

# Intro. to machine learning (CSI 5325)

## Lecture 2: concept learning

Greg Hamerly

Some content from Tom Mitchell.

- 1 Course administration
- 2 Concept learning
- 3 Learning from examples
- 4 General-to-specific ordering over hypotheses
- 5 Version spaces and candidate elimination algorithm

# Course planning

- Who wants to do the first paper?

# The definition of concept learning

Concept learning is learning a function which has a boolean-valued output.

$$f : X \rightarrow \{0, 1\}$$

Many machine learning approaches use this simplistic binary view of the world.

- aside: multiclass  $\rightarrow$  binary class reductions

# Training examples for EnjoySport

| Sky   | Temp | Humid  | Wind   | Water | Forecast | <b>EnjoySport</b> |
|-------|------|--------|--------|-------|----------|-------------------|
| Sunny | Warm | Normal | Strong | Warm  | Same     | <b>Yes</b>        |
| Sunny | Warm | High   | Strong | Warm  | Same     | <b>Yes</b>        |
| Rainy | Cold | High   | Strong | Warm  | Change   | <b>No</b>         |
| Sunny | Warm | High   | Strong | Cool  | Change   | <b>Yes</b>        |

What is the general concept?

## Representing Hypotheses

Many possible hypothesis representations...

Let's assume hypothesis  $h$  is a **conjunction of constraints on attribute values**.

Each constraint can be

- a specific value (e.g., "*Water = Warm*")
- don't care (e.g., "*Water = ?*")
- no value allowed (e.g., "*Water = ∅*")

For example, the hypothesis might be:

| Sky           | AirTemp  | Humid    | Wind          | Water    | Forecast     |
|---------------|----------|----------|---------------|----------|--------------|
| <i>⟨Sunny</i> | <i>?</i> | <i>?</i> | <i>Strong</i> | <i>?</i> | <i>Same⟩</i> |

# Prototypical Concept Learning Task

## Given:

- Instances  $X$ : Possible days, each described by the attributes *Sky*, *AirTemp*, *Humidity*, *Wind*, *Water*, *Forecast*
- Target function  $c$ :  $EnjoySport : X \rightarrow \{0, 1\}$
- Hypotheses  $H$ : Conjunctions of literals. E.g.

$\langle ?, Cold, High, ?, ?, ? \rangle$ .

- Training examples  $D$ : Positive and negative examples of the target function:

$\langle x_1, c(x_1) \rangle, \dots, \langle x_m, c(x_m) \rangle$

**Determine:** Hypothesis  $h \in H$  where  $h(x) = c(x)$  for all  $x \in D$ .

# The inductive learning hypothesis

**The 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.

# Noiseless learning

Today, assume there are no mistakes (aka noise) in the examples.

I.e., all attribute values and labels are correct for every example, and we expect a 'good' hypothesis to be correct on every training example.

We would like learning algorithms to be robust to noise (why?).

But with noise, we should expect even a 'good' hypothesis to make some 'mistakes' on the training examples (why?).

## Searching over hypotheses

For the *EnjoySport* problem, we have chosen a class of hypotheses.

- How should we choose one particular hypothesis?

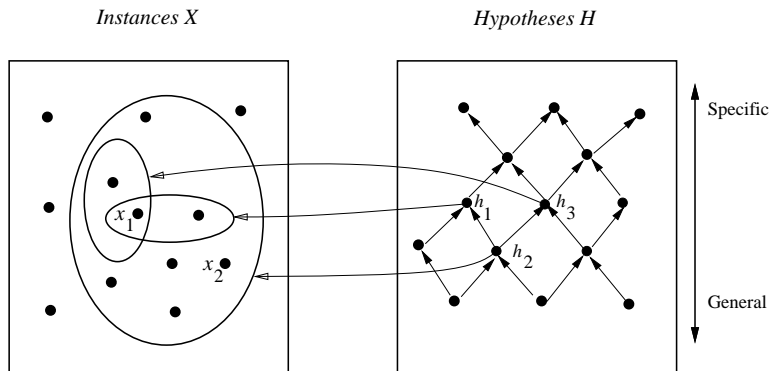
We could explicitly try all possible hypotheses. . .

- for each hypothesis, check that it 'fits' every training example

It would be more efficient if we could structure our search. . .

- The hypothesis space we've chosen has an inherent order: general-to-specific.

## Instance, hypothesis, and more-general-than relation


 $x_1 = \langle \text{Sunny, Warm, High, Strong, Cool, Same} \rangle$ 
 $x_2 = \langle \text{Sunny, Warm, High, Light, Warm, Same} \rangle$ 
 $h_1 = \langle \text{Sunny, ?, ?, Strong, ?, ?} \rangle$ 
 $h_2 = \langle \text{Sunny, ?, ?, ?, ?, ?} \rangle$ 
 $h_3 = \langle \text{Sunny, ?, ?, ?, Cool, ?} \rangle$

## 'More-general' relations

Let  $h_j$  and  $h_k$  be boolean-valued functions defined over  $X$ .

- here  $X$  is the set of possible examples

More-general-than-or-equal-to:

$$(h_j \geq_g h_k) \leftrightarrow (\forall x \in X)[(h_k(x) = 1) \rightarrow (h_j(x) = 1)]$$

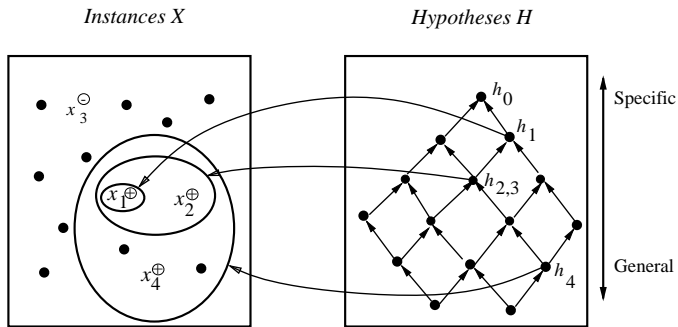
Strictly more-general-than:

$$(h_j >_g h_k) \leftrightarrow (h_j \geq_g h_k) \wedge (h_k \not\geq_g h_j)$$

# Find-S Algorithm

- 1 Initialize  $h$  to the most specific hypothesis in  $H$
- 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  $x$
    - Then do nothing
    - Else replace  $a_i$  in  $h$  by the next more general constraint that is satisfied by  $x$
- 3 Output hypothesis  $h$

## Hypothesis Space Search by Find-S



$x_1 = \langle \text{Sunny Warm Normal Strong Warm Same} \rangle, +$   
 $x_2 = \langle \text{Sunny Warm High Strong Warm Same} \rangle, +$   
 $x_3 = \langle \text{Rainy Cold High Strong Warm Change} \rangle, -$   
 $x_4 = \langle \text{Sunny Warm High Strong Cool Change} \rangle, +$

$h_0 = \langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$

$h_1 = \langle \text{Sunny Warm Normal Strong Warm Same} \rangle$

$h_2 = \langle \text{Sunny Warm ? Strong Warm Same} \rangle$

$h_3 = \langle \text{Sunny Warm ? Strong Warm Same} \rangle$

$h_4 = \langle \text{Sunny Warm ? Strong ? ?} \rangle$

## Complaints about Find-S

- Can't tell whether it has learned concept
- Can't tell when training data is inconsistent
- Picks a maximally specific  $h$
- Depending on the hypothesis space  $H$ , there might be several!

Why?

## General-to-specific?

Discuss: could we move **from** the most general hypothesis **towards** more specific ones?

# Version Spaces

To enable a more complete search of  $H$ , we turn to version spaces.

A version space defines **all** the hypotheses which fit the training data.

This allows a more complete picture of the approximation to the target concept than does looking for a single hypothesis.

## Some definitions

A hypothesis  $h$  is **consistent** with a set of training examples  $D$  of target concept  $c$  if and only if  $h(x) = c(x)$  for each training example  $\langle x, c(x) \rangle$  in  $D$ .

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

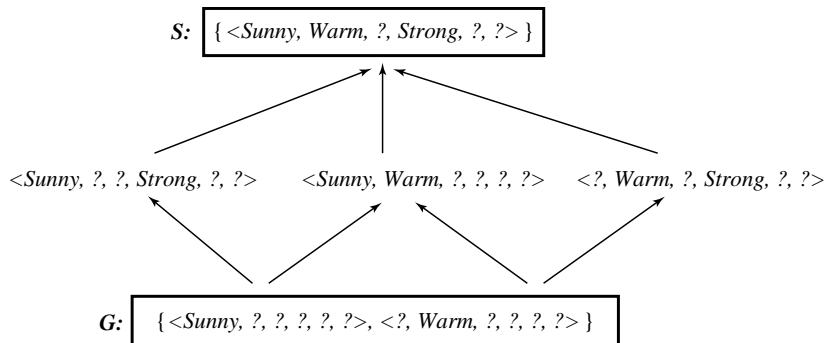
The **version space**,  $VS_{H,D}$ , with respect to hypothesis space  $H$  and training examples  $D$ , is the subset of hypotheses from  $H$  consistent with all training examples in  $D$ .

$$VS_{H,D} \equiv \{h \in H \mid \text{Consistent}(h, D)\}$$

## The List-Then-Eliminate Algorithm:

- 1  $VersionSpace \leftarrow$  a list containing every hypothesis 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  $VersionSpace$

# Example Version Space



## Representing Version Spaces

The **General boundary**,  $G$ , of version space  $VS_{H,D}$  is the set of its maximally general members

The **Specific boundary**,  $S$ , of version space  $VS_{H,D}$  is the set of its maximally specific members

Every member of the version space lies between these boundaries

$$VS_{H,D} = \{h \in H \mid (\exists s \in S)(\exists g \in G)(g \geq_g h \geq_g s)\}$$

Thus, we only need to define  $G$  and  $S$ .

# Candidate Elimination Algorithm

Initialize:

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

For each training example  $d$ , do

- If  $d$  is a positive example, adjust sets  $G$  and  $S$
- If  $d$  is a negative example, adjust sets  $G$  and  $S$

Each adjustment makes the hypotheses in  $G$  more specific, or hypotheses in  $S$  more general.

(This algorithm still doesn't work well with noisy data.)

## Candidate Elimination Algorithm – positive example

For a positive example  $d$ :

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

## Candidate Elimination Algorithm – negative example

For a negative example  $d$ :

- Remove from  $S$  any hypothesis inconsistent with  $d$
- For each hypothesis  $g$  in  $G$  inconsistent with  $d$ 
  - Remove  $g$  from  $G$
  - Add to  $G$  all minimal specializations  $h$  of  $g$  such that
    - $h$  is consistent with  $d$ , and
    - some member of  $S$  is more specific than  $h$
  - Remove from  $G$  any hypothesis that is less general than another hypothesis in  $G$