Machine Learning

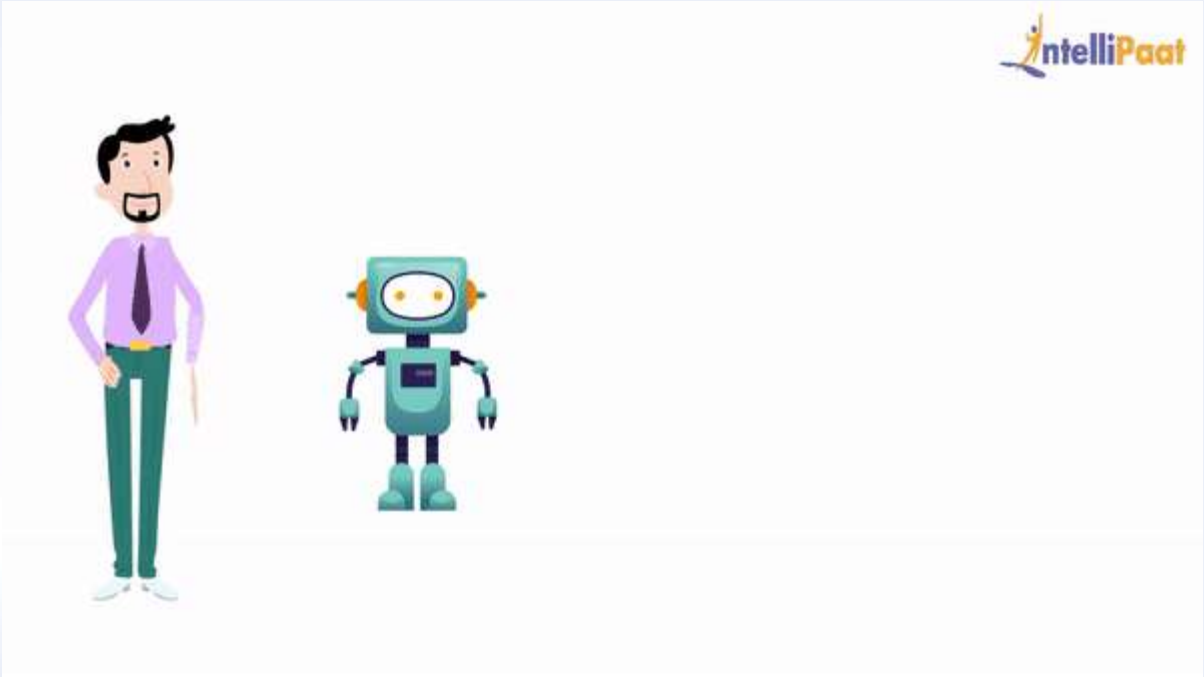
By
Ms. C.Jagadeeswari
Assistant Professor
Department of Computer Science and Engineering

Name of the Course	Machine Learning
Course Code	**CS601PC**
Year & Semester	B.Tech III Year II Sem
Section	CSE - B
Name of the Faculty	C. Jagadeeswari
Lecture Hour | Date	
Name of the Topic	Introduction to Machine Learning
Course Outcome(s)	

Introduction to Machine Learning

Evolution of Machines

Machine Learning Features


IntelliPaat

AI and ML defined

AI and ML Defined

- Artificial Intelligence
  - Attempt to build machines capable of simulating intelligent human behavior
- Machine Learning
  - The science of getting computers to act without being explicitly programmed – Stanford University

Applications of AI and ML

- When you do an Internet search using a facility like Google the search engine does not simply search based on the words entered
- Search algorithms will check what you have previously used to search and what you have clicked on, and then compare it to what others have looked for to give you a better selection for what you have typed in
- Search algorithms are constantly learning


GOOGLE MAPS
Product A
PRODUCTS
RECOMMENDATION
SELF DRIVING
CARS

Syllabus

UNIT - I

Introduction - Well-posed learning problems, designing a learning system, Perspectives and issues in machine learning

Concept learning and the general to specific ordering – introduction, a concept learning task, concept learning as search, find-S: finding a maximally specific hypothesis, version spaces and the candidate elimination algorithm, remarks on version spaces and candidate elimination, inductive bias.

Decision Tree Learning – Introduction, decision tree representation, appropriate problems for decision

tree learning, the basic decision tree learning algorithm, hypothesis space search in decision tree learning, inductive bias in decision tree learning, issues in decision tree learning.

What is the Learning Problem?

Learning = Improving with experience at some task

- Improve over task T,
- with respect to performance measure P,
- based on experience E.

E.g., Learn to play checkers

- T: Play checkers
- P: % of games won in world tournament
- E: opportunity to play against self

Learning to Play Checkers

- T: Play checkers
- P: Percent of games won in world tournament
- What experience?
- What exactly should be learned?
- How shall it be represented?
- What specific algorithm to learn it?

Designing a Learning System

1) Choosing a Training Experience
2) Choosing the Target Function
3) Choosing a representation for the Target Function
4) Choosing a Function Approximation Algorithm
    1) Estimating Training Values
    2) Adjusting the weights
5) Final Design

Choosing the Training Experience

The first design choice is to choose the *type of training experience* from which the system will learn.

There are three attributes which impact on success or failure of the learner

1. Whether the training experience provides *direct* or *indirect* feedback regarding the choices made by the performance system.
2. The degree to which the learner controls the *sequence of training examples*
3. How well it represents the *distribution of examples* over which the final system performance *P* must be measured.

1. Whether the training experience provides *direct or indirect feedback* regarding the choices made by the performance system.

For example, in checkers game

Direct training examples consisting of individual *checkers board states and the correct move for each*.

Indirect training examples consisting of the *move sequences and final outcomes of various games played*. Learner faces an additional problem of *credit assignment*

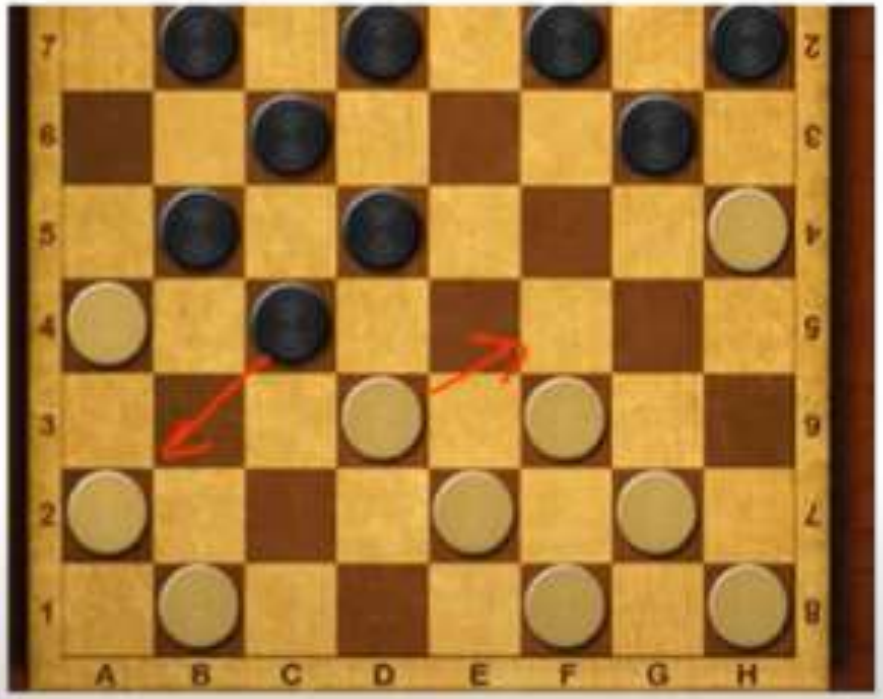


Image Reference : https://playpager.com/checkers-online/

2. A second important attribute of the training experience is the *degree to which the learner controls the sequence of training examples*

For example, in checkers game:

The learner might depends on the teacher to select informative board states and to provide the correct move for each.

Alternatively, the learner might itself propose board states that it finds particularly confusing and ask the teacher for the correct move.

The learner may have complete control over both the board states and (indirect) training classifications, as it does when it learns by playing against itself with no teacher present.

3. A third attribute of the training experience is how well it represents *the distribution of examples* over which the final system performance P must be measured.

For example, in checkers game:

In checkers learning scenario, the performance metric P is the percent of games the system wins in the world tournament.

Type of Training Experience

- Direct or indirect?
- Teacher or not?

A problem: is training experience representative of performance goal?

Choosing the Target Function

The next design choice is to determine exactly what type of knowledge will be learned and how this will be used by the performance program.

For example, in checkers game:

The program needs only to learn how to choose the best move from among these legal moves.

1. Let ***ChooseMove*** be the target function

$$ChooseMove : B \longrightarrow M$$

2. An alternative target function is an *evaluation function* that assigns a numerical score to any given board state

- Let the target function V

$$V : B \longrightarrow R$$

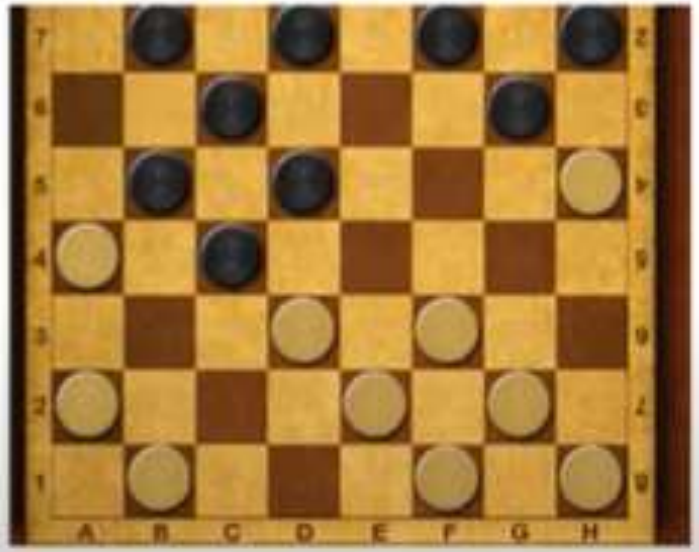
- Let us define the target value V(b) for an arbitrary board state b in B, as follows:
  1. if b is a final board state that is won, then V(b) = 100
  2. if b is a final board state that is lost, then V(b) = -100
  3. if b is a final board state that is drawn, then V(b) = 0
  4. if b is a not a final state in the game, then V(b) = V(b')

where b' is the best final board state that can be achieved starting from b and playing optimally until the end of the game

Choosing a Representation for the Target Function

Let us choose a simple representation - for any given board state, the function c will be calculated as a linear combination of the following board features:

- x_1: the number of black pieces on the board
- x_2: the number of red pieces on the board
- x_3: the number of black kings on the board
- x_4: the number of red kings on the board
- x_5: the number of black pieces threatened by red (i.e., which can be captured on red's next turn)
- x_6: the number of red pieces threatened by black



3. Choosing a representation of the target function

Thus, learning program will represent as a linear function of the form

$$\hat{V}(b) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5 + w_6 x_6$$

Where,

- w_0 through w_6 are numerical coefficients, or weights, to be chosen by the learning algorithm.
- Learned values for the weights w_1 through w_6 will determine the relative importance of the various board features in determining the value of the board
- The weight w_0 will provide an additive constant to the board value

Partial Design

Partial design of a checkers learning program:

- Task T: playing checkers
- Performance measure P: percent of games won in the world tournament
- Training experience E: games played against itself
- Target function: *V: Board* $\longrightarrow$ *R*
- Target function representation

$$\hat{V}(b) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5 + w_6 x_6$$

Choosing a Function Approximation Algorithm

- In order to learn the target function f we require a set of training examples, each describing a specific board state b and the training value $V_{train}(b)$ for b.

- Each training example is an ordered pair of the form $(b, V_{train}(b))$.

Example: A board state b in which black has won the game

$$((x_1=3, x_2=0, x_3=1, x_4=0, x_5=0, x_6=0), +100)$$

Function Approximation Procedure

1. Derive training examples from the indirect training experience available to the learner
2. Adjusts the weights w_i to best fit these training examples

1. Estimating training values

A simple approach for estimating training values for intermediate board states is to assign the training value of $V_{train}(b)$ for any intermediate board state b to be $\hat{V}(Successor(b))$

Where ,

- $\hat{V}$ is the learner's current approximation to V
- Successor(b) denotes the next board state following b for which it is again the program's turn to move

Rule for estimating training values

$$V_{train}(b) \leftarrow \hat{V}(Successor(b))$$

2. Adjusting the weights

Specify the learning algorithm for choosing the weights w_i to best fit the set of training examples $\{(b, V_{train}(b))\}$

- A first step is to define what we mean by the bestfit to the training data.
- One common approach is to define the best hypothesis, or set of weights, as that which minimizes the squared error E between the training values and the values predicted by the hypothesis.

$$E \equiv \sum_{\langle b, V_{train}(b)\rangle \in \text{ training examples}} (V_{train}(b) - \hat{V}(b))^2$$

Several algorithms are known for finding weights of a linear function that minimize E.

Least mean squares or ***LMS training rule.***

LMS weight update rule :- For each training example $(b, V_{train}(b))$

Use the current weights to calculate $\hat{V}(b)$

For each weight w_i, update it as

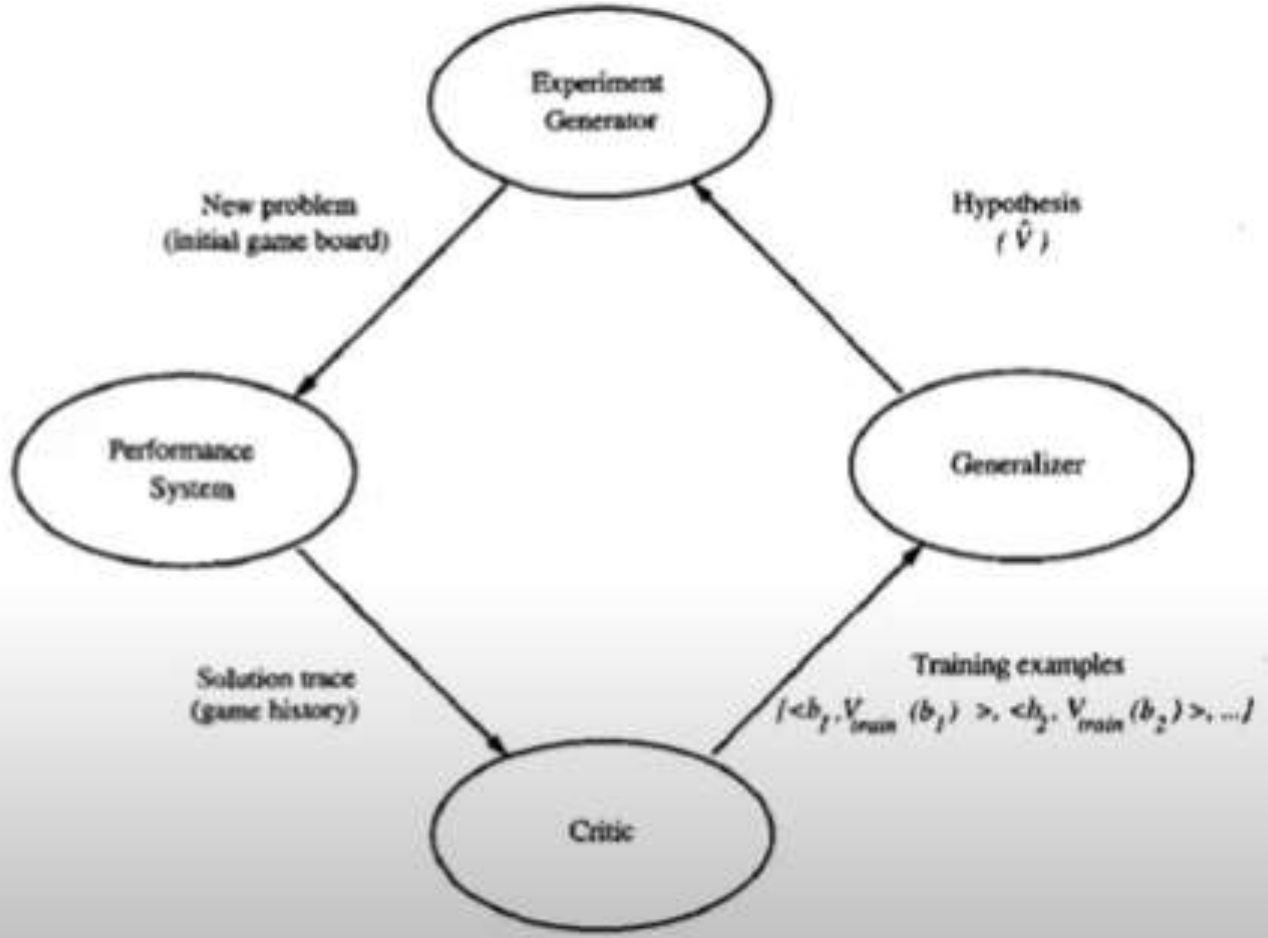
$$w_i \leftarrow w_i + \eta \, (V_{train}(b) - \hat{V}(b)) \, x_i$$

Here η is a small constant (e.g., 0.1) that moderates the size of the weight update.

5. Final Design

The final design of checkers learning system can be described by four distinct program modules that represent the central components in many learning systems

1. The Performance System
2. The Critic
3. The Generalizer
4. The Experiment Generator



Designing the Checkers Learning Algorithm

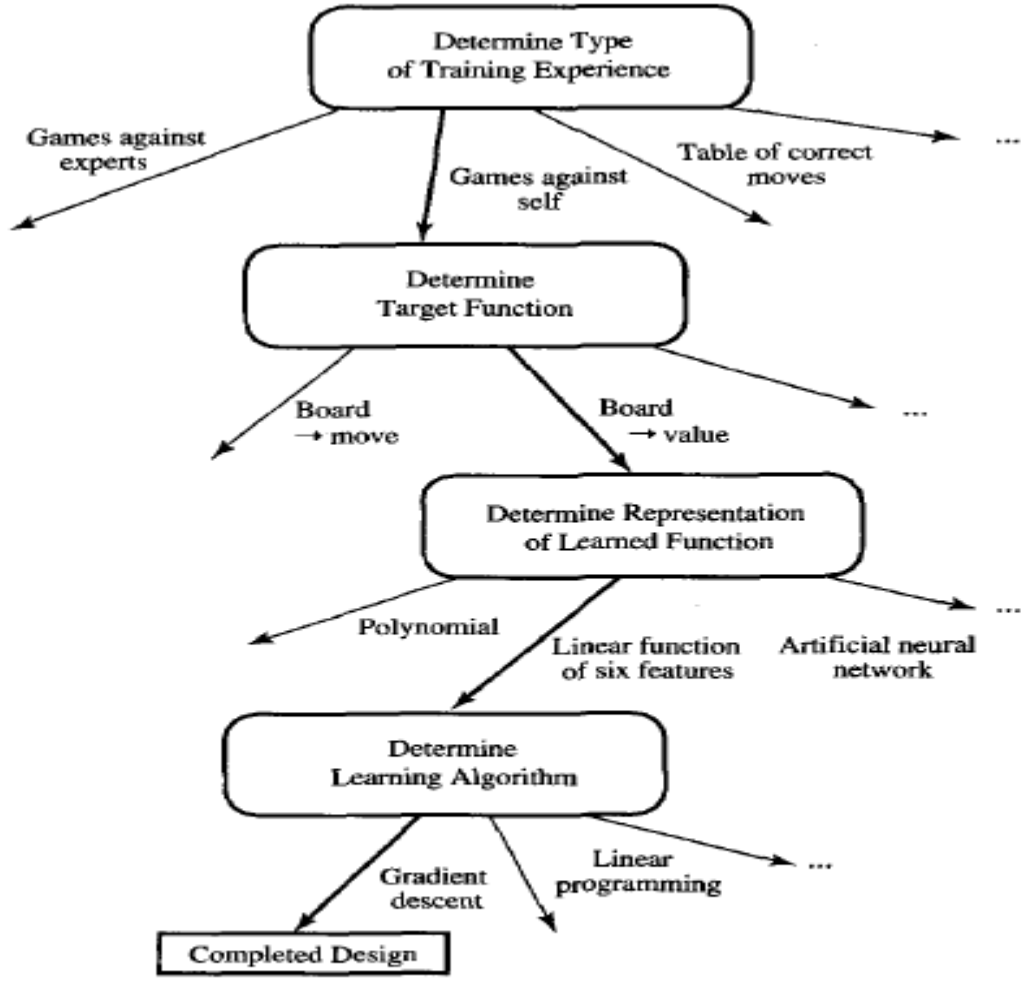


FIGURE 1.2
Summary of choices in designing the checkers learning program.

Concept Learning

What is a Concept

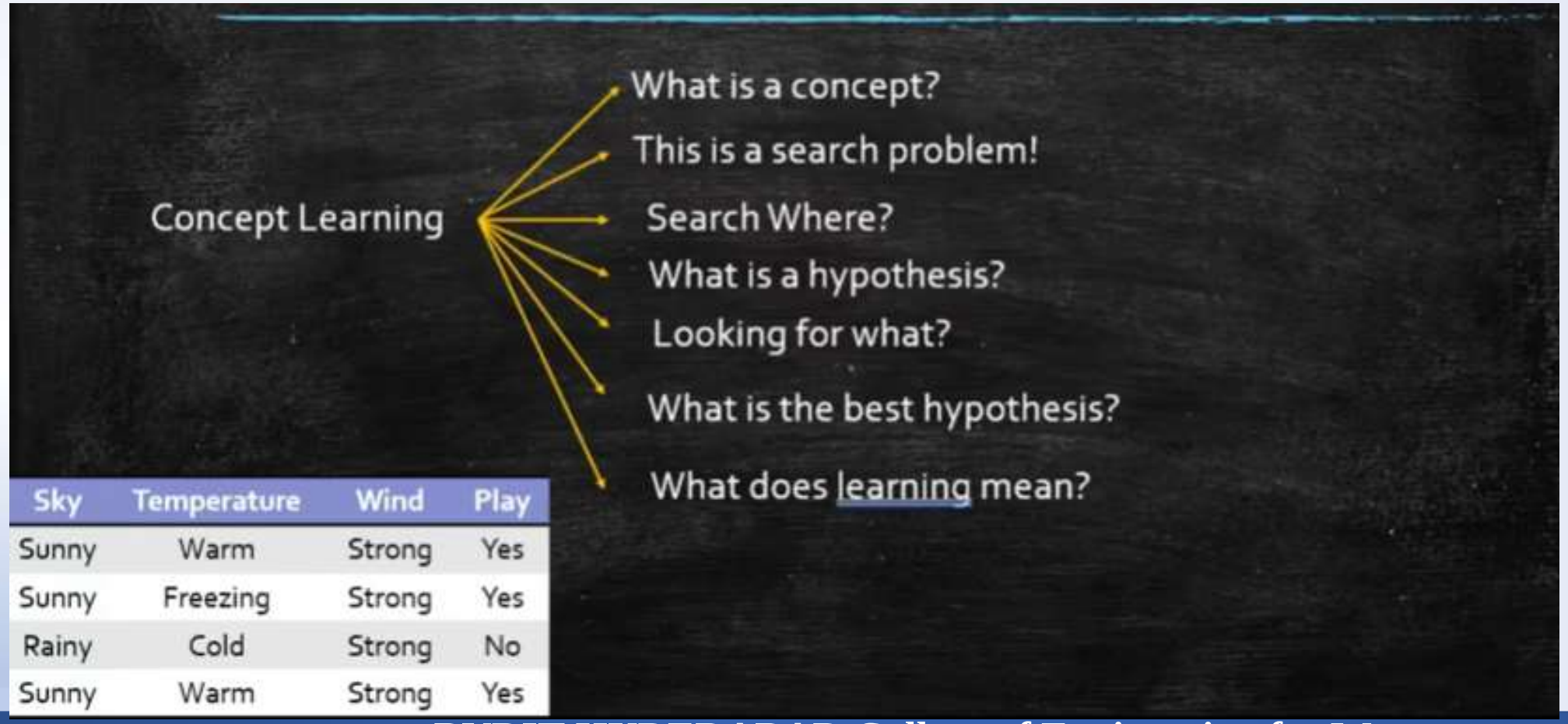
Each concept can be viewed as describing some subset of objects or events defined over a larger set (e.g., the subset of animals that constitute birds).

Alternatively, each concept can be thought of as a boolean-valued function defined over this larger set (e.g., a function defined over all animals, whose value is true for birds and false for other animals).

Concept learning. Inferring a boolean-valued function from training examples of its input and output

Concept Learning

In a **concept learning** task, a human or **machine** learner is trained to classify objects by being shown a set of example objects along with their class labels. The learner simplifies what has been observed by condensing it in the form of an example.



Sky	Temperature	Wind	Play
Sunny	Warm	Strong	Yes
Sunny	Freezing	Strong	Yes
Rainy	Cold	Strong	No
Sunny	Warm	Strong	Yes

Which days does SRK come out to enjoy sports?

- Sky condition
- Humidity
- Temperature
- Wind
- Water
- Forecast

- Attributes of a day: takes on values

Learning Task

- We want to make a hypothesis about the day on which SRK comes out..
  - in the form of a boolean function on the attributes of the day.

- Find the right hypothesis/function from historical data

Training Examples for EnjoySport

Sky	Temp	Humid	Wind	Water	Forecst	EnjoySpt
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

- Negative and positive learning examples
- Concept learning:

  c is the target concept

  - Deriving a Boolean function from training examples
    - Many "hypothetical" boolean functions
      - Hypotheses; find h such that h = c.
    - Other more complex examples:
      - Non-boolean functions
- Generate hypotheses for concept from Training Examples

Representing Hypotheses

- Task of finding appropriate set of hypotheses for concept given training data
- Represent hypothesis as Conjunction of constraints of the following form:
  - Values possible in any hypothesis
- Specific value : Water = *Warm* **The most general hypothesis-that every day is a positive example-is represented by (?, ?, ?, ?, ?, ?)**
- **The most specific possible hypothesis-that no day is a positive example-is represented by (0,0,0,0,0,0)**
    - Don't-care value: Water = ?
    - No value allowed : Water = ∅
      - i.e., no permissible value given values of other attributes
  - Use vector of such values as hypothesis:
    - ⟨ Sky AirTemp Humid Wind Water Forecast ⟩
      - Example: ⟨*Sunny* ? ? *Strong* ? *Same* ⟩
- Idea of *satisfaction of hypothesis* by some example
  - say "example satisfies hypothesis"
  - defined by a function *h(x):*

$$h(x) = 1 \text{ if } h \text{ is true on } x$$

$$= 0 \text{ otherwise}$$

- Want hypothesis that best fits examples:
  - Can reduce learning to search problem over space of hypotheses

Prototypical Concept Learning Task

TASK T: predicting when person will enjoy sport

- **Target function c**: $\text{EnjoySport} : X \rightarrow \{0, 1\}$
- Cannot, in general, know Target function c
  - Adopt hypotheses H about c
- Form of hypotheses H:
  - Conjunctions of literals $\langle ?, \text{Cold}, \text{High}, ?, ?, ? \rangle$

■ **EXPERIENCE E**

- **Instances X**: possible days described by attributes $Sky, AirTemp, Humidity, Wind, Water, Forecast$
- **Training examples** D: Positive/negative examples of target function $\{\langle x_1, c(x_1)\rangle, \ldots \langle x_m, c(x_m)\rangle\}$

■ **PERFORMANCE MEASURE P**: Hypotheses h in H such that $h(x) = c(x)$ for all x in D ()

- There may exist several alternative hypotheses that fit examples

- **Given:**
  - Instances X: Possible days, each described by the attributes
    - *Sky* (with possible values *Sunny*, *Cloudy*, and *Rainy*),
    - *AirTemp* (with values *Warm* and *Cold*),
    - *Humidity* (with values *Normal* and *High*),
    - *Wind* (with values *Strong* and *Weak*),
    - *Water* (with values *Warm* and *Cool*), and
    - *Forecast* (with values *Same* and *Change*).
  - Hypotheses H: Each hypothesis is described by a conjunction of constraints on the attributes *Sky*, *AirTemp*, *Humidity*, *Wind*, *Water*, and *Forecast*. The constraints may be "?" (any value is acceptable), "Ø" (no value is acceptable), or a specific value.
  - Target concept c: $EnjoySport : X \rightarrow \{0, 1\}$
  - Training examples D: Positive and negative examples of the target function (see Table 2.1).
- **Determine:**
  - A hypothesis h in H such that $h(x) = c(x)$ for all x in X.

TABLE 2.2
The *EnjoySport* concept learning task.

Inductive Learning Hypothesis

Any hypothesis found to approximate the target function well over a sufficiently large set of training examples will also approximate the target function well over other unobserved examples

Concept Learning as a Search

Concept learning can be viewed as the task of searching through a large space of hypotheses implicitly defined by the hypothesis representation.
The goal of this search is to find the hypothesis that best fits the training examples

Example: The instances ***X*** and hypotheses ***H*** in the ***EnjoySport*** learning task.

- The attribute ***Sky*** has three possible values, and ***AirTemp, Humidity, Wind, Water, Forecast*** each have two possible values, the instance space X contains exactly

3.2.2.2.2.2 = 96 Distinct instances

5.4.4.4.4.4 = 5120 Syntactically distinct hypotheses within H.

- Every hypothesis containing one or more " Φ" symbols represents the empty set of instances; that is, it classifies every instance as ***negative.***

1 + (4.3.3.3.3.3) = 973. Semantically distinct hypotheses

General-to-Specific Ordering of Hypotheses

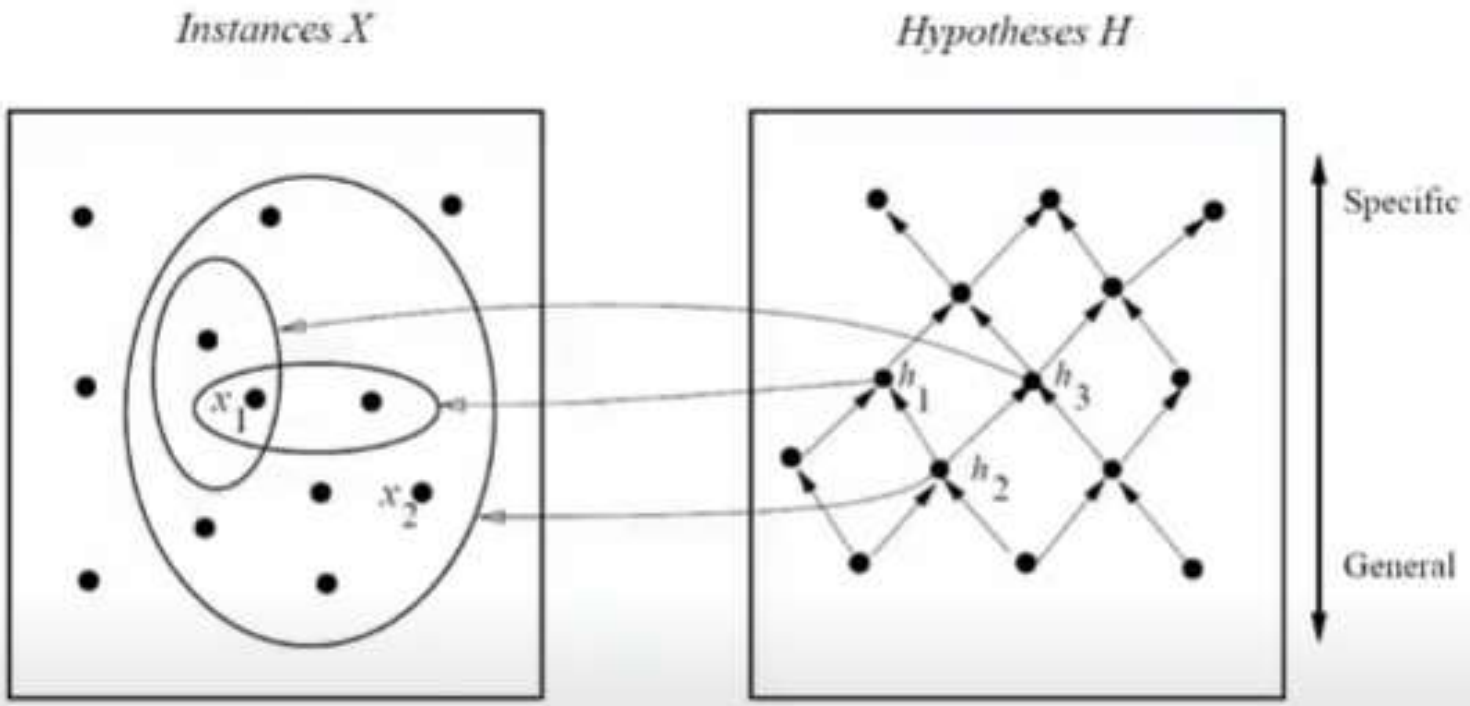
Consider the two hypotheses

$h_1 = <$ Sunny, ?, ?, Strong, ?, ? $>$

$h_2 = <$ Sunny, ?, ?, ?, ?, ? $>$

- Consider the sets of instances that are classified positive by h_1 and by h_2.
- h_2 imposes fewer constraints on the instance, it classifies more instances as positive. So, any instance classified positive by h_1 will also be classified positive by h_2. Therefore, h_2 is more general than h_1.

Definition: Let ***hj and hk be boolean-valued functions defined over X. Then hj is*** **moregeneral-than-or-equal-to** ***hk (written hj 2, hk) if and only if*** $(\forall x \in X)[(h_k(x) = 1) \rightarrow (h_j(x) = 1)]$



x_1 = <Sunny, Warm, High, Strong, Cool, Same>
x_2 = <Sunny, Warm, High, Light, Warm, Same>

h_1 = <Sunny, ?, ?, Strong, ?, ?>
h_2 = <Sunny, ?, ?, ?, ?, ?>
h_3 = <Sunny, ?, ?, ?, Cool, ?>

Find-S Algorithm

Assumes

There is hypothesis h in H describing target function c

There are no errors in the TEs

Procedure

1. Initialize *h* to the most specific hypothesis in *H (what is this?)*
2. For each *positive* training instance *x*

   For each attribute constraint a_i in *h*

   If the constraint a_i in *h* is satisfied by *x*

   do nothing

   Else

   replace a_i in *h* by the next more general constraint that is satisfied by *x*
3. Output hypothesis *h*

Note

There is no change for a negative example, so they are ignored.

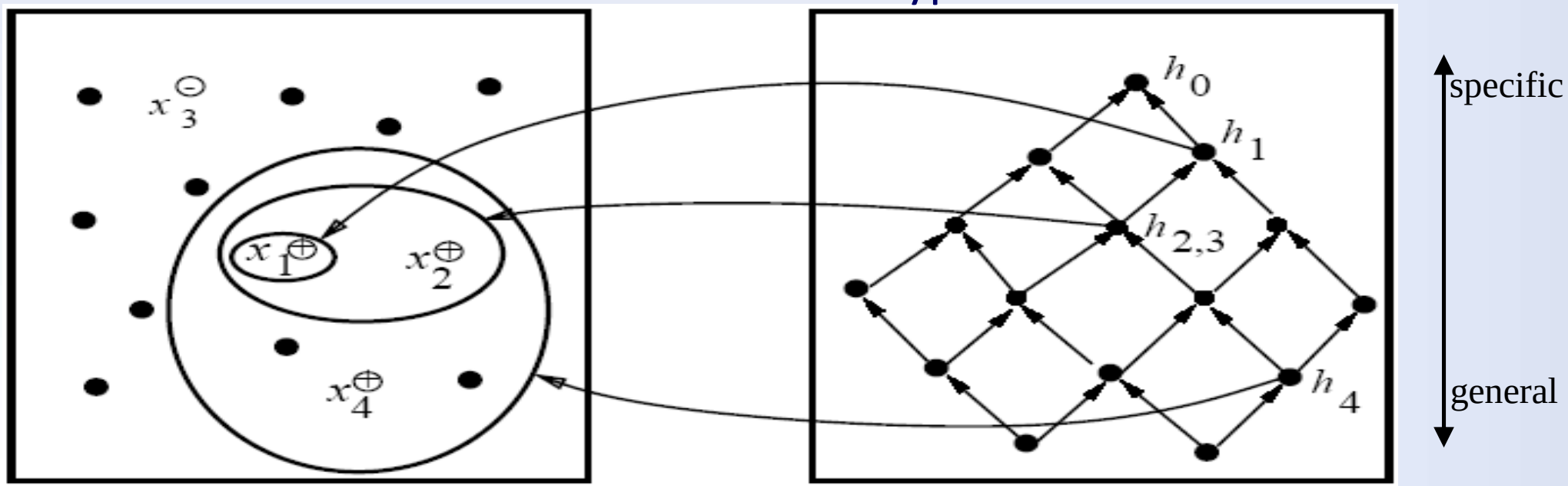
This follows from assumptions that there is h in H describing target function c *(ie.,for this h, h=c)*

and that there are no errors in data. In particular, it follows that the hypothesis at any stage cannot be changed by neg example

Assumption: Everything except the positive examples is negative

Example of Find-S

Instances X Hypotheses H



x_1=⟨*Sunny Warm Normal Strong Warm Same*⟩ +

x_2=⟨*Sunny Warm High Strong Warm Same*⟩ +

x_3=⟨*Rainy Cold High Strong Warm Change*⟩ –

x_4=⟨*Sunny Warm High Strong Cool Change*⟩ +

h_0=⟨∅ ∅ ∅ ∅ ∅ ∅⟩

h_1=⟨*Sunny Warm Normal Strong Warm Same*⟩

h_2=⟨*Sunny Warm ? Strong Warm Same*⟩

h_3=⟨*Sunny Warm ? Strong Warm Same*⟩

h_4=⟨*Sunny Warm ? Strong ? ?*⟩

Problems with Find-S

- Problems:
  - Throws away information!
    - Negative examples
  - Can't tell whether it has learned the concept
    - Depending on *H*, there might be several h's that fit TEs!
    - Picks a maximally specific *h* (why?)
  - Can't tell when training data is inconsistent
    - Since ignores negative TEs
- But
  - It is simple
  - Outcome is independent of order of examples
    - Why?
- What alternative overcomes these problems?
  - Keep *all* consistent hypotheses!
    - Candidate elimination algorithm

Consistent Hypotheses and Version Space

- A hypothesis h is consistent with a set of training examples D of target concept c
  if $h(x) = c(x)$ for each training example $\langle x, c(x)\rangle$ in D
  - Note that consistency is with respect to specific D.
- Notation:

$$Consistent(h, D) \equiv \forall \langle x, c(x)\rangle \in D :: h(x) = c(x)$$

- The version space, $VS_{H,D}$, with respect to hypothesis space H and training examples D, is the subset of hypotheses from H consistent with D
- Notation:

$$VS_{H,D} = \{h \mid h \in H \wedge Consistent(h, D)\}$$

List-Then-Eliminate Algorithm

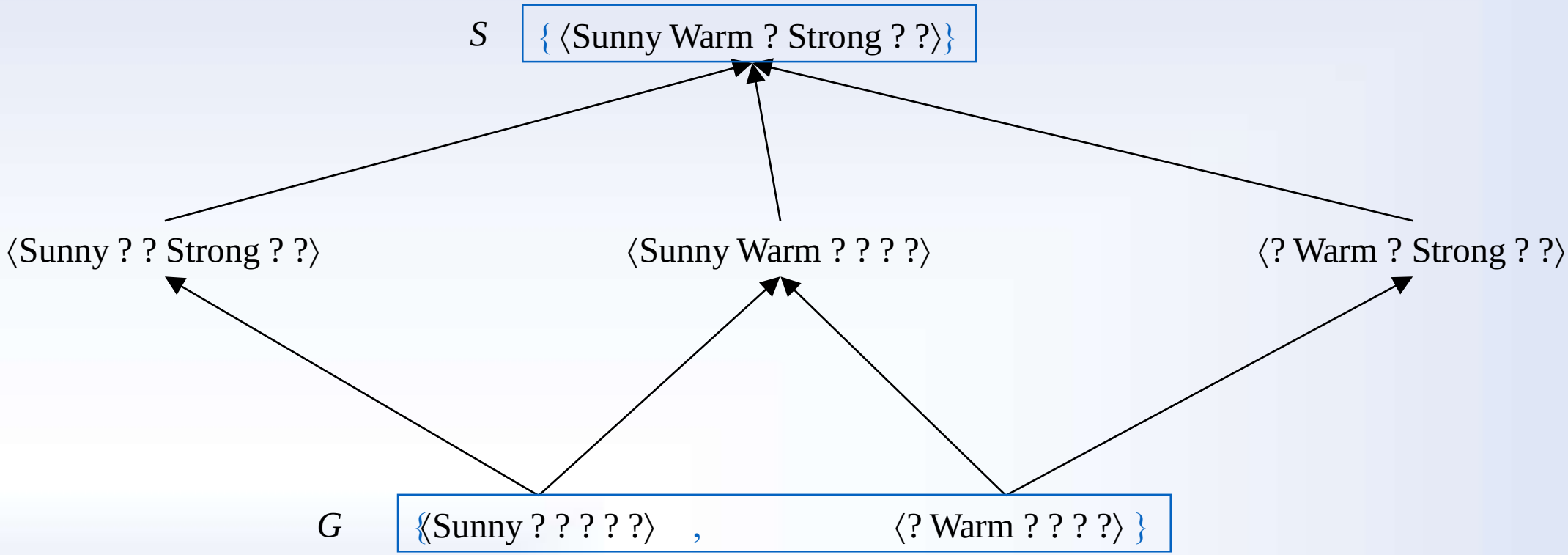
1. *VersionSpace* $\leftarrow$ list of all hypotheses in *H*
2. For each training example $\langle x, c(x) \rangle$

   remove from *VersionSpace* any hypothesis *h* for which $h(x) \neq c(x)$
3. Output the list of hypotheses in *VersionSpace*
4. This is essentially a brute force procedure

Representing Version Spaces

- Want more compact representation of VS
  - Store most/least general boundaries of space
  - Generate all intermediate h's in VS
  - Idea that any h in VS must be consistent with all TE's
    - Generalize from most specific boundaries
    - Specialize from most general boundaries
- The general boundary, *G*, of version space $VS_{H,D}$ is the set of its maximally general members consistent with D
  - Summarizes the negative examples; anything more general will cover a negative TE
- The specific boundary, *S*, of version space $VS_{H,D}$ is the set of its maximally specific members consistent with D
  - Summarizes the positive examples; anything more specific will fail to cover a positive TE

Version Space for this Example



Candidate Elimination Algorithm

$G \leftarrow$ maximally general hypotheses in H
$S \leftarrow$ maximally specific hypotheses in H

For each training example d, do

- If d is positive
  - Remove from G every hypothesis inconsistent with d
  - For each hypothesis s in S that is inconsistent with d
    - Remove s from S
    - Add to S all minimal generalizations h of s such that
      1. h is consistent with d, and
      2. some member of G is more general than h
  - Remove from S every hypothesis that is more general than another hypothesis in S

Candidate Elimination Algorithm (cont)

- If *d* is a negative example
  - Remove from *S* every hypothesis inconsistent with *d*
  - For each hypothesis *g* in *G* that is inconsistent with *d*
    - Remove *g* from *G*
    - Add to *G* all minimal specializations *h* of *g* such that
      1. *h* is consistent with *d*, and
      2. some member of *S* is more specific than *h*
  - Remove from *G* every hypothesis that is less general than another hypothesis in *G*
- Essentially use
  - Pos TEs to generalize S
  - Neg TEs to specialize G
- Independent of order of TEs
- Convergence guaranteed if:
  - ***no errors***
  - ***there is h in H describing c***.

Example

S_0 $\{\langle \varnothing\ \varnothing\ \varnothing\ \varnothing\ \varnothing\ \varnothing \rangle\}$

G_0 $\{\langle ?\ ?\ ?\ ?\ ?\ ? \rangle\}$

Recall : If d is positive

Remove from G every hypothesis inconsistent with d

For each hypothesis s in S that is inconsistent with d

- Remove s from S
- Add to S all minimal generalizations h of s that are specializations of a hypothesis in G
- Remove from S every hypothesis that is more general than another hypothesis in S

$\langle$*Sunny Warm Normal Strong Warm Same*$\rangle$ +

S_1 $\{\langle$Sunny Warm Normal Strong Warm Same$\rangle\}$

G_1 $\{\langle ?\ ?\ ?\ ?\ ?\ ? \rangle\}$

Example (contd)

S_1 {⟨Sunny Warm Normal Strong Warm Same⟩}

G_1 {⟨? ? ? ? ? ?⟩}

⟨*Sunny Warm High Strong Warm Same*⟩ **+**

S_2 {⟨Sunny Warm ? Strong Warm Same⟩}

G_2 {⟨? ? ? ? ? ?⟩}

Example (contd)

S_2 {⟨Sunny Warm ? Strong Warm Same⟩}

Recall: If d is a negative example

G_2 {⟨? ? ? ? ? ?⟩}

- Remove from S every hypothesis inconsistent with d
- For each hypothesis g in G that is inconsistent with d
  - ❖ Remove g from G
  - ❖ Add to G all minimal specializations h of g that generalize some hypothesis in S
  - ❖ Remove from G every hypothesis that is less general than another hypothesis in G

⟨*Rainy Cold High Strong Warm Change*⟩ –

S_3 {⟨Sunny Warm ? Strong Warm Same⟩}

Current G boundary is incorrect
So, need to make it more specific.

G_3 { ⟨Sunny ? ? ? ? ?⟩ , ⟨? Warm ? ? ? ?⟩ , ⟨? ? ? ? ? Same⟩

Example (contd)

- Why are there no hypotheses left relating to:
  - ⟨ *Cloudy ? ? ? ? ?*⟩
- The following specialization using the third value

  ⟨*? ? Normal ? ? ?*⟩,

  is not more general than the specific boundary

{⟨Sunny Warm ? Strong Warm Same⟩}

- The specializations ⟨*? ? ? Weak ? ?*⟩,

  ⟨*? ? ? ? Cool ?*⟩ are also inconsistent with S

Example (contd)

S_3 {⟨Sunny Warm ? Strong Warm Same⟩}

G_3 {⟨Sunny ? ? ? ? ?⟩, ⟨? Warm ? ? ? ?⟩, ⟨? ? ? ? ?Same⟩}

⟨*Sunny Warm High Strong Cool Change*⟩ +

S_4 {⟨Sunny Warm ? Strong ? ?⟩}

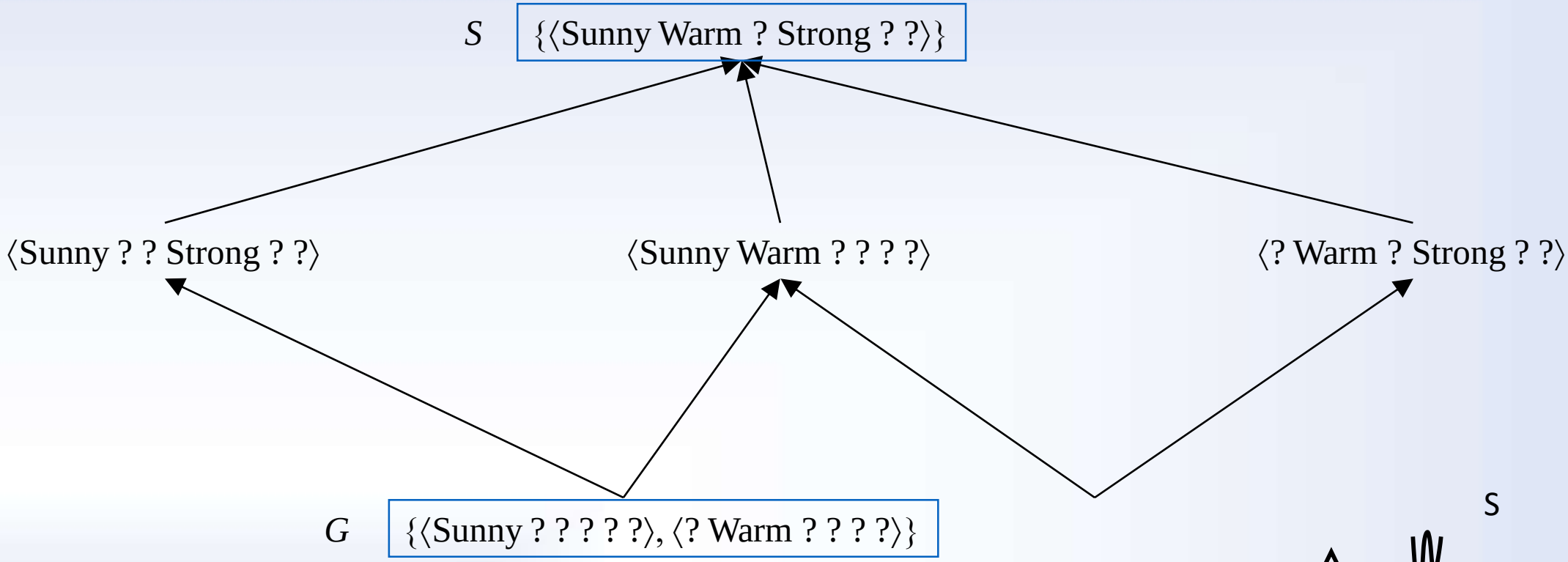
G_4 {⟨Sunny ? ? ? ? ?⟩, ⟨? Warm ? ? ? ?⟩}

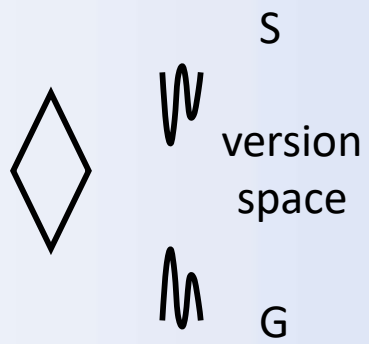
Example (contd)

⟨*Sunny Warm High Strong Cool Change*⟩ +

- Why does this example remove a hypothesis from G?:
  - ⟨*? ? ? ? ? Same*⟩
- This hypothesis
  - Cannot be specialized, since would not cover new TE
  - Cannot be generalized, because more general would cover negative TE.
  - Hence must drop hypothesis.

Version Space of the Example





Convergence of algorithm

- Convergence guaranteed if:
  - ***no errors***
  - ***there is h in H describing c***.
- Ambiguity removed from VS when S = G
  - Containing single h
  - When have seen enough TEs
- If have false negative TE, algorithm will remove every h consistent with TE, and hence will remove correct target concept from VS
  - If observe enough TEs will find that S, G boundaries converge to empty VS

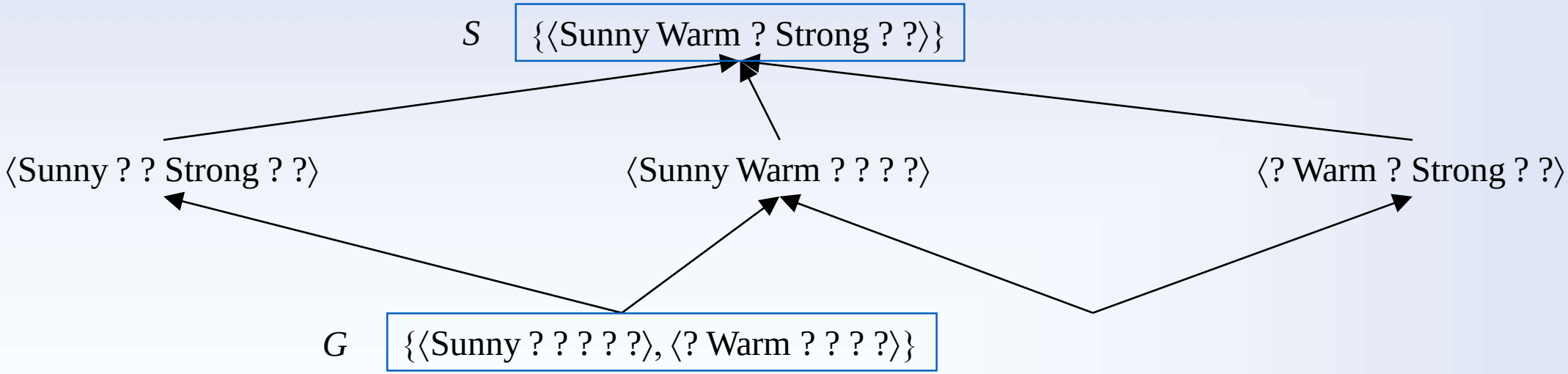
Let us try this

Origin	Manufacturer	Color	Decade	Type	
Japan	Honda	Blue	1980	Economy	+
Japan	Toyota	Green	1970	Sports	-
Japan	Toyota	Blue	1990	Economy	+
USA	Chrysler	Red	1980	Economy	-
Japan	Honda	White	1980	Economy	+

And this

Origin	Manufacturer	Color	Decade	Type	
Japan	Honda	Blue	1980	Economy	+
Japan	Toyota	Green	1970	Sports	-
Japan	Toyota	Blue	1990	Economy	+
USA	Chrysler	Red	1980	Economy	-
Japan	Honda	White	1980	Economy	+
Japan	Toyota	Green	1980	Economy	+
Japan	Honda	Red	1990	Economy	-

Which Next Training Example?

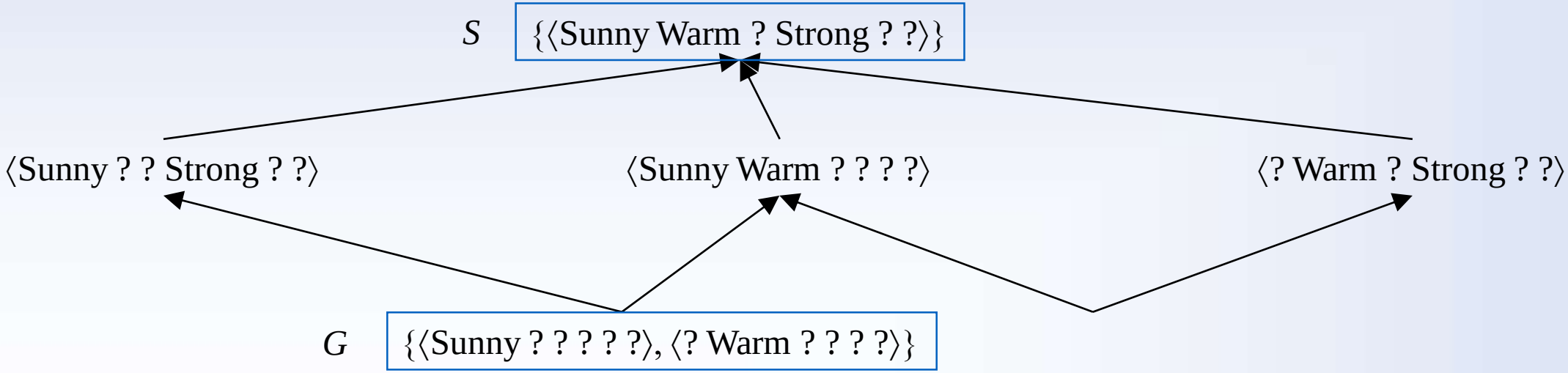


Assume learner can choose the next TE

- Should choose d such that
  - Reduces maximally the number of hypotheses in VS
  - Best TE: satisfies precisely 50% hypotheses;
    - Can't always be done
  - Example:
    - ⟨Sunny Warm Normal Weak Warm Same⟩ ?
    - If pos, generalizes S
    - If neg, specializes G

Order of examples matters for intermediate sizes of S,G; not for the final S, G

Classifying new cases using VS



- Use *voting procedure* on following examples:
  - ⟨Sunny Warm Normal Strong Cool Change⟩
  - ⟨Rainy Cool Normal Weak Warm Same⟩
  - ⟨Sunny Warm Normal Weak Warm Same⟩
  - ⟨Sunny Cold Normal Strong Warm Same⟩

Effect of incomplete hypothesis space

- Preceding algorithms work if target function is in H
  - Will generally not work if target function *not* in H
- Consider following examples which represent target function "sky = sunny or sky = cloudy":
  - ⟨Sunny Warm Normal Strong Cool Change⟩ Y
  - ⟨Cloudy Warm Normal Strong Cool Change⟩ Y
  - ⟨⟨Rainy Warm Normal Strong Cool Change⟩ N
- If apply CE algorithm as before, end up with empty VS
  - After first two TEs, S= ⟨? Warm Normal Strong Cool Change⟩
  - New hypothesis is overly general
    - it covers the third negative TE!
- Our H does not include the appropriate c

Need more expressive hypotheses

Incomplete hypothesis space

- If c not in H, then consider generalizing representation of H to contain c
  - For example, add disjunctions or negations to representation of hypotheses in H
- One way to avoid problem is to allow **all** possible representations of h's
  - Equivalent to allowing all possible subsets of instances as defining the concept of EnjoySport
    - Recall that there are 96 instances in EnjoySport; hence there are 2^{96} possible hypotheses in full space H
    - Can do this by using full propositional calculus with AND, OR, NOT
    - Hence H defined only by conjunctions of attributes is biased (containing only 973 h's)

Unbiased Learners and Inductive Bias

- BUT if have no limits on representation of hypotheses

  (i.e., full logical representation: *and, or, not*), can only learn examples…no generalization possible!
  - Say have 5 TEs {x1, x2, x3, x4, x5}, with x4, x5 negative TEs
- Apply CE algorithm
  - S will be disjunction of positive examples (S={x1 OR x2 OR x3})
  - G will be negation of disjunction of negative examples (G={*not (*x4 or x5)})
  - Need to use all instances to learn the concept!
- Cannot predict usefully:
  - TEs have unanimous vote
  - other h's have 50/50 vote!
    - For every h in H that predicts +, there is another that predicts -

Unbiased Learners and Inductive Bias

- Approach:
  - Place constraints on representation of hypotheses
    - Example of limiting connectives to conjunctions
    - Allows learning of generalized hypotheses
    - Introduces bias that depends on hypothesis representation
- Need formal definition of inductive bias of learning algorithm

Inductive Syst and Equiv Deductive Syst

- Inductive bias made explicit in *equivalent deductive system*
  - *Logically represented system that produces same outputs (classification) from inputs (TEs, instance x, bias B) as CE procedure*
- Inductive bias (IB) of learning algorithm L is any minimal set of assertions B such that for any target concept c and training examples D, we can logically infer value c(x) of any instance x from B, D, and x
  - E.g., for rote learner, B = {}, and there is no IB
- Difficult to apply in many cases, but a useful guide

Inductive Bias and specific learning algs

- Rote learners:

  no IB

- Version space candidate elimination algorithm:

  c can be represented in H

- Find-S: c can be represented in H;

  all instances that are not positive are negative

Computational Complexity of VS

- The *S* set for conjunctive feature vectors and tree-structured attributes is linear in the number of features and the number of training examples.
- The *G* set for conjunctive feature vectors and tree-structured attributes can be exponential in the number of training examples.
- In more expressive languages, both *S* and *G* can grow exponentially.
- The order in which examples are processed can significantly affect computational complexity.

Summary

- Concept learning as search through *H*
- General-to-specific ordering over *H*
- Version space candidate elimination algorithm
- *S* and *G* boundaries characterize learner's uncertainty
- Learner can generate useful queries
- Inductive leaps possible only if learner is biased!
- Inductive learners can be modeled as equiv deductive systems
- Biggest problem is inability to handle data with errors
  - Overcome with procedures for learning decision trees

Decision Tree Learning

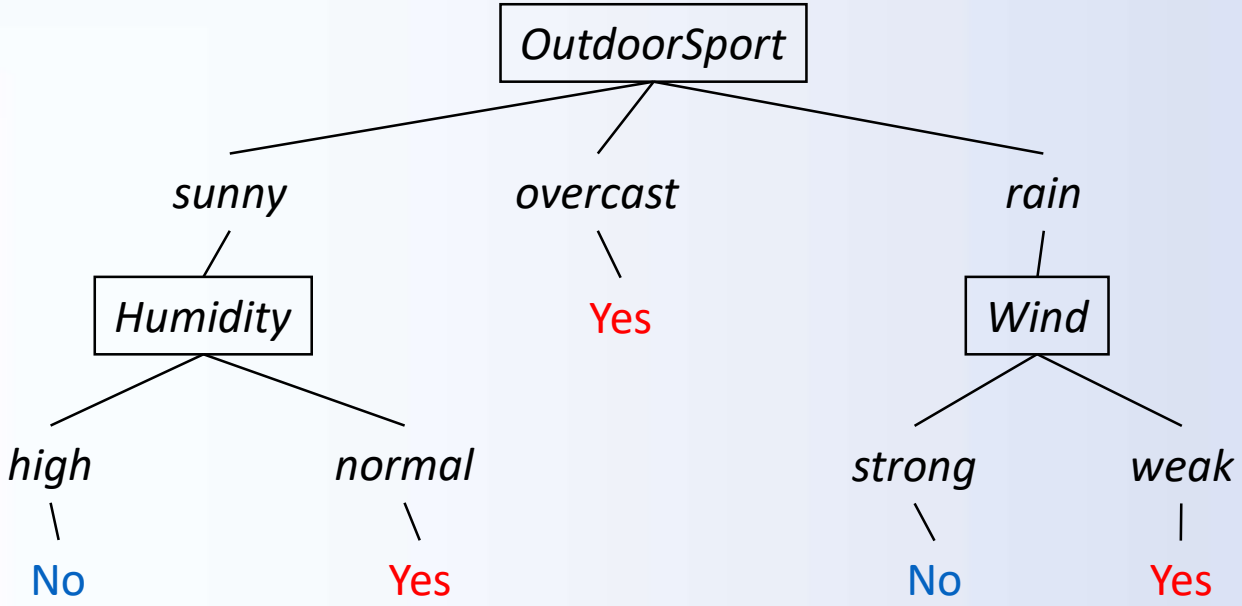
Training Examples

Day	Outlook	Temp	Humidity	Wind	Tennis?
D1	Sunny	Hot	High	Weak	*No*
D2	Sunny	Hot	High	Strong	*No*
D3	Overcast	Hot	High	Weak	*Yes*
D4	Rain	Mild	High	Weak	*Yes*
D5	Rain	Cool	Normal	Weak	*Yes*
D6	Rain	Cool	Normal	Strong	*No*
D7	Overcast	Cool	Normal	Strong	*Yes*
D8	Sunny	Mild	High	Weak	*No*
D9	Sunny	Cool	Normal	Weak	*Yes*
D10	Rain	Mild	Normal	Weak	*Yes*
D11	Sunny	Mild	Normal	Strong	*Yes*
D12	Overcast	Mild	High	Strong	*Yes*
D13	Overcast	Hot	Normal	Weak	*Yes*
D14	Rain	Mild	High	Strong	*No*

Representation of Concepts

- Concept learning: conjunction of attributes
  - (Sunny AND Hot AND Humid AND Windy) +

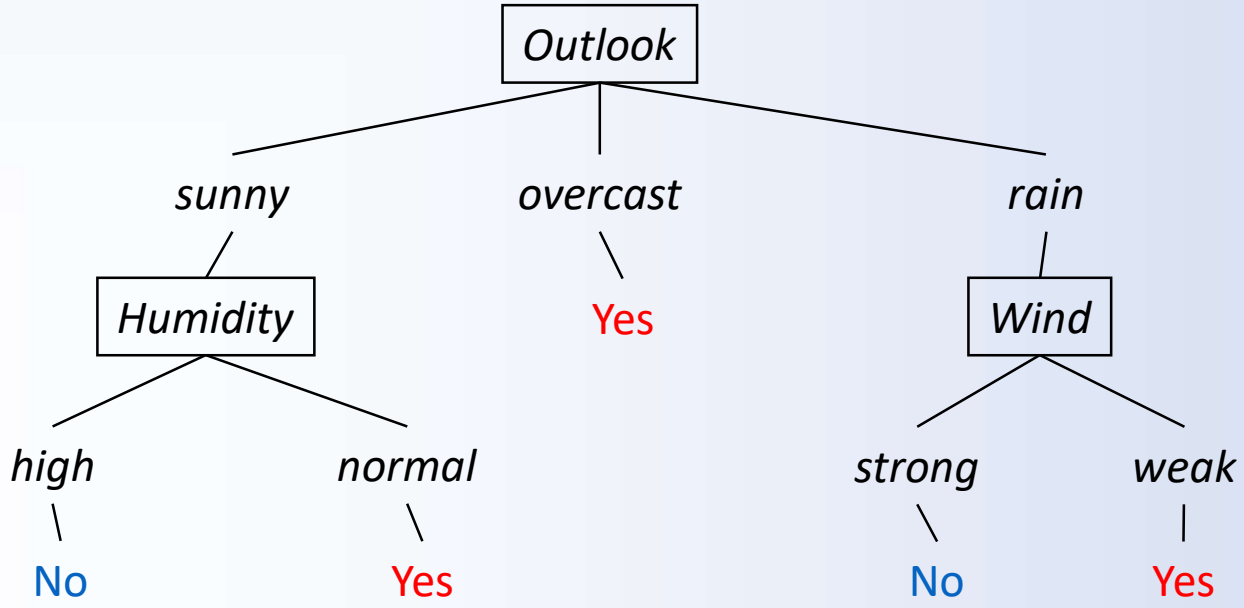
- Decision trees: disjunction of conjunction of attributes
  - (Sunny AND Normal) OR (Overcast) OR (Rain AND Weak) +
  - More powerful representation
  - Larger hypothesis space H
  - Can be represented as a tree
  - Common form of decision making in humans



Decision Trees

- Decision tree to represent learned target functions
  - Each internal node tests an attribute
  - Each branch corresponds to attribute value
  - Each leaf node assigns a classification

- Can be represented by logical formulas



Representation in decision trees

- Example of representing rule in DT's:

  if outlook = sunny AND humidity = normal

  OR

  if outlook = overcast

  OR

  if outlook = rain AND wind = weak

  then playtennis

Applications of Decision Trees

- Instances describable by a fixed set of attributes and their values
- Target function is discrete valued
  - 2-valued
  - N-valued
  - But can approximate continuous functions
- Disjunctive hypothesis space
- Possibly noisy training data
  - Errors, missing values, …
- Examples:
  - Equipment or medical diagnosis
  - Credit risk analysis
  - Calendar scheduling preferences

Top-Down Construction

- Start with empty tree
- Main loop:
  1. Split the "best" decision attribute (*A*) for next node
  2. Assign *A* as decision attribute for node
  3. For each value of *A*, create new descendant of node
  4. Sort training examples to leaf nodes
  5. If training examples perfectly classified, STOP,
     Else iterate over new leaf nodes
- Grow tree just deep enough for perfect classification
  - If possible (or can approximate at chosen depth)
- Which attribute is best?

Entropy

- A measure for
  - uncertainty
  - purity
  - information content
- S is a sample of training examples
  - p_+ is the proportion of positive examples in S
  - p_- is the proportion of negative examples in S
- Entropy of S: average optimal number of bits to encode information about certainty/uncertainty about S

$$Entropy(S) = p_+(-\log_2 p_+) + p_-(-\log_2 p_-) = -p_+\log_2 p_+ - p_-\log_2 p_-$$

- Can be generalized to more than two values

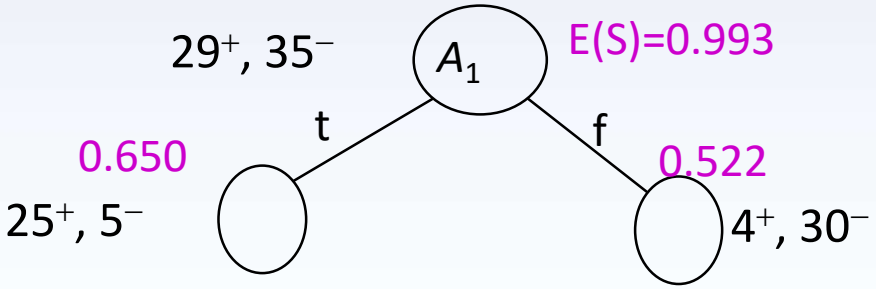
Entropy

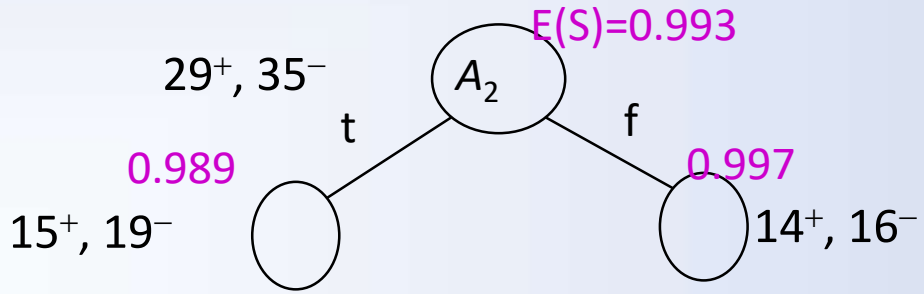
- Entropy can also be viewed as measuring
  - purity of S,
  - uncertainty in S,
  - information in S, …
- E.g.: values of entropy for p+=1, p+=0, p+=.5

$$Entropy(S) \equiv \sum_{i=1}^{c} -p_i \log_2 p_i$$

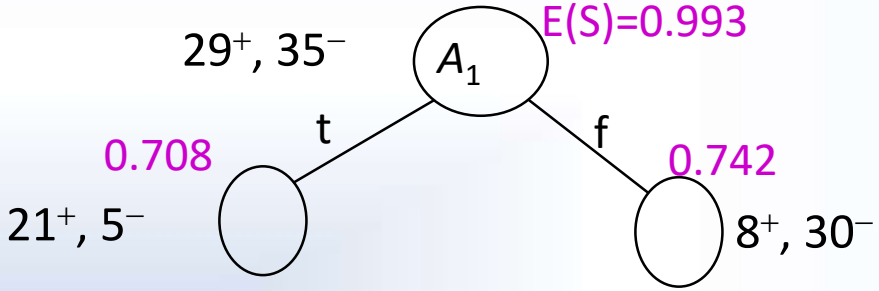
Choosing Best Attribute?

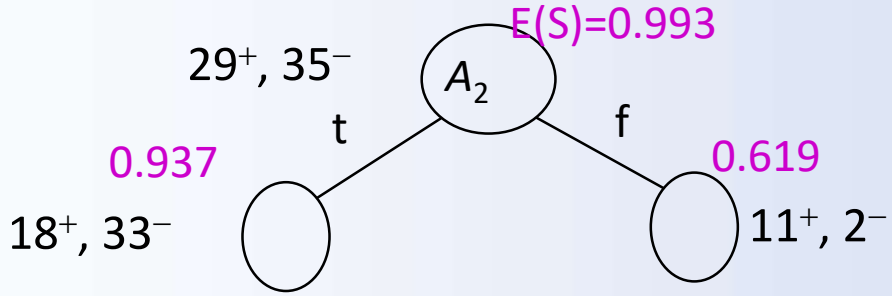
- Consider 64 examples $(29^+, 35^-)$ and compute entropies:
- Which one is better?





- Which is better?

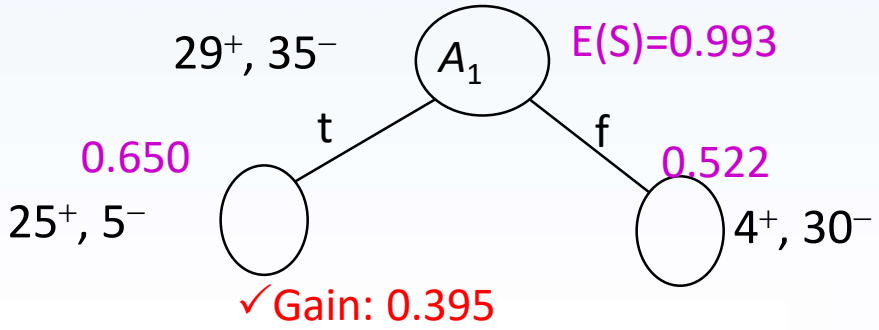


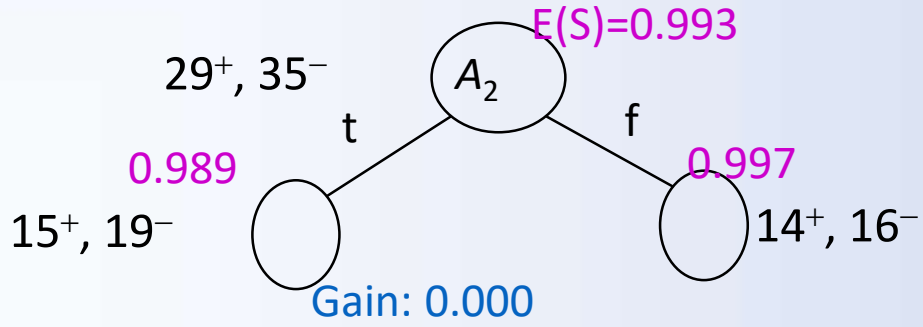


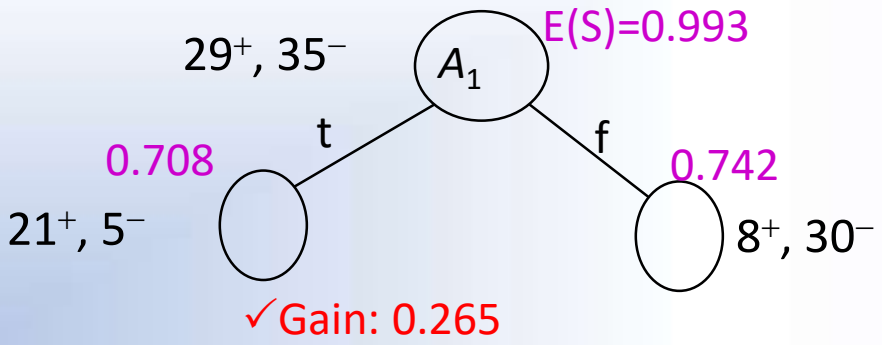
Information Gain

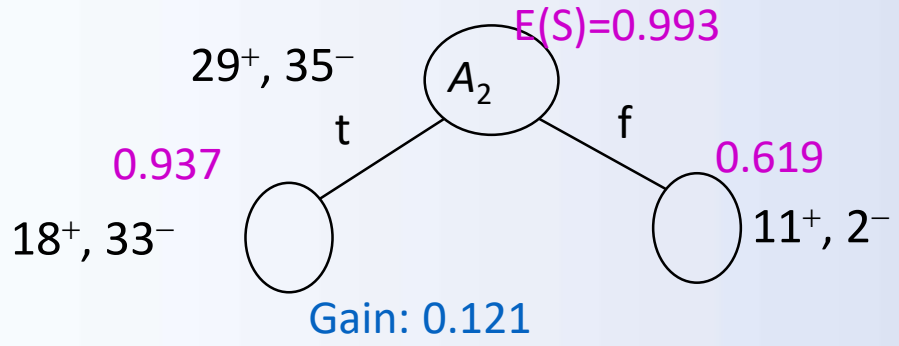
- $Gain(S,A)$: reduction in entropy after choosing attr. A

$$Gain(S,A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$









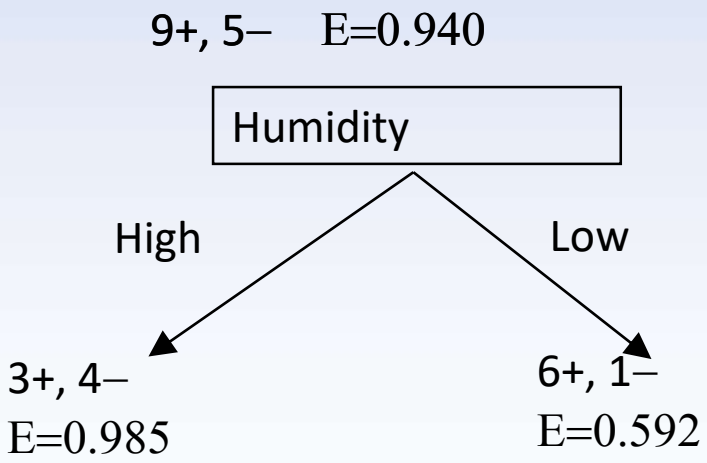
Gain function

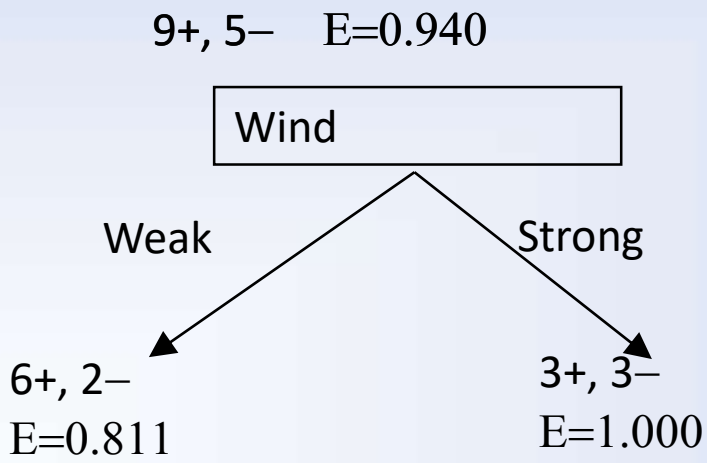
- Gain is measure of how much can
  - Reduce uncertainty
    - Value lies between 0,1
    - What is significance of
      - gain of 0?
        - example where have 50/50 split of +/- both before *and* after discriminating on attributes values
      - gain of 1?
        - Example of going from “perfect uncertainty” to perfect certainty after splitting example with predictive attribute
  - Find “patterns” in TE’s relating to attribute values
    - Move to locally minimal representation of TE’s

Training Examples

Day	Outlook	Temp	Humidity	Wind	Tennis?
D1	Sunny	Hot	High	Weak	*No*
D2	Sunny	Hot	High	Strong	*No*
D3	Overcast	Hot	High	Weak	*Yes*
D4	Rain	Mild	High	Weak	*Yes*
D5	Rain	Cool	Normal	Weak	*Yes*
D6	Rain	Cool	Normal	Strong	*No*
D7	Overcast	Cool	Normal	Strong	*Yes*
D8	Sunny	Mild	High	Weak	*No*
D9	Sunny	Cool	Normal	Weak	*Yes*
D10	Rain	Mild	Normal	Weak	*Yes*
D11	Sunny	Mild	Normal	Strong	*Yes*
D12	Overcast	Mild	High	Strong	*Yes*
D13	Overcast	Hot	Normal	Weak	*Yes*
D14	Rain	Mild	High	Strong	*No*

Determine the Root Attribute





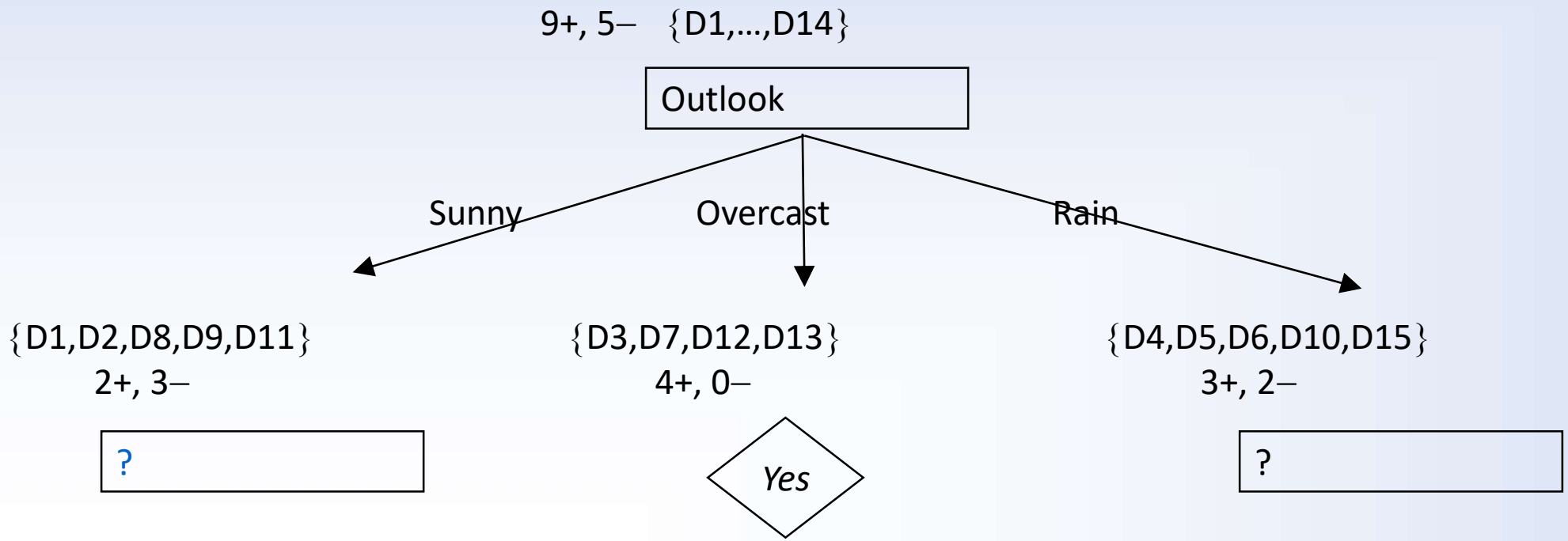
Gain (S, Humidity) = 0.151

Gain (S, Wind) = 0.048

Gain (S, Outlook) = 0.246

Gain (S, Temp) = 0.029

Hypothesis Space Search in Decision Tree Learning



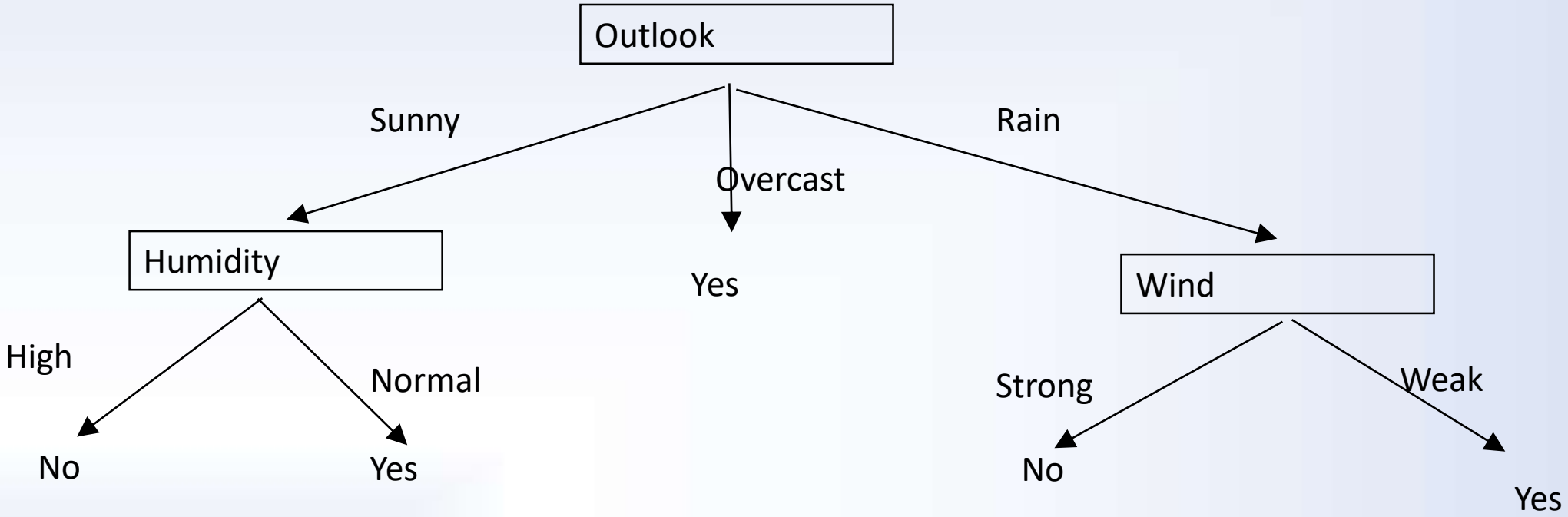
S_{sunny}= {D1,D2,D8,D9,D11}

Gain (S_{sunny}, Humidity) = .970

Gain (S_{sunny}, Temp) = .570

Gain (S_{sunny}, Wind) = .019

Final Decision Tree for Example



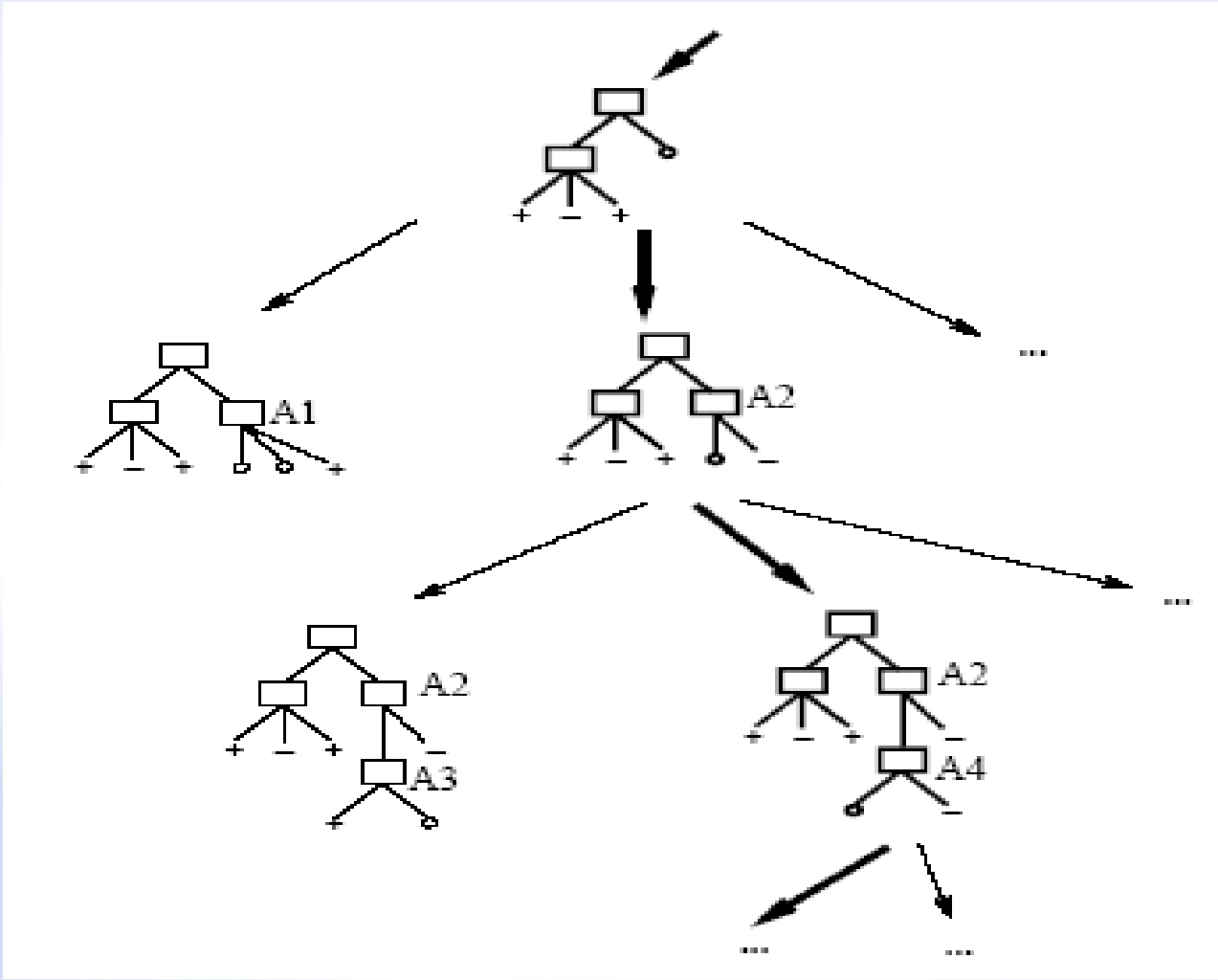
Hypothesis Space Search in Decision Trees

- ID3 can be characterized as searching a space of hypotheses for one that fits the training examples.
- ID3 performs a simple to complex, hill-climbing search through this hypothesis space, beginning with the empty tree, then considering progressively more elaborate hypotheses in search of a decision tree that correctly classifies the training data.

- Capabilities and limitations By viewing ID3 in terms of its search space and search strategy are
  - ID3 hypothesis space of all decision trees is a ***complete space of finite*** discrete-valued functions, relative to the available attributes.
  - Because every finite discrete-valued function can be represented by some decision tree, ID3 avoids one of the major risks of methods that search incomplete hypothesis spaces that the hypothesis space might not contain the target function.

Hypothesis Space Search (ID3)

- Hypothesis space (all possible trees) is complete!
  - Target fun



- ID3 maintains only a single current hypothesis as it searches through the space of decision trees. By determining only a single hypothesis, ID3 loses the capabilities that follow from explicitly representing all consistent hypotheses.
- ID3 in its pure form performs no backtracking in its search. Once it, selects an attribute to test at a particular level in the tree, it never backtracks to reconsider this choice. It is susceptible to the usual risks of hill-climbing search without backtracking converging to locally optimal solutions that are not globally optimal

- ID3 uses all training examples at each step in the search to make statistically based decisions regarding how to refine its current hypothesis. Advantage of using statistical properties of all the examples (e.g., information gain) is that the resulting search is much less sensitive to errors in individual training examples.

Inductive Bias in ID3

- Restriction Biases and Preference Biases: There is difference between the types of inductive bias exhibited by ID3 and by the Candidate-elimination Algorithm.
  - ID3 searches incompletely through this space, from simple to complex hypotheses, until its termination condition is met. Its inductive bias is solely a consequence of the ordering of hypotheses by its search strategy.
  - The inductive bias of ID3 is thus a preference for certain hypotheses over others. with no hard restriction on the hypotheses that can be eventually enumerated. This form of bias is typically called a *preferential bias*.

Restriction bias vs. Preference bias

- Restriction bias (or Language bias)
  - Incomplete hypothesis space
- Preference (or search) bias
  - Incomplete search strategy
- Candidate Elimination has restriction bias
- ID3 has preference bias
- In most cases, we have both a restriction and a preference bias.

Inductive Bias in DTL

- **Why Prefer Short Hypotheses?**
  - Occam's razor: Prefer the simplest hypothesis that fits the data.
  - There are fewer short hypotheses than long ones. it is less likely that one will find a short hypothesis that coincidentally fits the training data.
  - Why should we believe that the small set of hypotheses consisting of decision trees with ***short descriptions should be any more relevant than the multitude of*** other small sets of hypotheses that we might define? Is first argument against Occam's razor.
  - A second problem with the above argument for Occam's razor is that the size of a hypothesis is determined by the particular representation used ***internally by*** the learner.

- The last argument on Occam's razor will produce two different hypotheses from the same training examples when it is applied by two learners that perceive these examples in terms of different internal representations.
- Occam's razor supports the scenario that examines the question of which internal representations might arise from a process of evolution and natural selection.

Issues in DTL

- **Avoiding Over fitting the Data**
  - Reduced Error Pruning
  - Rule Post-pruning
- **Incorporating Continuous-Valued Attributes**
- **Alternative Measures for Selecting Attributes**
- **Handling Training Examples with Missing Attribute Values**
- **Handling Attributes with Differing Costs**

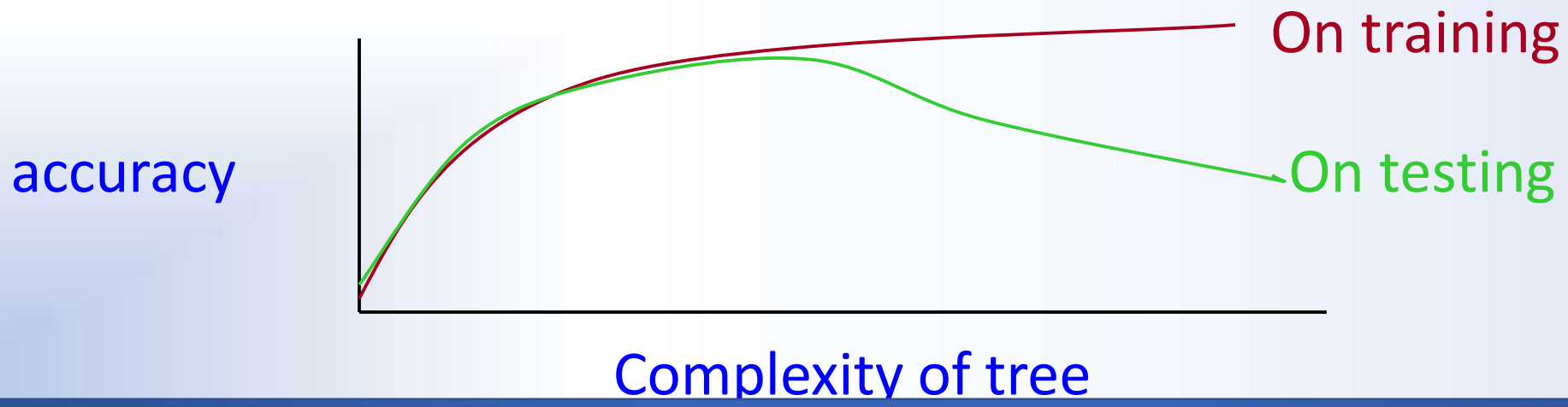
1. Avoiding Over fitting the Data

Definition: Given a hypothesis space H, a hypothesis h ∈ H is said to over fit the training data if there exists some alternative hypothesis *h' ∈ H, such that h has* smaller error than *h' over the training examples, but h' has a smaller error than h* over the entire distribution of instances

- Example:the ID3 algorithm is applied to the task of learning which medical patients have a form of diabetes. The horizontal axis of this plot indicates the total number of nodes in the decision tree, as the tree is being constructed. The vertical axis indicates the accuracy of predictions made by the tree. The solid line shows the accuracy of the decision tree over the training examples, whereas the broken line shows accuracy measured over an independent set of test examples

Overfitting the Data

- Learning a tree that classifies the training data perfectly may not lead to the tree with the best generalization performance.
  - There may be noise in the training data the tree is fitting
  - The algorithm might be making decisions based on very little data
- A hypothesis *h* is said to overfit the training data if the is another hypothesis, h', such that *h* has smaller error than *h'* on the training data but *h* has larger error on the test data than *h'*.

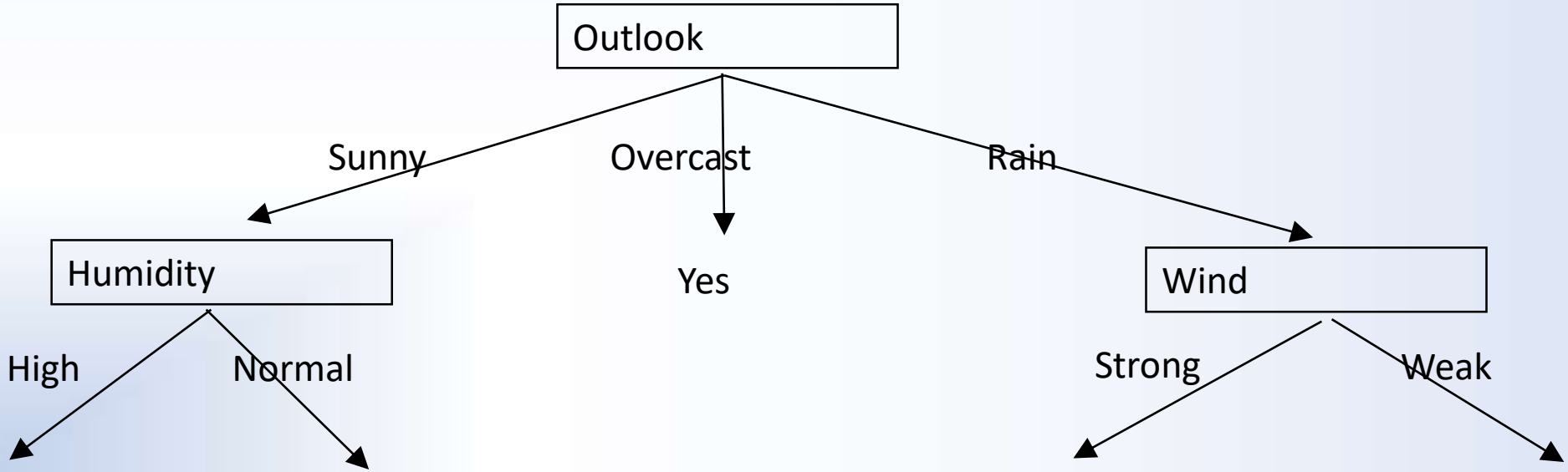


Overfitting in Decision Trees

- Consider adding *noisy* training example (should be +):

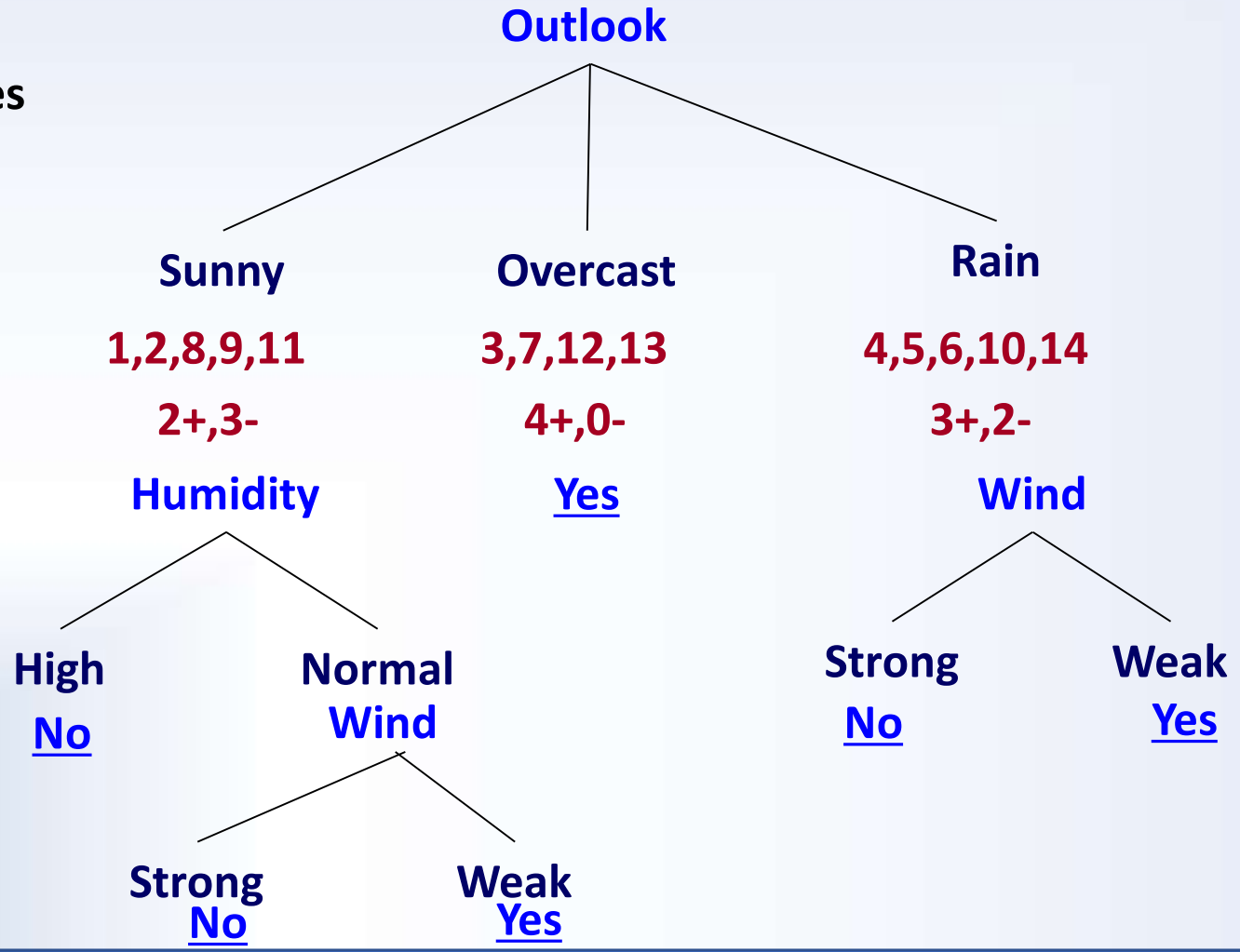
Day	Outlook	Temp	Humidity	Wind	Tennis?
D15	Sunny	Hot	Normal	Strong	*No*

- What effect on earlier tree?



Overfitting - Example

Noise or other coincidental regularities



Avoiding Overfitting

- Two basic approaches
  - Prepruning: Stop growing the tree at some point during construction when it is determined that there is not enough data to make reliable choices.
  - Postpruning: Grow the full tree and then remove nodes that seem not to have sufficient evidence. (more popular)

Reduced-Error Pruning

- A post-pruning, cross validation approach
  - Partition training data into "grow" set and "validation" set.
  - Build a complete tree for the "grow" data
  - Until accuracy on validation set decreases, do:
    For each non-leaf node in the tree
    Temporarily prune the tree below; replace it by majority vote.
    Test the accuracy of the hypothesis on the validation set
    Permanently prune the node with the greatest increase
    in accuracy on the validation test.
- Problem: Uses less data to construct the tree
- Sometimes done at the rules level

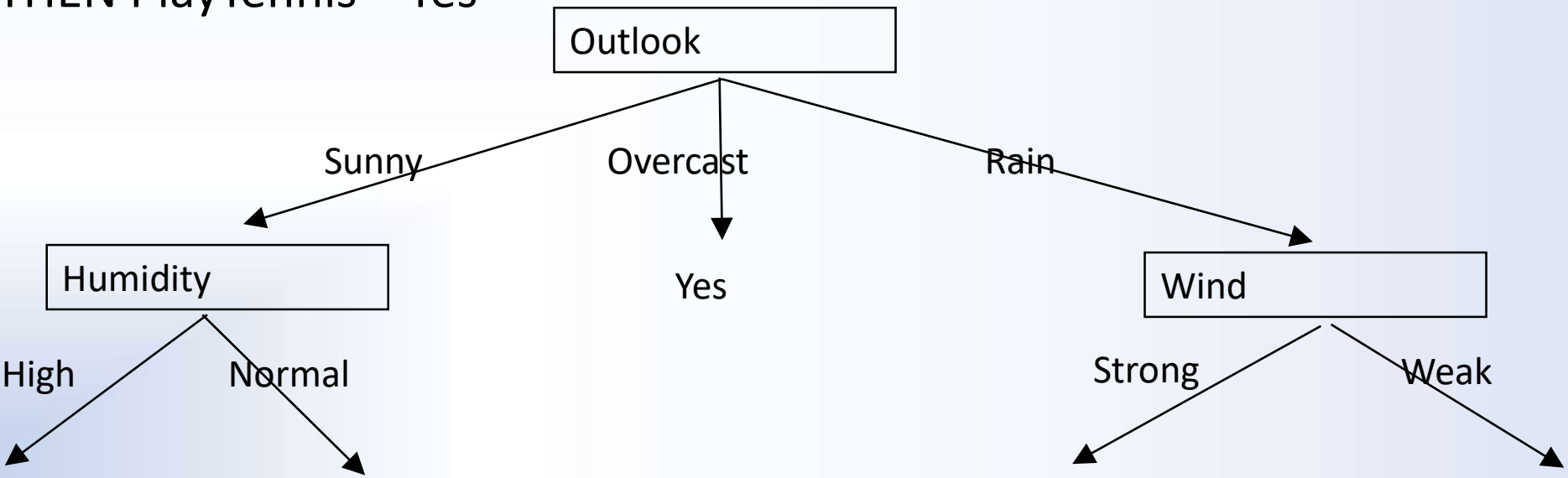
General Strategy: Overfit and Simplify

Rule post-pruning

- Allow tree to grow until best fit (allow overfitting)
- Convert tree to equivalent set of rules
  - One rule per leaf node
  - Prune each rule independently of others
    - Remove various preconditions to improve performance
  - Sort final rules into desired sequence for use

Example of rule post pruning

- IF (Outlook = Sunny) ^ (Humidity = High)
  - THEN PlayTennis = No
- IF (Outlook = Sunny) ^ (Humidity = Normal)
  - THEN PlayTennis = Yes



Extensions of basic algorithm

- Continuous valued attributes
- Attributes with many values
- TE's with missing data
- Attributes with associated costs
- Other impurity measures
- Regression tree

Continuous Valued Attributes

- Create a discrete attribute from continuous variables
  - E.g., define critical Temperature = 82.5
- Candidate thresholds
  - chosen by gain function
  - can have more than one threshold
  - typically where values change quickly

(48+60)/2

(80+90)/2

Temp	40	48	60	72	80	90
Tennis?	N	N	Y	Y	Y	N

Attributes with Many Values

- Problem:
  - If attribute has many values, *Gain* will select it (why?)
  - E.g. of birthdates attribute
    - 365 possible values
    - Likely to discriminate well on small sample

Unknown Attribute Values

- What if some examples are missing values of attribute *A*?
- Use training example anyway, sort through tree
  - if node *n* tests *A*, assign most common value of *A* among other examples sorted to node *n*
  - assign most common value of *A* among other examples with same target value
  - assign probability p_i to each possible value v_i of *A*
    - assign fraction p_i of example to each descendant in tree
- Classify test instances with missing values in same fashion
- Used in C4.5

When Are Decision Trees Useful ?

- Advantages
  - Very fast: can handle very large datasets with many attributes
  - Flexible: several attribute types, classification and regression problems, missing values…
  - Interpretability: provide rules and attribute importance
- Disadvantages
  - Instability of the trees (high variance)
  - Not always competitive with other algorithms in terms of accuracy

History of Decision Tree Research

- Hunt and colleagues in Psychology used full search decision trees methods to model human concept learning in the 60's
- Quinlan developed ID3, with the information gain heuristics in the late 70's to learn expert systems from examples
- Breiman, Friedmans and colleagues in statistics developed CART (classification and regression trees simultaneously
- A variety of improvements in the 80's: coping with noise, continuous attributes, missing data, non-axis parallel etc.
- Quinlan's updated algorithm, C4.5 (1993) is commonly used (New:C5)
- Boosting (or Bagging) over DTs is a good general purpose algorithm

Summary

- Decision trees are practical for concept learning
- Basic information measure and gain function for best first search of space of DTs
- ID3 procedure
  - search space is complete
  - Preference for shorter trees
- Overfitting is an important issue with various solutions
- Many variations and extensions possible

Software

- In R:
  - Packages tree and rpart
- C4.5:
  - http://www.cse.unwe.edu.au/~quinlan
- Weka
  - http://www.cs.waikato.ac.nz/ml/weka