

Machine Learning-II

Module-1

Introduction: Well-Posed Learning Problems, Designing a Learning System, Perspectives and Issues in Machine Learning. Concept Learning and the General-to-Specific Ordering: 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

INTRODUCTION

Ever since computers were invented, we have wondered whether they might be made to learn. If we could understand how to program them to learn-to improve automatically with experience-the impact would be dramatic.

- Imagine computers learning from medical records which treatments are most effective for new diseases
- Houses learning from experience to optimize energy costs based on the particular usage patterns of their occupants.
- Personal software assistants learning the evolving interests of their users in order to highlight especially relevant stories from the online morning newspaper

A successful understanding of how to make computers learn would open up many new uses of computers and new levels of competence and customization

Some successful applications of machine learning

- Learning to recognize spoken words
- Learning to drive an autonomous vehicle
- Learning to classify new astronomical structures
- Learning to play world-class backgammon

Why is Machine Learning Important?

- Some tasks cannot be defined well, except by examples (e.g., recognizing people).
 - Relationships and correlations can be hidden within large amounts of data. Machine Learning/Data Mining may be able to find these relationships.
 - Human designers often produce machines that do not work as well as desired in the environments in which they are used.
 - The amount of knowledge available about certain tasks might be too large for explicit encoding by humans (e.g., medical diagnostic).
 - Environments change over time.
 - New knowledge about tasks is constantly being discovered by humans.
-

1.1 WELL-POSED LEARNING PROBLEMS:

Definition: A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

To have a well-defined learning problem, three features need to be identified:

1. The class of tasks
2. The measure of performance to be improved
3. The source of experience

Examples

1. **Checkers game:** A computer program that learns to play **checkers** might improve its performance as measured by its ability to win at the class of tasks involving playing checkers games, through experience obtained by playing games against itself.

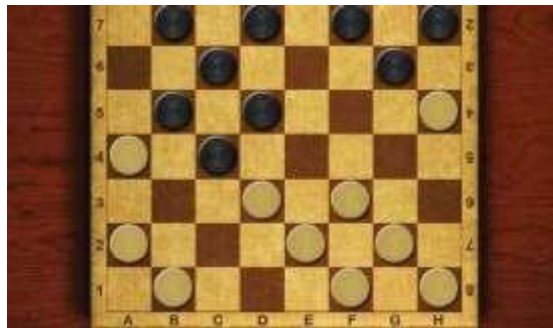


Fig: Checker game board

A checkers learning problem:

- Task T: playing checkers
- Performance measure P: percent of games won against opponents
- Training experience E: playing practice games against itself

2. A handwriting recognition learning problem:

- Task T: recognizing and classifying handwritten words within images
- Performance measure P: percent of words correctly classified
- Training experience E: a database of handwritten words with given classifications

3. A robot driving learning problem:

- Task T: driving on public four-lane highways using vision sensors
- Performance measure P: average distance travelled before an error (as judged by human overseer)
- Training experience E: a sequence of images and steering commands recorded while observing a human driver

1.2 DESIGNING A LEARNING SYSTEM:

The basic design issues and approaches to machine learning are illustrated by designing a program to learn to play checkers, with the goal of entering it in the world checkers tournament

1. Choosing the 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. The Final Design

1. *Choosing the Training Experience*

- The first design choice is to choose the type of training experience from which the system will learn.
- The type of training experience available can have a significant impact on success or failure of the learner.

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.

For example, in checkers game:

In learning to play checkers, the system might learn from ***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. The information about the correctness of specific moves early in the game must be inferred indirectly from the fact that the game was eventually won or lost.

Here the learner faces an additional problem of ***credit assignment***, or determining the degree to which each move in the sequence deserves credit or blame for the final outcome. Credit assignment can be a particularly difficult problem because the game can be lost even when early moves are optimal, if these are followed later by poor moves. Hence, learning from direct training feedback is typically easier than learning from indirect feedback.

2. The degree to which the *learner controls the sequence of training examples*

For example, in checkers game:

The learner might depend 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. 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.

If its training experience E consists only of games played against itself, there is a danger that this training experience might not be fully representative of the distribution of situations over which it will later be tested.

It is necessary to learn from a distribution of examples that is different from those on which the final system will be evaluated.

2. 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.

Let's consider a checkers-playing program that can generate the legal moves from any board state.

The program needs only to learn how to choose the best move from among these legal moves. We must learn to choose among the legal moves, the most obvious choice for the type of information to be learned is a program, or function, that chooses the best move for any given board state.

1. Let *ChooseMove* be the target function and the notation is

ChooseMove : B → M

which indicate that this function accepts as input any board from the set of legal board states B and produces as output some move from the set of legal moves M.

ChooseMove is a choice for the target function in checkers example, but this function will turn out to be very difficult to learn given the kind of indirect training experience available to our system

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

Let the target function V and the notation

$$V: B \rightarrow R$$

which denote that V maps any legal board state from the set B to some real value. Intend for this target function V to assign higher scores to better board states. If the system can successfully learn such a target function V , then it can easily use it to select the best move from any current board position.

Let us define the target value $V(b)$ for an arbitrary board state b in B , as follows:

- If b is a final board state that is won, then $V(b) = 100$
- If b is a final board state that is lost, then $V(b) = -100$
- If b is a final board state that is drawn, then $V(b) = 0$
- If b is 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

3. Choosing a Representation for the Target Function

Let's 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

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

$$\hat{V}(b) = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6$$

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

4. 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 $V_{\text{train}}(b)$ for b .

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

For instance, the following training example describes a board state b in which black has won the game (note $x_2 = 0$ indicates that red has no remaining pieces) and for which the target function value $V_{\text{train}}(b)$ is therefore $+100$.

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

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 $V_{\text{train}}(b)$ for any intermediate board state b to be $\hat{V}(\text{Successor}(b))$

Where ,

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

Rule for estimating training values

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

2. Justifying 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 best fit 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.

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

Several algorithms are known for finding weights of a linear function that minimize E . One such algorithm is called the **least mean squares, or LMS training rule**. For each observed training example it adjusts the weights a small amount in the direction that reduces the error on this training example

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

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

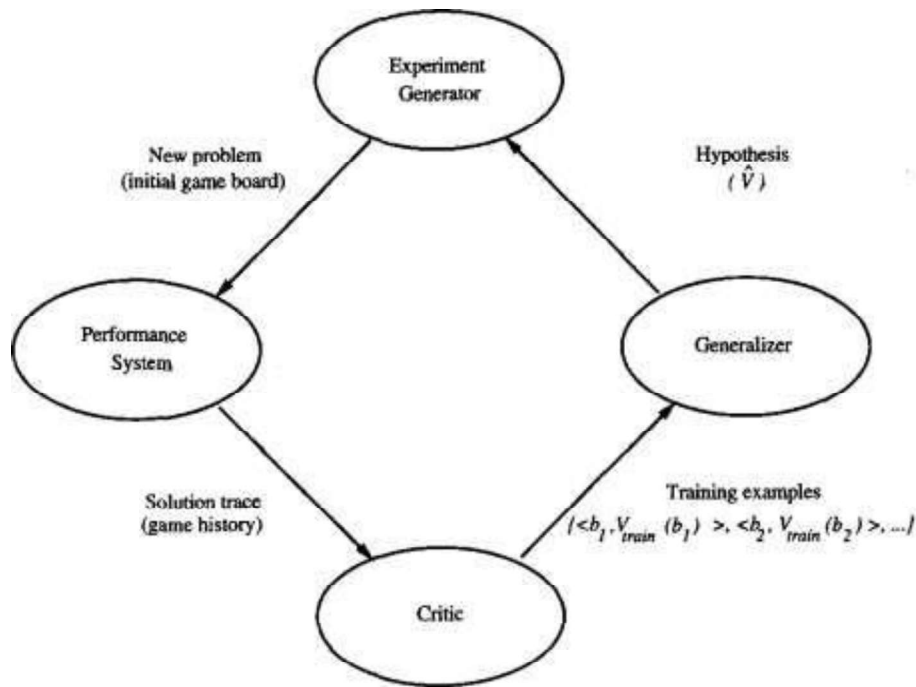
Here η is a small constant (e.g., 0.1) that moderates the size of the weight update.

Working of weight update rule

- When the error $(V_{train}(b) - \hat{V}(b))$ is zero, no weights are changed.
- When $(V_{train}(b) - \hat{V}(b))$ is positive (i.e., when $\hat{V}(b)$ is too low), then each weight is increased in proportion to the value of its corresponding feature. This will raise the value of $\hat{V}(b)$, reducing the error.
- If the value of some feature x_i is zero, then its weight is not altered regardless of the error, so that the only weights updated are those whose features actually occur on the training example board.

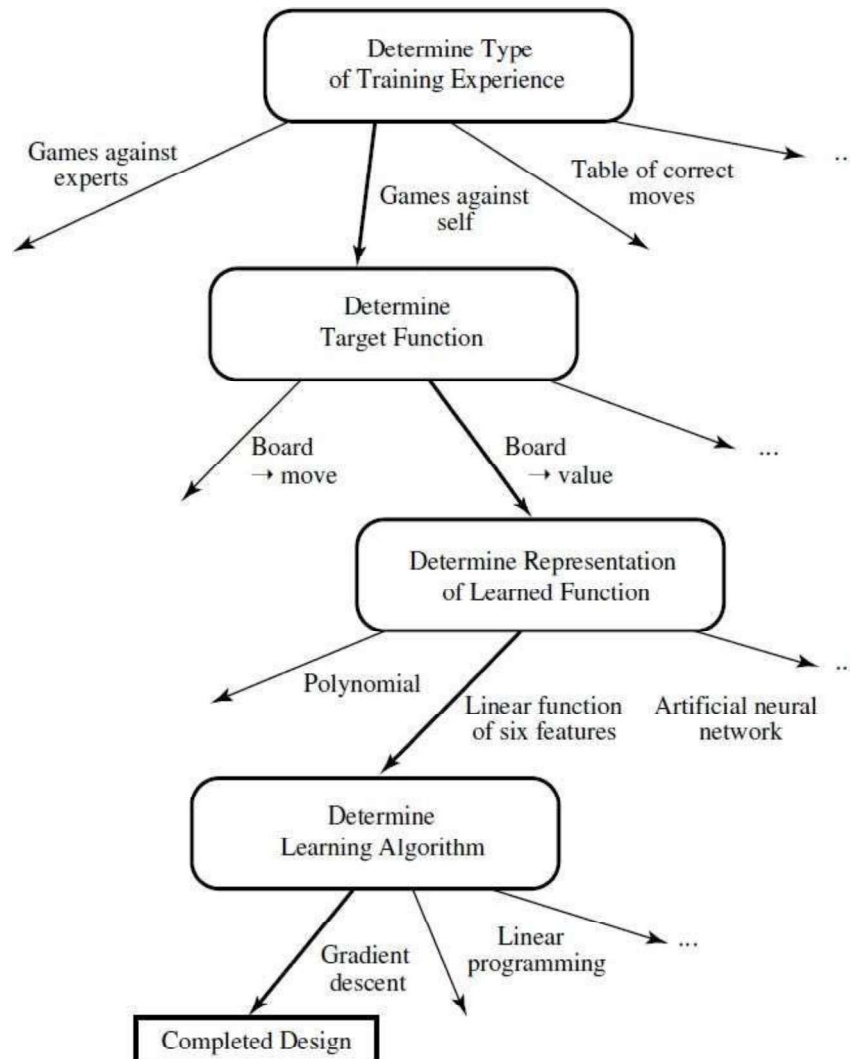
5. The Final Design

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** is the module that must solve the given performance task by using the learned target function(s). It takes an instance of a new problem (new game) as input and produces a trace of its solution (game history) as output.
2. **The Critic** takes as input the history or trace of the game and produces as output a set of training examples of the target function
3. **The Generalizer** takes as input the training examples and produces an output hypothesis that is its estimate of the target function. It generalizes from the specific training examples, hypothesizing a general function that covers these examples and other cases beyond the training examples.
4. **The Experiment Generator** takes as input the current hypothesis and outputs a new problem (i.e., initial board state) for the Performance System to explore. Its role is to pick new practice problems that will maximize the learning rate of the overall system.

The sequence of design choices made for the checkers program is summarized in below figure



1.3 PERSPECTIVES AND ISSUES IN MACHINE LEARNING:

- **One useful perspective** on machine learning is that it involves searching a very large space of possible hypotheses to determine one that best fits the observed data and any prior knowledge held by the learner.
- For example, consider the space of hypotheses that could in principle be output by the above checkers learner. This hypothesis space consists of all evaluation functions that can be represented by some choice of values for the weights w_0 through w_6 .
- The learner's task is thus to search through this vast space to locate the hypothesis that is most consistent with the available training examples.

- The LMS algorithm for fitting weights achieves this goal by iteratively tuning the weights, adding a correction to each weight each time the hypothesized evaluation function predicts a value that differs from the training value.
- This algorithm works well when the hypothesis representation considered by the learner defines a continuously parameterized space of potential hypotheses.

Issues in Machine Learning

The field of machine learning, and much of this book, is concerned with answering questions such as the following

- What algorithms exist for learning general target functions from specific training examples? In what settings will particular algorithms converge to the desired function, given sufficient training data? Which algorithms perform best for which types of problems and representations?
- How much training data is sufficient? What general bounds can be found to relate the confidence in learned hypotheses to the amount of training experience and the character of the learner's hypothesis space?
- When and how can prior knowledge held by the learner guide the process of generalizing from examples? Can prior knowledge be helpful even when it is only approximately correct?
- What is the best strategy for choosing a useful next training experience, and how does the choice of this strategy alter the complexity of the learning problem?
- What is the best way to reduce the learning task to one or more function approximation problems? Put another way, what specific functions should the system attempt to learn? Can this process itself be automated?
- How can the learner automatically alter its representation to improve its ability to represent and learn the target function?

CONCEPT LEARNING:

2.1 INTRODUCTION:

- Learning involves acquiring general concepts from specific training examples. Example: People continually learn general concepts or categories such as "bird," "car," "situations in which I should study more in order to pass the exam," etc.
- Each such concept can be viewed as describing some subset of objects or events defined over a larger set
- Alternatively, each concept can be thought of as a Boolean-valued function defined over this larger set. (Example: A function defined over all animals, whose value is true for birds and false for other animals).

Definition: Concept learning - Inferring a Boolean-valued function from training examples of its input and output

2.2 A CONCEPT LEARNING TASK:

To ground our discussion of concept learning, consider the example task of learning the target concept "days on which my friend Aldo enjoys his favorite water sport."

Table 2.1 describes a set of example days, each represented by a set of *attributes*.

The attribute **EnjoySport** indicates whether or not Aldo enjoys his favorite water sport on this day.

The task is to learn to predict the value of **EnjoySport** for an arbitrary day, based on the values of its other attributes.

Consider the example task of learning the target concept "Days on which **Aldo** enjoys his favorite water sport"

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

Table 2.1 : Positive and negative training examples for the target concept **EnjoySport**.

What hypothesis representation is provided to the learner?

- Let's consider a simple representation in which each hypothesis consists of a conjunction of constraints on the instance attributes.
- Let each hypothesis be a vector of six constraints, specifying the values of the six attributes *Sky*, *AirTemp*, *Humidity*, *Wind*, *Water*, and *Forecast*.

For each attribute, the hypothesis will either

- Indicate by a "?" that any value is acceptable for this attribute,
- Specify a single required value (e.g., Warm) for the attribute, or
- Indicate by a "Φ" that no value is acceptable

If some instance x satisfies all the constraints of hypothesis h , then h classifies x as a positive example ($h(x) = 1$).

The hypothesis that **PERSON** enjoys his favorite sport only on cold days with high humidity is represented by the expression

(?, Cold, High, ?, ?, ?)

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

(Φ, Φ, Φ, Φ, Φ, Φ)

Notation

- The set of items over which the concept is defined is called the **set of instances**, which is denoted by X .

Example: X is the set of all possible days, each represented by the attributes: Sky, AirTemp, Humidity, Wind, Water, and Forecast

- The concept or function to be learned is called the **target concept**, which is denoted by c . c can be any Boolean valued function defined over the instances X

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

Example: The target concept corresponds to the value of the attribute **EnjoySport** (i.e., $c(x) = 1$ if **EnjoySport** = Yes, and $c(x) = 0$ if **EnjoySport** = No).

- Instances for which $c(x) = 1$ are called **positive examples**, or members of the target concept.
- Instances for which $c(x) = 0$ are called **negative examples**, or non-members of the target concept.
- The ordered pair $(x, c(x))$ to describe the training example consisting of the instance x and

its target **concept value** $c(x)$.

- D to denote the set of available training examples
- The symbol H to denote the set of all possible hypotheses that the learner may consider regarding the identity of the target concept. Each hypothesis h in H represents a Boolean-valued function defined over X

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

The goal of the learner is to find a hypothesis h such that $h(x) = c(x)$ for all x in X .

- 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),
 - *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), " Φ " (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
 - Determine:
 - A hypothesis h in H such that $h(x) = c(x)$ for all x in X .
-

Table 2.2 : The *Enjoy Sport* concept learning task.

The inductive learning hypothesis:

Notice that although the learning task is to determine a hypothesis h identical to the target concept c over the entire set of instances X , the only information available about c is its value over the training examples.

Therefore, inductive learning algorithms can at best guarantee that the output hypothesis fits the target concept over the training data. Lacking any further information, our assumption is that the best hypothesis regarding unseen instances is the hypothesis that best fits the observed training data.

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.

2.3 CONCEPT LEARNING AS 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:

Consider the instances X and hypotheses H in the *Enjoy Sport* 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 = 96 distinct instances

5.4.4.4.4 = 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) = 973. Semantically distinct hypotheses

General-to-Specific Ordering of Hypotheses

Consider the two hypotheses

$h_1 = (\text{Sunny}, ?, ?, \text{Strong}, ?, ?)$

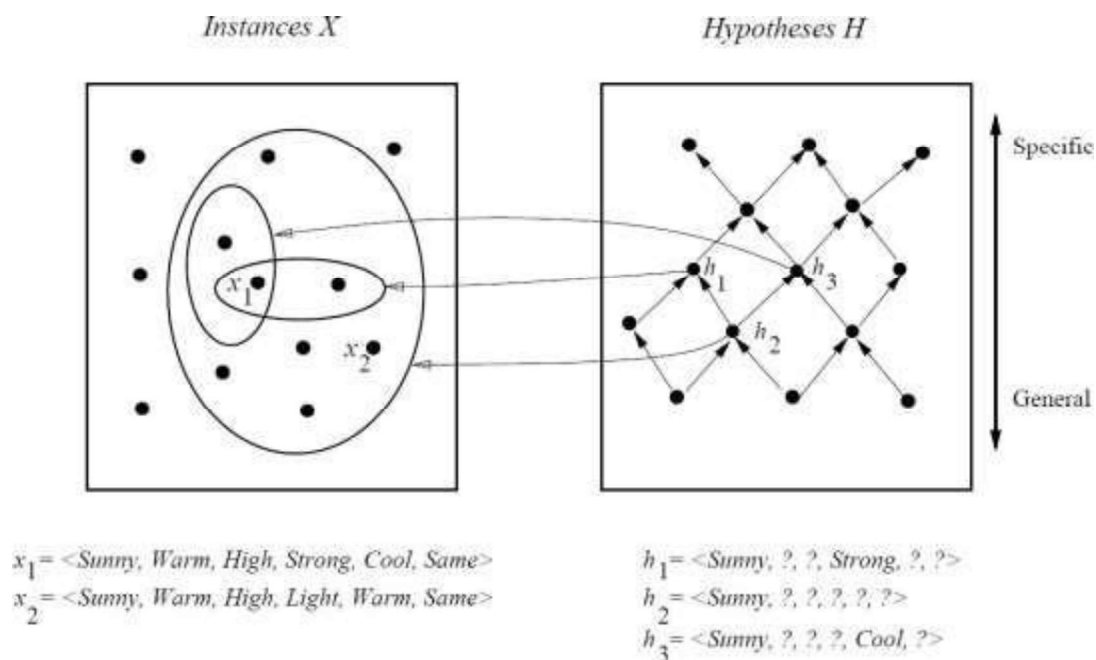
$h_2 = (\text{Sunny}, ?, ?, ?, ?, ?)$

- 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 .

Given hypotheses h_j and h_k , h_j is more-general-than or- equal do h_k if and only if any instance that satisfies h_k also satisfies h_j

Definition: Let h_j and h_k be Boolean-valued functions defined over X . Then h_j is **more general-than-or-equal-to** h_k (written $h_j \geq h_k$) if and only if

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



- In the figure, the box on the left represents the set X of all instances, the box on the right the set H of all hypotheses.
- Each hypothesis corresponds to some subset of X -the subset of instances that it classifies positive.
- The arrows connecting hypotheses represent the more - general -than relation, with the arrow pointing toward the less general hypothesis.
- Note the subset of instances characterized by h_2 subsumes the subset characterized by h_1 , hence h_2 is more - general- than h_1

2.4 FIND-S: FINDING A MAXIMALLY SPECIFIC HYPOTHESIS :

How can we use the *more-general-than* partial ordering to organize the search for a hypothesis consistent with the observed training examples?

One way is to begin with the most specific possible hypothesis in H , then generalize this hypothesis each time it fails to cover an observed positive training example.

(We say that a hypothesis "covers" a positive example if it correctly classifies the example as positive.)

To be more precise about how the partial ordering is used, consider the FIND-S algorithm defined in Table 2.3.

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 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

FIND-S algorithm Table 2.3

To illustrate this algorithm, assume the learner is given the sequence of training examples from the *EnjoySport* task

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

- The first step of FIND-S is to initialize h to the most specific hypothesis in H
 $h = (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$

- Consider the first training example
 $x_1 = \langle \text{Sunny Warm Normal Strong Warm Same} \rangle, +$

Observing the first training example, it is clear that hypothesis h is too specific. None of the " \emptyset " constraints in h are satisfied by this example, so each is replaced by the next *more general constraint* that fits the example

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

- Consider the second training example
 $x_2 = \langle \text{Sunny, Warm, High, Strong, Warm, Same} \rangle, +$

The second training example forces the algorithm to further generalize h , this time substituting a "?" in place of any attribute value in h that is not satisfied by the new example

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

- Consider the third training example
 $x_3 = \langle \text{Rainy, Cold, High, Strong, Warm, Change} \rangle, -$

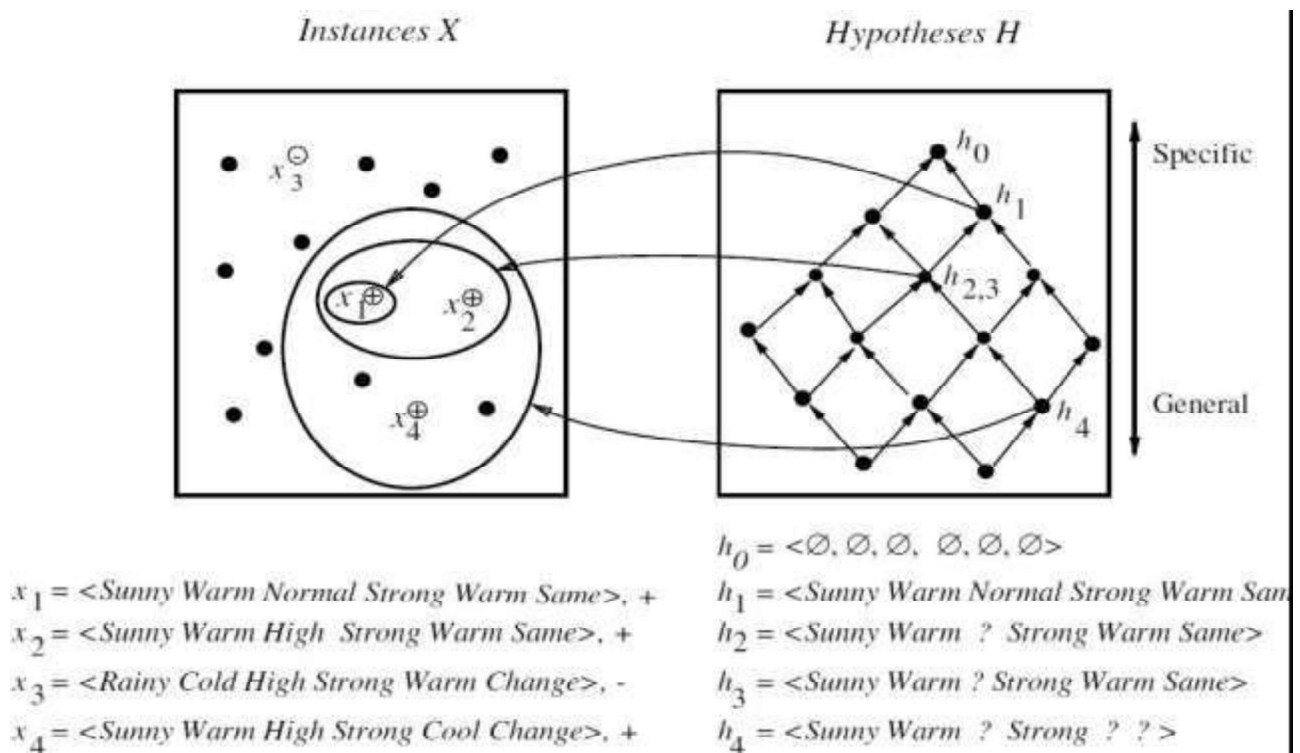
Upon encountering the third training the algorithm makes no change to h . The FIND-S algorithm simply ignores every negative example.

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

- Consider the fourth training example
 $x_4 = \langle \text{Sunny Warm High Strong Cool Change} \rangle, +$

The fourth example leads to a further generalization of h

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



The key property of the FIND-S algorithm

- FIND-S is guaranteed to output the most specific hypothesis within H that is consistent with the positive training examples
- FIND-S algorithm's final hypothesis will also be consistent with the negative examples provided the correct target concept is contained in H , and provided the training examples are correct.

Unanswered by FIND-S

- Has the learner converged to the correct target concept?
- Why prefer the most specific hypothesis?
- Are the training examples consistent?
- What if there are several maximally specific consistent hypotheses?

2.5 VERSION SPACES AND THE CANDIDATE-ELIMINATION ALGORITHM :

- This section describes a second approach to concept learning, the CANDIDATE-ELIMINATION algorithm that addresses several of the limitations of FIND-S.
- Notice that although FIND-S outputs a hypothesis from H that is consistent with the training examples, this is just one of many hypotheses from H that might fit the training data equally well.
- The key idea in the CANDIDATE-ELIMINATION algorithm is to output a description of the set of *all hypotheses consistent with the training examples*.
- Surprisingly, the CANDIDATE-ELIMINATION algorithm computes the description of this set without explicitly enumerating all of its members.
- This is accomplished by again using the *more-general-than* partial ordering, this time to maintain a compact representation of the set of consistent hypotheses and to incrementally refine this representation as each new training example is encountered.
- The CANDIDATE-ELIMINATION algorithm has been applied to problems such as learning regularities in chemical mass spectroscopy (Mitchell 1979) and learning control rules for heuristic search (Mitchell et al. 1983).
- Nevertheless, practical applications of the CANDIDATE-ELIMINATION, FIND-S algorithms are limited by the fact that they both perform poorly when given noisy training data.
- More importantly for our purposes here, the CANDIDATE-ELIMINATION algorithm provides a useful conceptual framework for introducing several fundamental issues in machine learning. In the remainder of this chapter we present the algorithm and discuss these issues. Beginning with the next chapter,
- we will examine learning algorithms that are used more frequently with noisy training data.
- The key idea in the CANDIDATE-ELIMINATION algorithm is to output a description of the set of all *hypotheses consistent with the training examples*
- One limitation of the FIND-S algorithm is that it outputs just one hypothesis consistent with the training data – there might be many.
- To overcome this, introduce notion of version space and algorithms to compute it.