution (aka entropy of P) is:

  I(P) = -(p1*log(p1) + p2*log(p2) + .. + pn*log(pn))

  $= - \sum_i pi*log(pi)$

Information Theory Background

- Information conveyed by distribution (aka entropy of P) is:

  I(P) = -(p1*log(p1) + p2*log(p2) + .. + pn*log(pn))
- Examples:
  - if P is (0.5, 0.5) then I(P) is 1
  - if P is (0.67, 0.33) then I(P) is 0.92,
  - if P is (1, 0) or (0,1) then I(P) is 0.
- The more uniform is the probability distribution, the greater is its information.
- The entropy is the average number of bits/message needed to represent a stream of messages.

Example: attribute “Outlook”, 1

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

Example: attribute "Outlook", 2

- "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)

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

Computing the information gain

- Information gain:

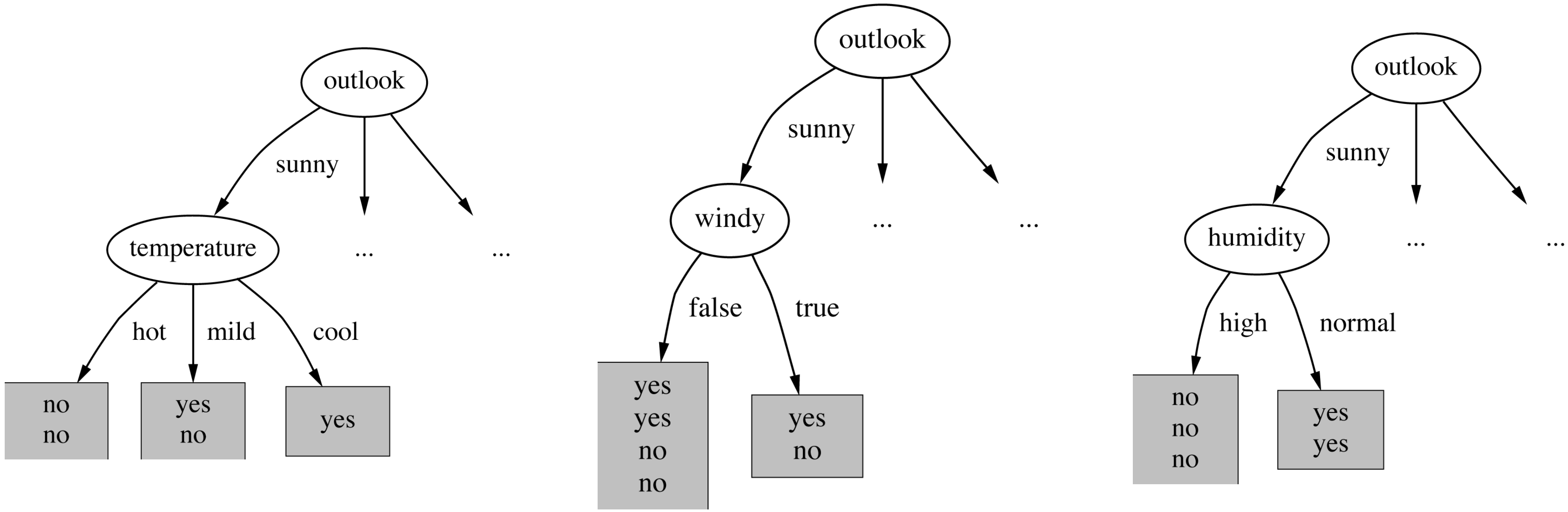
(information before split) – (information after split)

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

- Information gain for attributes from weather data:

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

Continuing to split

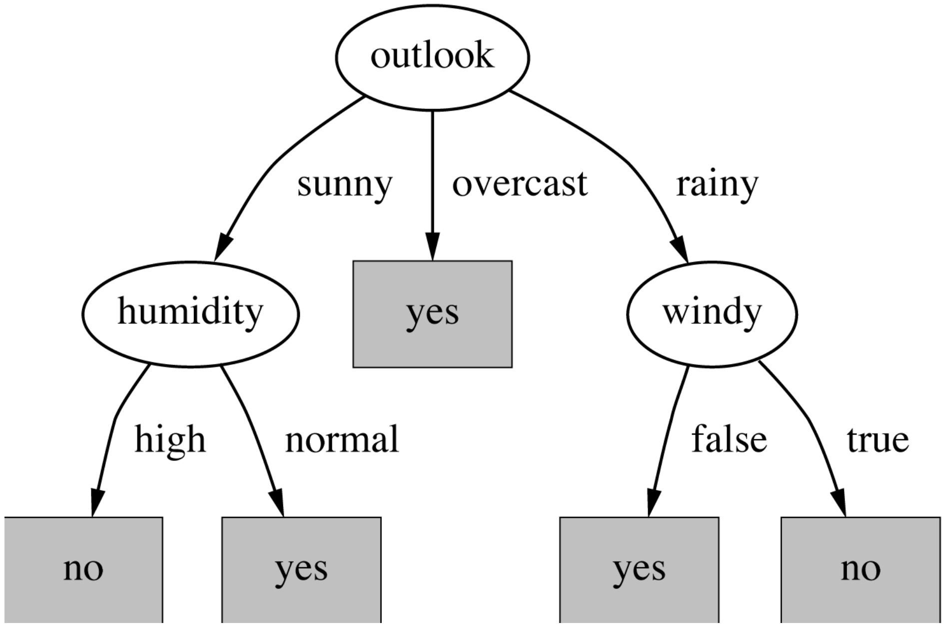


$$gain("Humidity") = 0.971 \text{ bits}$$

$$gain("Temperature") = 0.571 \text{ bits}$$

$$gain("Windy") = 0.020 \text{ bits}$$

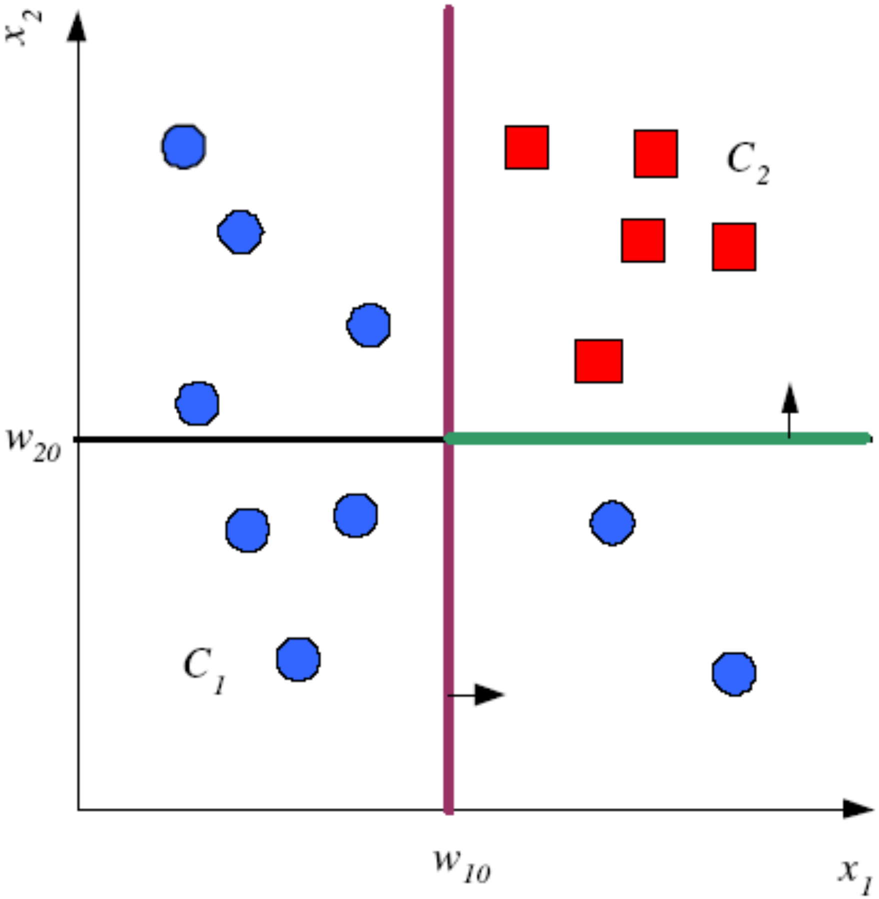
The final decision tree

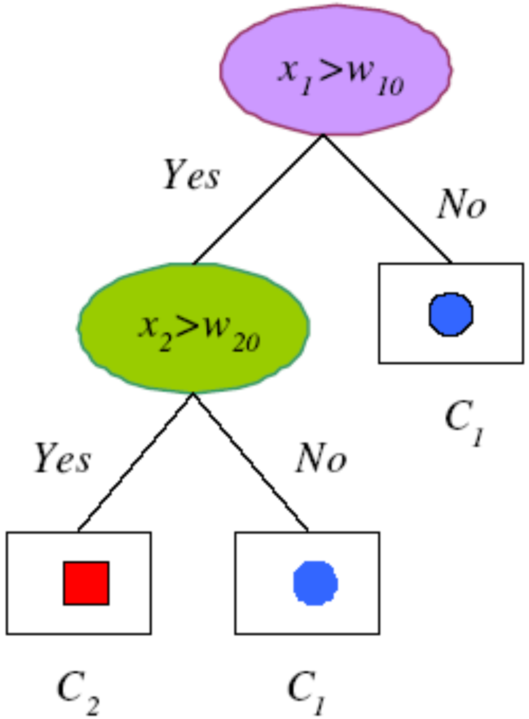


- Note: not all leaves need to be pure; sometimes identical instances have different classes

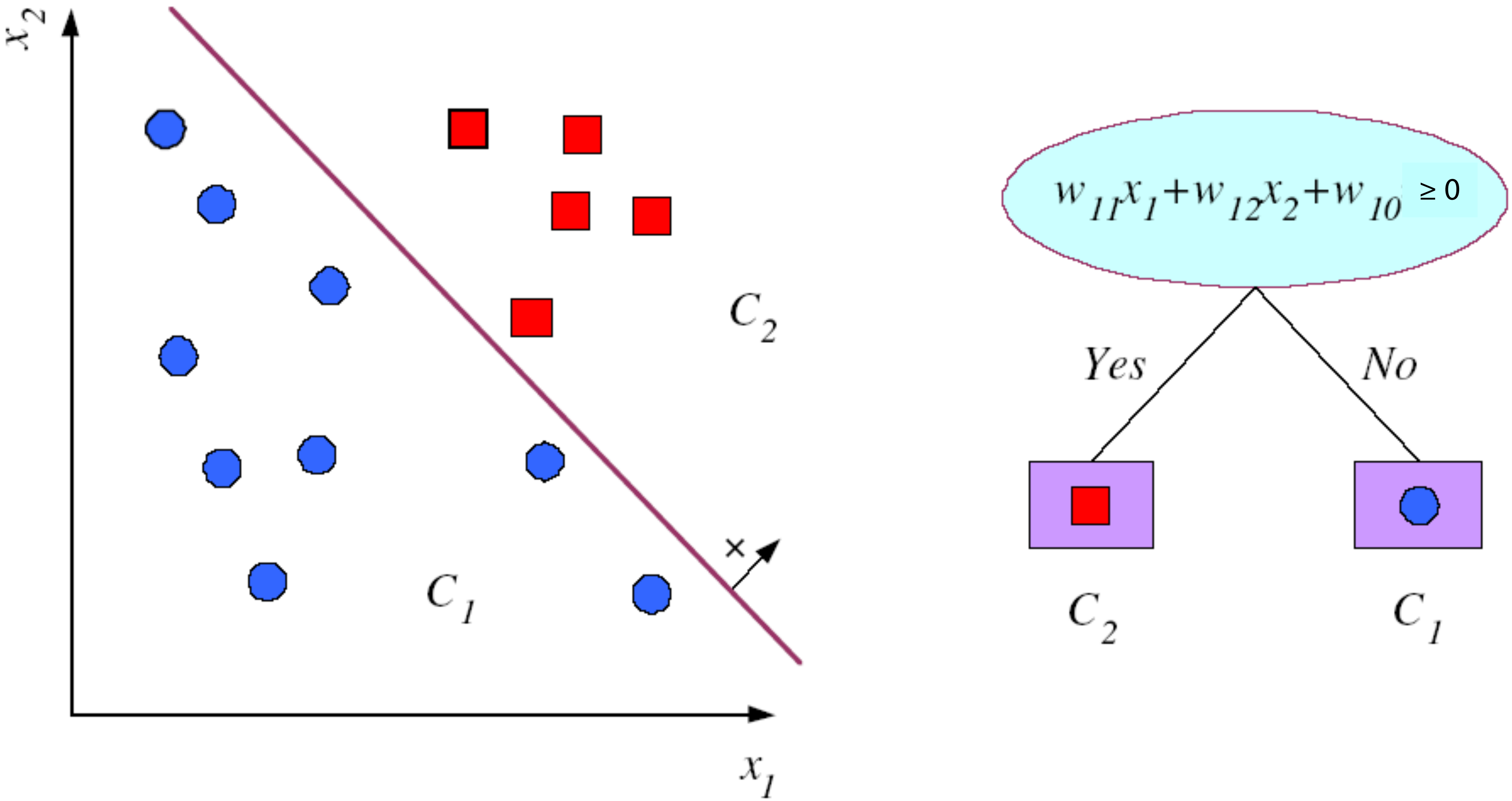
  $\Rightarrow$ Splitting stops when data can't be split any further

Univariate Splits





Multivariate Splits



1R – Simplicity First!

Given: Table with data

Goal: Learn decision function

- Based on rules that all test one particular attribute
- One branch for each value
- Each branch assigns most frequent class
- Error rate: proportion of instances that don't belong to the majority class of their corresponding branch
- Choose attribute with lowest error rate

(*Assumes nominal attributes*)

Classification

Outlook	Temp	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

Evaluating the Weather Attributes

Classification

Outlook	Temp	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

Attribute	Rules	Errors	Total errors
Outlook	Sunny $\rightarrow$ No	2/5	4/14
	Overcast $\rightarrow$ Yes	0/4	
	Rainy $\rightarrow$ Yes	2/5	
Temp	Hot $\rightarrow$ No*	2/4	5/14
	Mild $\rightarrow$ Yes	2/6	
	Cool $\rightarrow$ Yes	1/4	
Humidity	High $\rightarrow$ No	3/7	4/14
	Normal $\rightarrow$ Yes	1/7	
Windy	False $\rightarrow$ Yes	2/8	5/14
	True $\rightarrow$ No*	3/6	

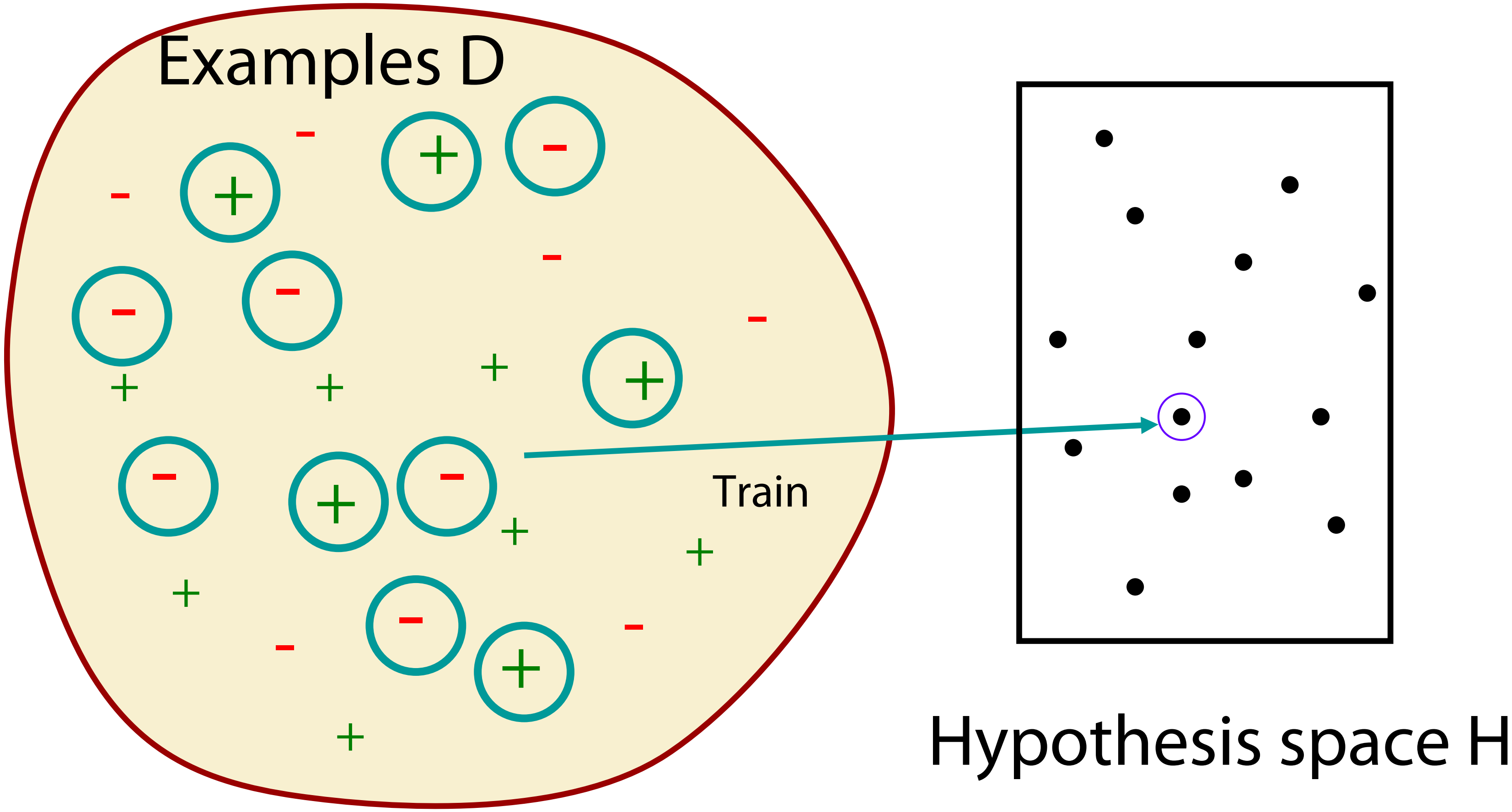
* indicates a tie

Assessing Performance of a Learning Algorithm

- Take out some of the training set
  - Train on the remaining training set
  - Test on the excluded instances
  - *Cross-validation*

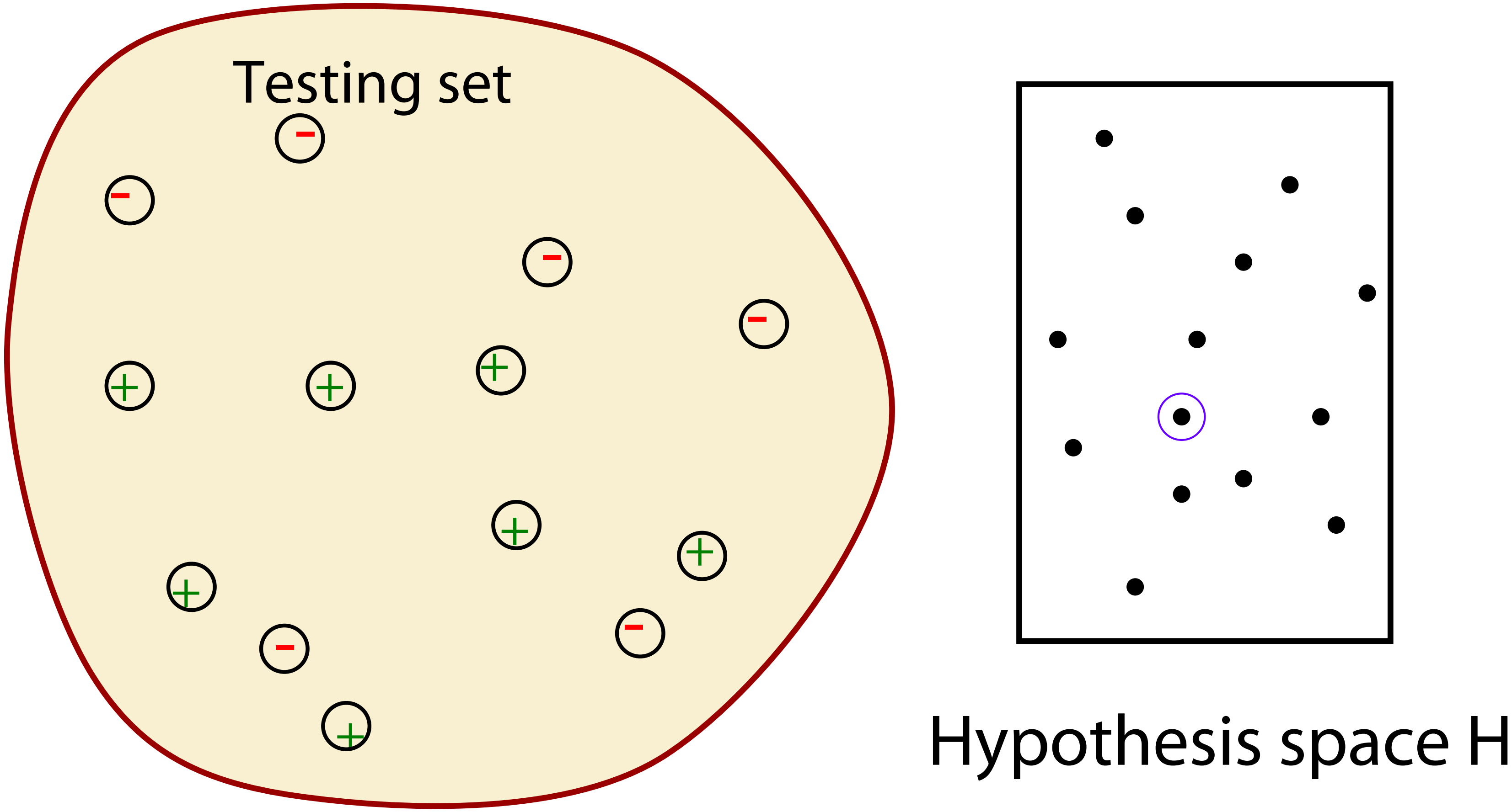
Cross-Validation

- Split original set of examples, train



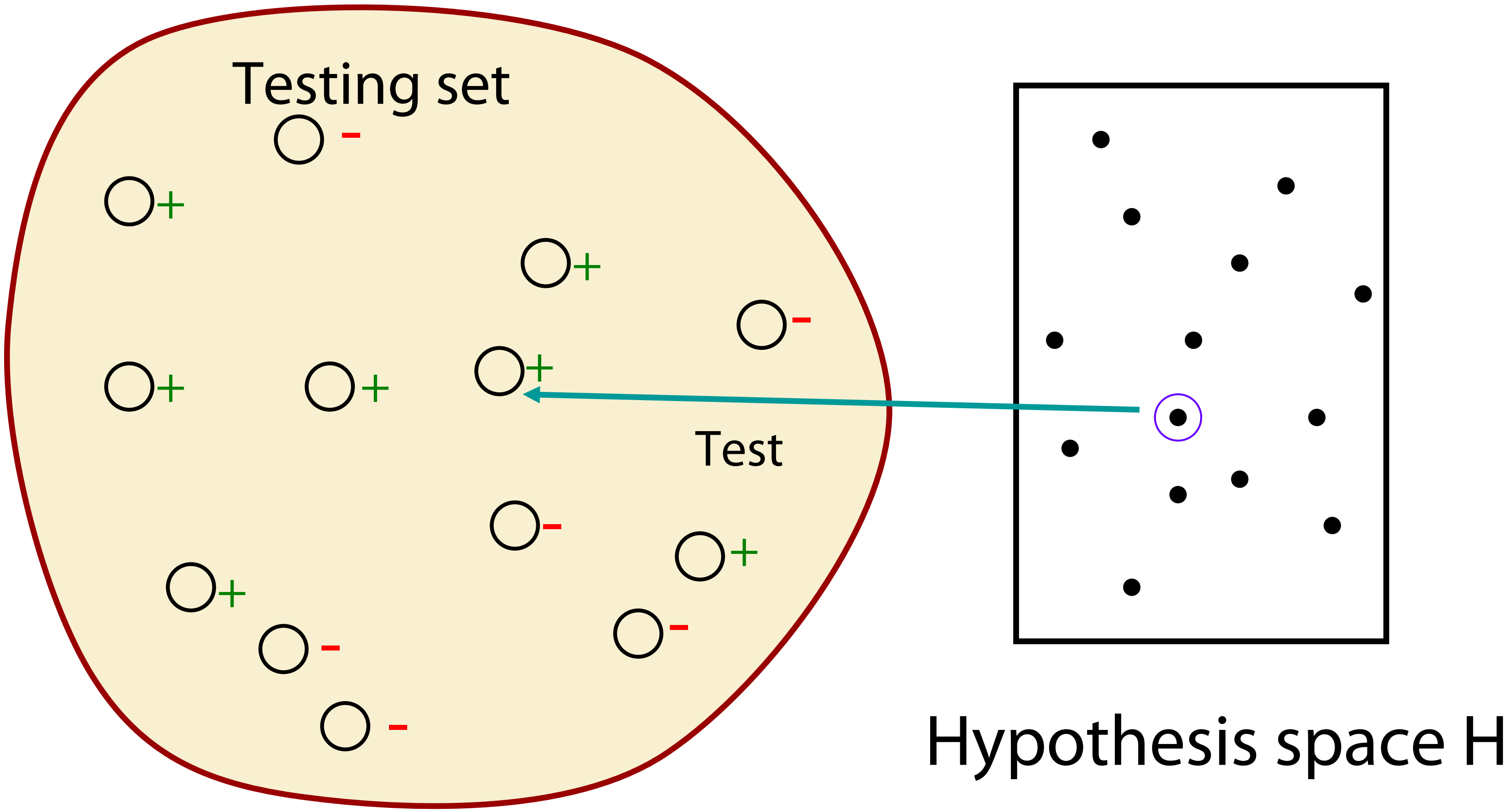
Cross-Validation

- Evaluate hypothesis on testing set



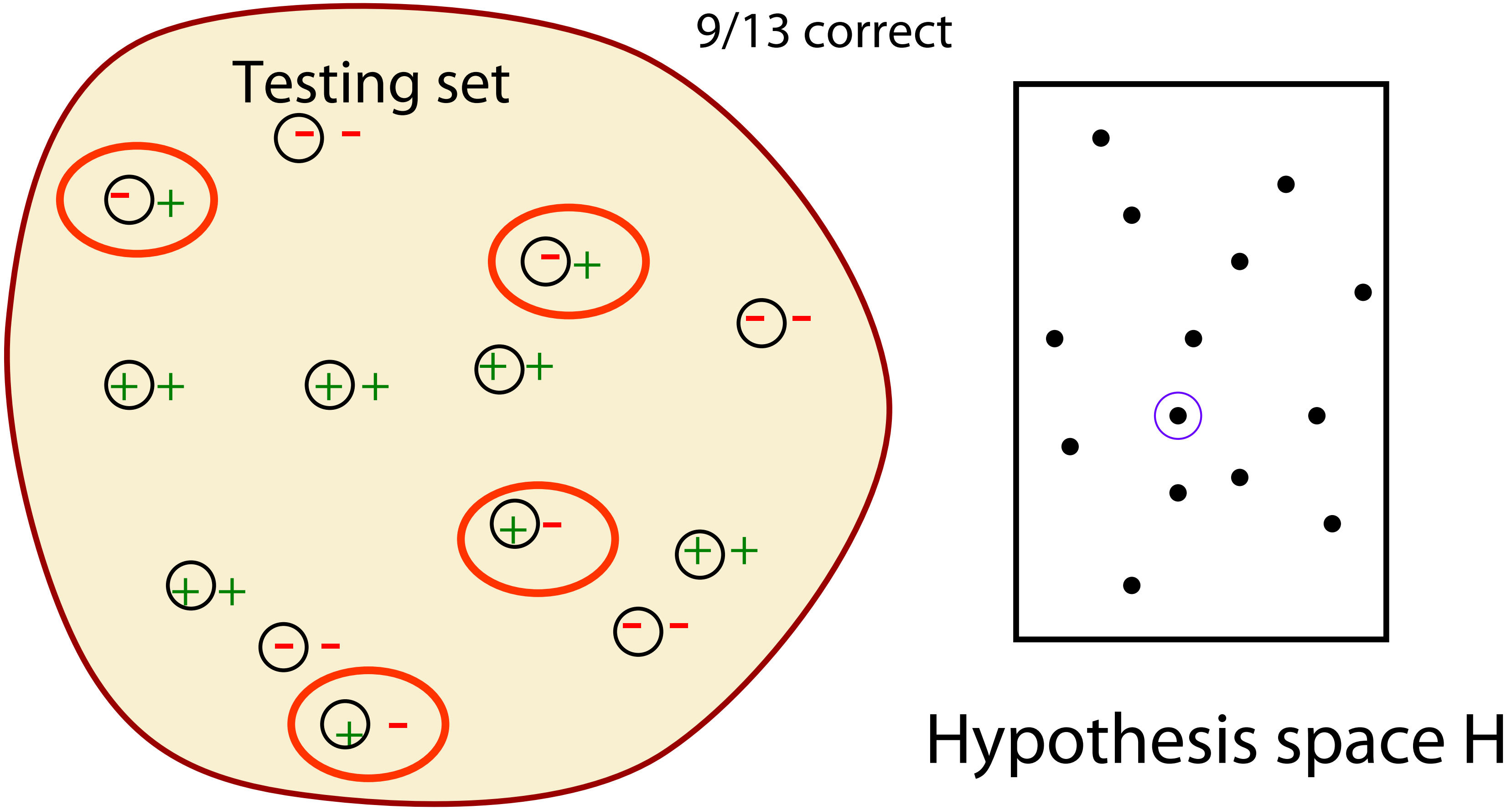
Cross-Validation

- Evaluate hypothesis on testing set



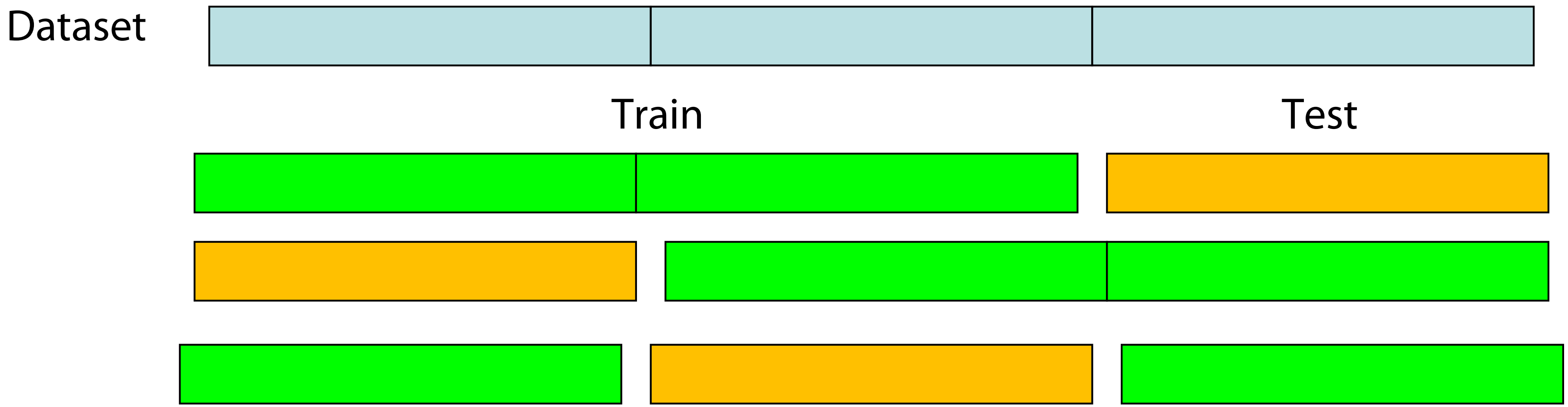
Cross-Validation

- Compare true concept against prediction



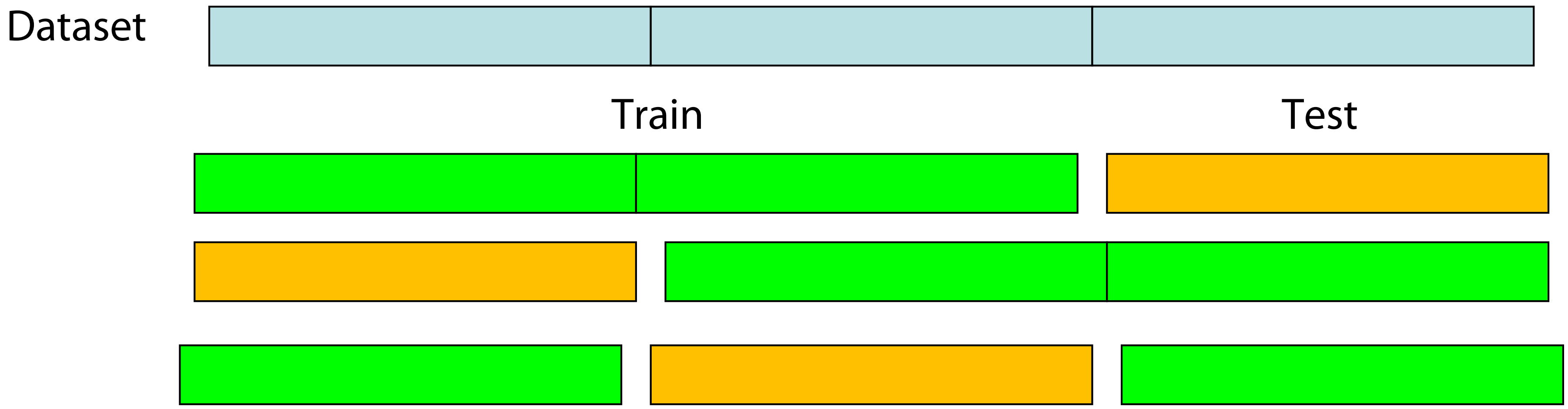
Common Splitting Strategies

- k-fold cross-validation: k random partitions

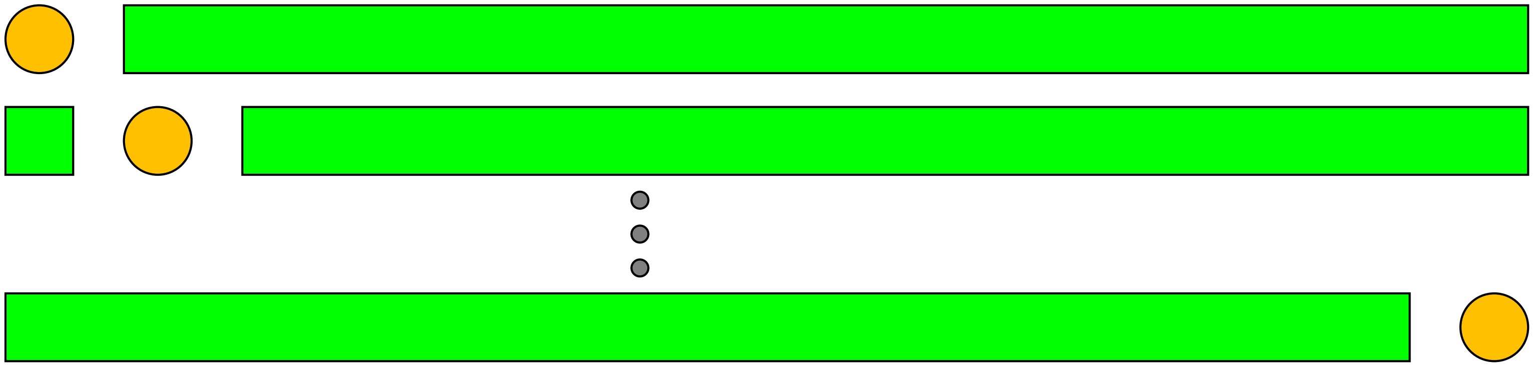


Common Splitting Strategies

- k-fold cross-validation: k random partitions



- Leave-p-out: all possible combinations of p instances



Discussion of 1R

- 1R was described in a paper by Holte (1993)
  - Contains an experimental evaluation on 16 datasets (using *cross-validation* so that results were representative of performance on future data)
  - Minimum number of instances was set to 6 after some experimentation
  - 1R's simple rules performed not much worse than much more complex classifiers
- Simplicity first pays off!

Robert C. Holte, Very Simple Classification Rules Perform Well on Most Commonly Used Datasets, Journal Machine Learning Volume 11, Issue 1 , pp 63-90, **1993**

From ID3 to C4.5: History

- ID3 (Quinlan) – 1960s
- CHAID (CHi-squared Automatic Interaction Detector) – 1960s
- CART (Classification And Regression Tree)
  - Uses another split heuristics (Gini impurity measure)
- C4.5 innovations (Quinlan):
  - Permit numeric attributes
  - Deal 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)

Dealing with Numeric (Metric) Attributes

Outlook	Temperature	Humidity	Windy	Play
Sunny	85.3	85	False	No
Sunny	80.2	90	True	No
Overcast	83.8	86	False	Yes
Rainy	75.2	80	False	Yes
…	…	…	…	…

64 65 68 69 70 71 72 72 75 75 80 81 83 85

- Discretize numeric attributes
- Divide each attribute's range into intervals
  - Sort instances according to attribute's values
  - Place breakpoints where the class changes
    This minimizes the total error

The problem of Overfitting

- This procedure is very sensitive to noise
  - One instance with an incorrect class label will probably produce a separate interval
- Also: *time stamp* attribute will have zero errors
- Simple solution:
  enforce minimum number of instances in majority class per interval

Discretization Example

- Example (with min = 3):

```
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
```

Same decision for both intervals

- Final result for 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
```

With Overfitting Avoidance

- Resulting rule set:

Attribute	Rules	Errors	Total errors
Outlook	Sunny $\rightarrow$ No	2/5	4/14
	Overcast $\rightarrow$ Yes	0/4	
	Rainy $\rightarrow$ Yes	2/5	
Temperature	$\leq$ 77.5 $\rightarrow$ Yes	3/10	5/14
	> 77.5 $\rightarrow$ No*	2/4	
Humidity	$\leq$ 82.5 $\rightarrow$ Yes	1/7	3/14
	> 82.5 and $\leq$ 95.5 $\rightarrow$ No	2/6	
	> 95.5 $\rightarrow$ Yes	0/1	
Windy	False $\rightarrow$ Yes	2/8	5/14
	True $\rightarrow$ No*	3/6	

Numeric Attributes – Advanced

- 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 ≥ 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!

Missing as a Separate Value

- Missing value denoted "?" in C4.X (Null value)
- Simple idea: treat missing as a separate value
- Q: When is this 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:

    $$\begin{aligned} \text{Gain}(X) &= \text{probability that A is known} * (\text{info}(T) - \text{info}_X(T)) + \\ &\quad \text{probability that A is unknown} * 0 \\ &= F * (\text{info}(T) - \text{info}_X(T)) \end{aligned}$$
- Consider instances w/o missing values
- Split w.r.t. those instances
- Distribute instances with missing values proportionally

Pruning

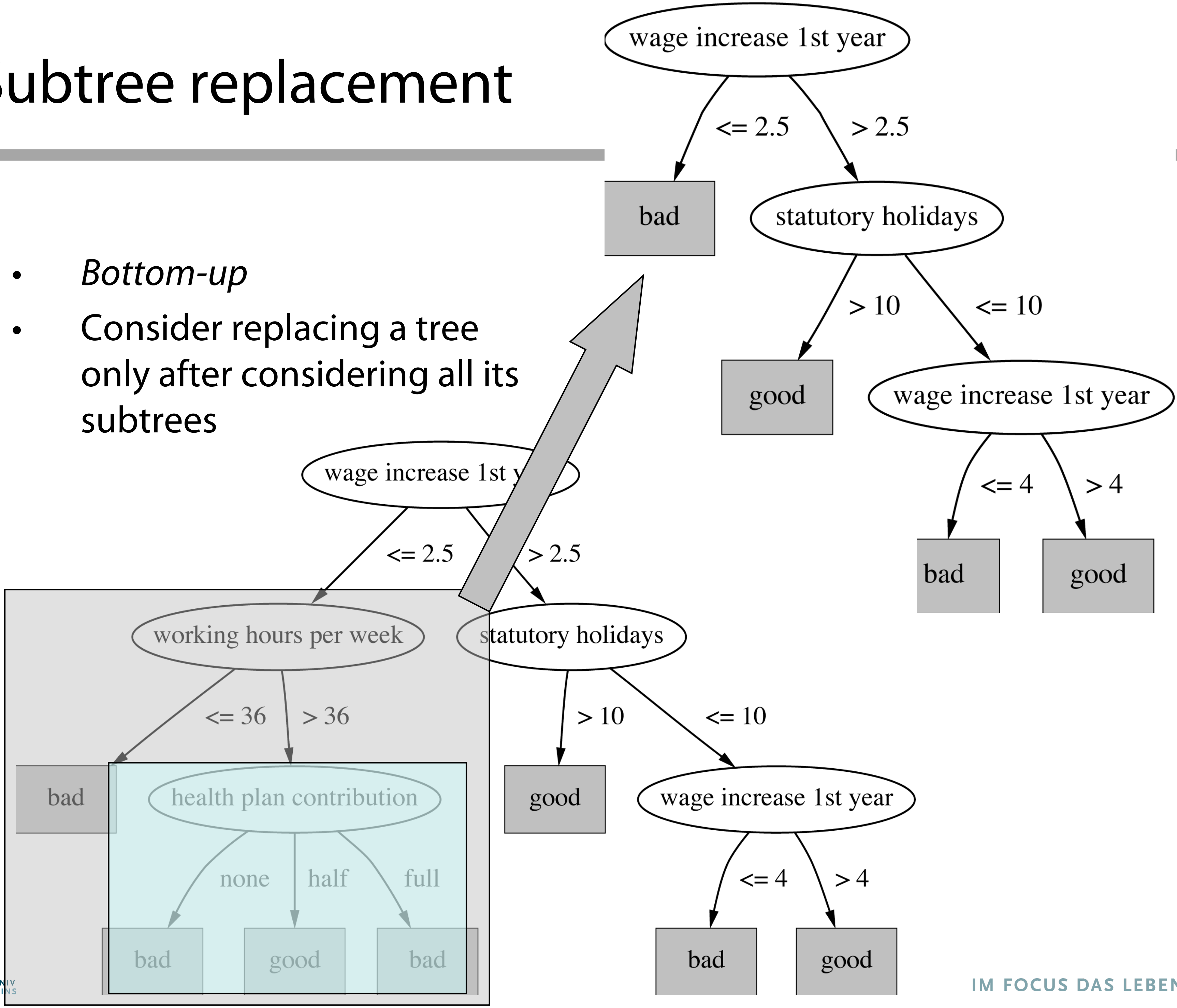
- Goal: Prevent overfitting to noise in the data
- Two strategies for "pruning" the decision tree:
  - Postpruning - take a fully-grown decision tree and discard unreliable parts
  - Prepruning - stop growing a branch when information becomes unreliable
- Postpruning preferred in practice—prepruning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

Post-pruning

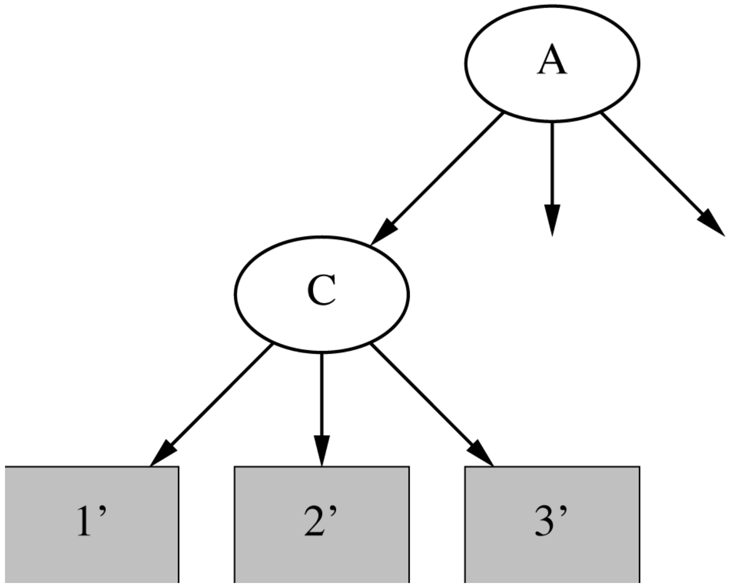
- First, build full tree
- Then, prune it
  - Fully-grown tree shows all attribute interactions
- Two pruning operations:
  1. *Subtree replacement*
  2. *Subtree raising*

Subtree replacement

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

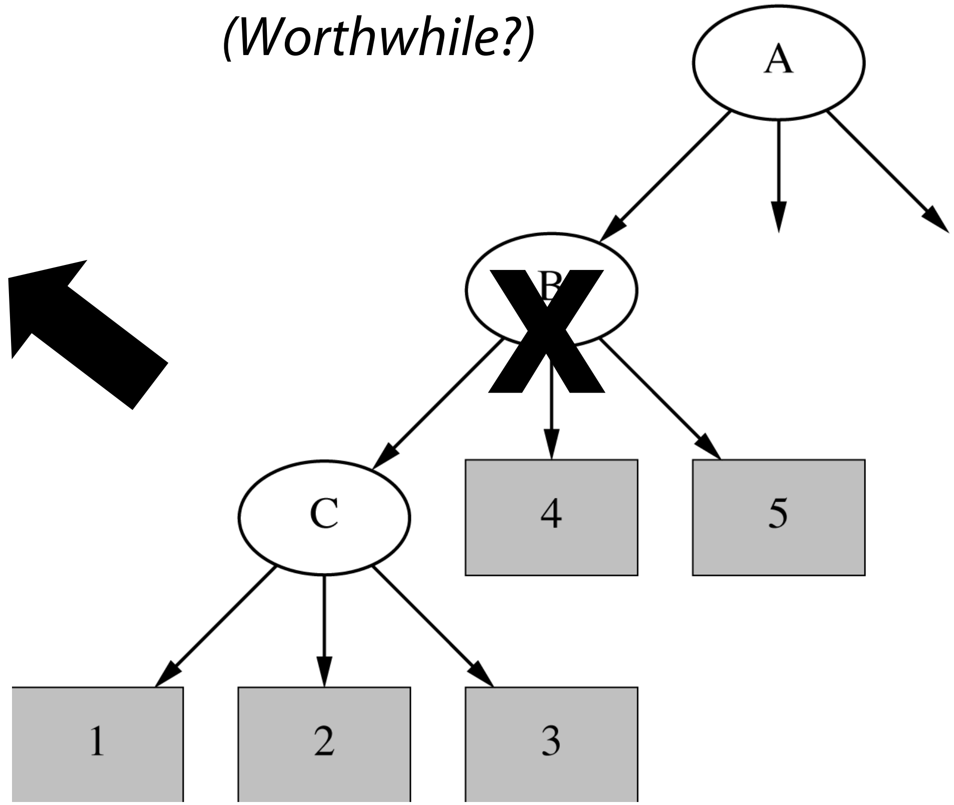


*Subtree raising



- Delete node
- Redistribute instances
- Slower than subtree replacement

  (Worthwhile?)



Post-pruning

- First, build full tree
- Then, prune it
  - Fully-grown tree shows all attribute interactions

→ Expected Error Pruning

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?

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

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

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

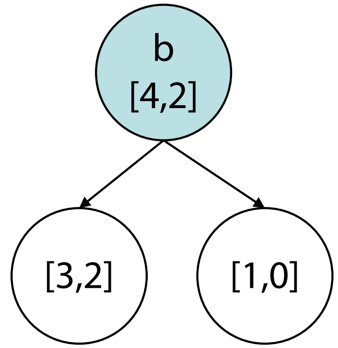
Backed-up Error

- For a non-leaf node Node
- Let children of Node be $Node_1$, $Node_2$, etc.
  - Probabilities can be estimated by relative frequencies of attribute values in sets of examples that fall into child nodes

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

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

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**

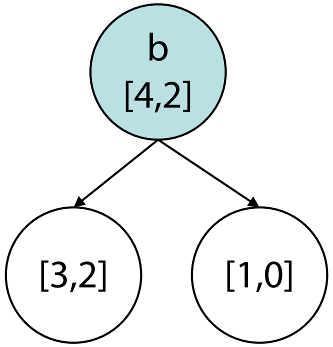
Example Calculation

- Static Expected Error of b

$$E([4,2]) = \frac{N - n + k}{N + k} \qquad 0{,}375$$

- Left child of b

$$E([3,2]) = \frac{5 - 3 + 2 - 1}{5 + 2} = 0{,}429$$



- Right child of b

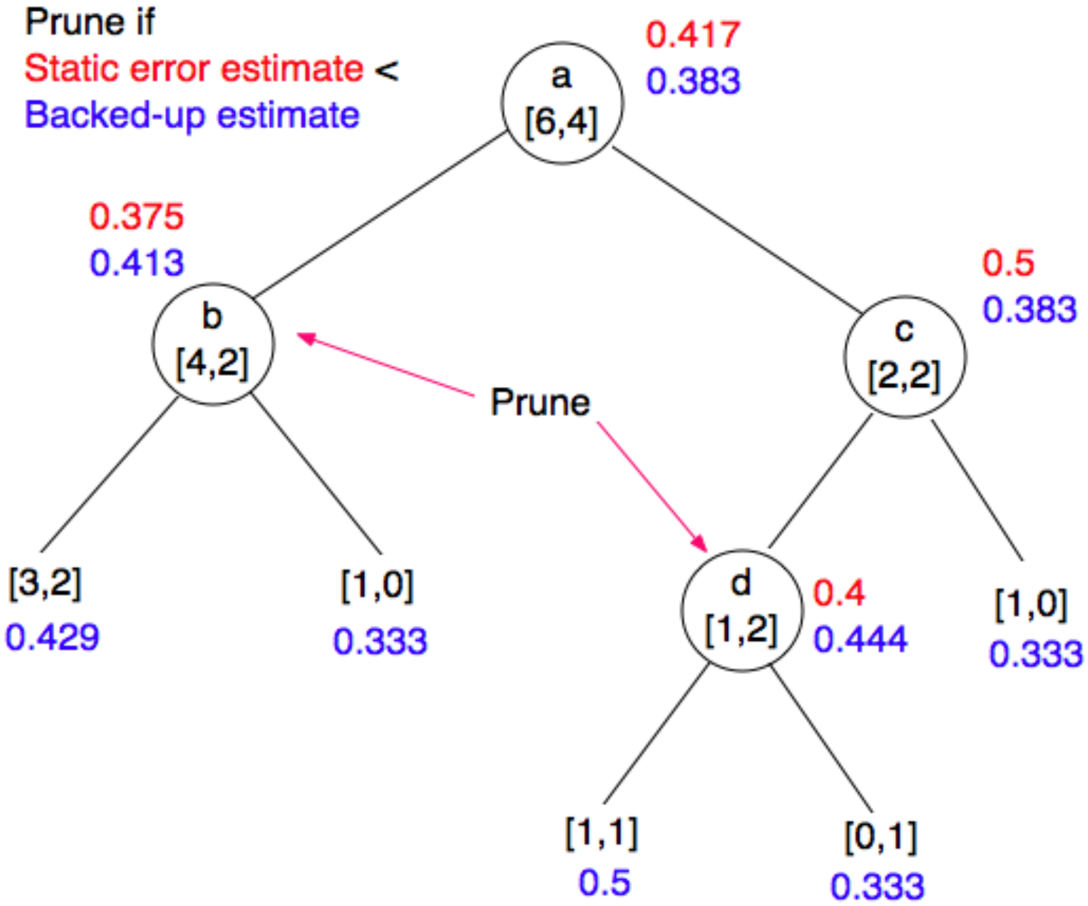
$$E([1,0]) = \frac{1 - 1 + 2 - 1}{1 + 2} = 0{,}333$$

- Backed Up Error of b

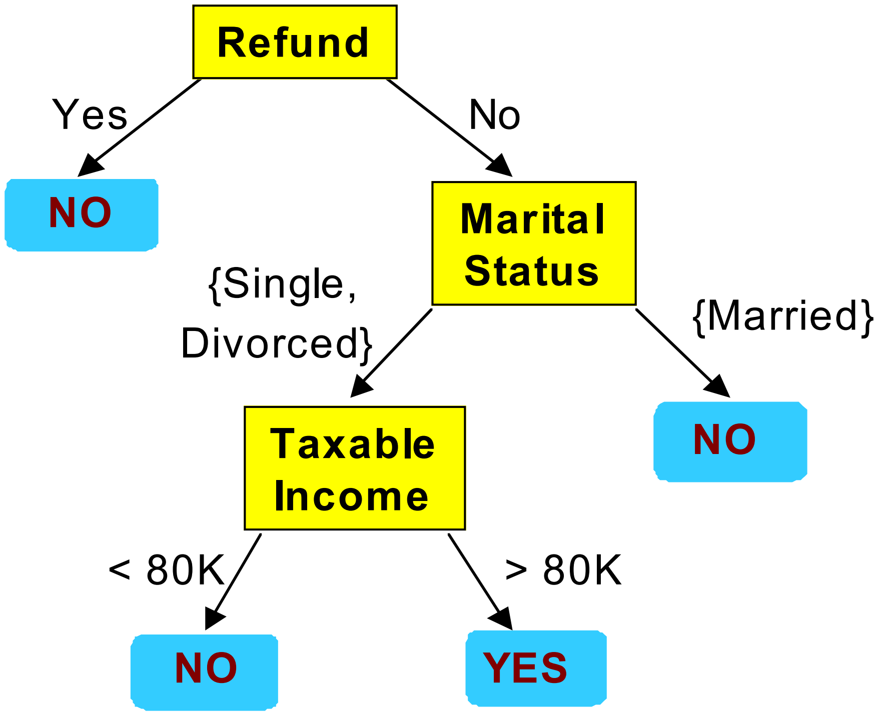
$$BackedUpError(b) = \frac{5}{6}E([3,2]) + \frac{1}{6}E([1,0]) = 0{,}413$$

- 0,375 < 0,413 → Prune tree.

Example


Prune if
Static error estimate <
Backed-up estimate
a
[6,4]
0.417
0.383
b
[4,2]
0.375
0.413
c
[2,2]
0.5
0.383
Prune
[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

From Decision Trees To Rules

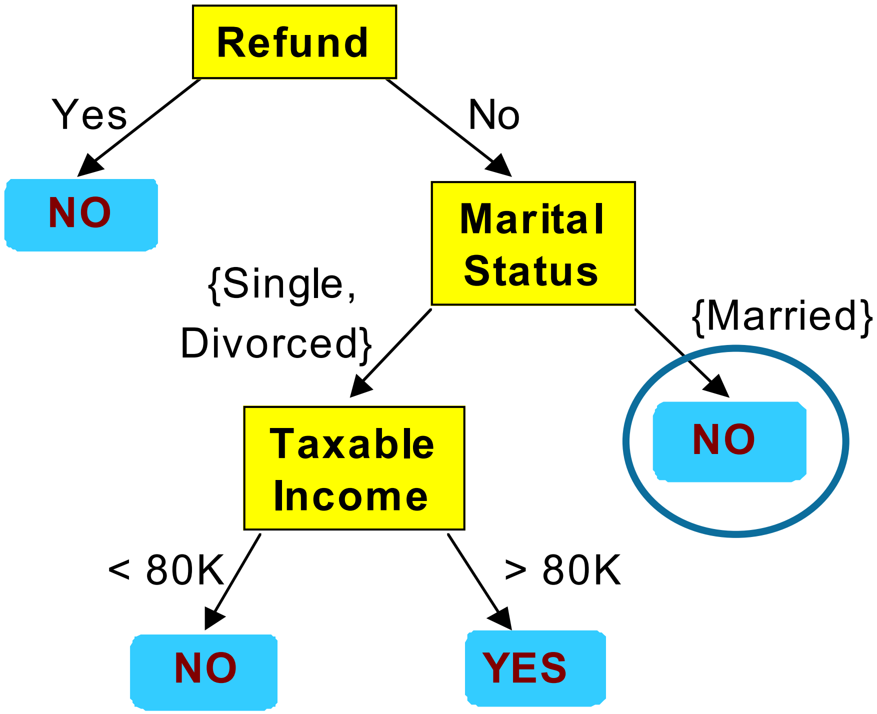


- Refund = Yes → No
- Refund = No ∧ Marital Status = {Single, Divorced} ∧ Taxable Income < 80k → No
- Refund = No ∧ Marital Status = {Single, Divorced} ∧ Taxable Income > 80k → Yes
- Refund = No ∧ Marital Status = Married → No

From Decision Trees to Rules

- Derive a rule set from a decision tree:
  Write a rule for each path from the root to a leaf.
  - The left-hand side is easily built from the label of the nodes and the labels of the arcs.
- Rules are mutually exclusive and exhaustive.
- Rule set contains as much information as the tree

Rules Can Be Simplified



Tid	Refund	Marital Status	Taxable 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

Initial Rule: $(\text{Refund=No}) \wedge (\text{Status=Married}) \rightarrow \text{No}$

Rules Can Be Simplified

- The resulting rules set can be simplified:
  - Let LHS be the left hand side of a rule.
  - Let LHS' be obtained from LHS by eliminating some conditions.
  - We can certainly replace LHS by LHS' in this rule if the subsets of the training set that satisfy respectively LHS and LHS' are equal.
  - A rule may be eliminated by using meta-conditions such as "if no other rule applies".

VSL vs DTL

- Decision tree learning (DTL) is more efficient if all examples are given in advance; else, it may produce successive hypotheses, each poorly related to the previous one
- Version space learning (VSL) is incremental
- DTL can produce simplified hypotheses that do not agree with all examples
- DTL has been more widely used in practice