Module 1

Chapter 1:
1.1: Well posed learning problems
1.2: Designing a Learning system
1.3: Perspective and Issues in Machine Learning.

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. This is machine learning.

Examples: 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.

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.***

For example, a computer program that learns to play checkers might improve its performance as ***measured by its ability to win (P)*** at the class of tasks involving ***playing checkers games (T),*** through experience ***obtained*** by ***playing games against itself (E).***

In general, to have a well-defined learning problem, we must identity these three features: the class of tasks, the measure of performance to be improved, and the source of experience.

A checkers learning proble m:

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

We can specify many learning problems in this fashion, such as learning to recognize handwritten words, or learning to drive a robotic automobile autonomously.

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

A robot driving learning problem:

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

Some successful applications of machine learning

✓ **Learning to recognize spoken words.**

Neural network learning methods and methods for learning are effective for automatically customizing to, individual speakers, vocabularies, microphone characteristics, background noise, etc. Similar techniques have potential applications in many signal- interpretation problems

✓ **Learning to drive an autonomous vehicle.**

Machine learning methods have been used to train computer- controlled vehicles to steer correctly when driving on a variety of road types. For example, the ALVINN system (Pomerleau 1989) has used its learned strategies to drive unassisted at 70 miles per hour for 90 miles on public highways among othercars.

✓ **Learning to classify ne w astronomical structures.**

Machine learning methods have been applied to a variety of large databases to learn general regularities implicit in the data. For Example:decision tree learning algorithms

✓ **Learning to play world-class backgammon.**

The most successful computer programs for playing games such as backgammon are based on machine learning algorithms.

Some disciplines and examples of their influence on machine learning.

1. Artificial intelligence: Using prior knowledge together with training data to guide learning.

2. Bayesian methods: Bayes' theorem is the basis for calculating probabilities of hypotheses. The naive Bayes classifier Algorithms for estimating values of unobserved variables.

3. Computational complexity theory: Complexity of different learning tasks, measured in terms of the computatio nal effort, number of training examples, number of mistakes, etc. required in order to learn.

4. Control theory: Procedures that learn to control processes in order to optimize predefined objectives and that learn to predict the next state of the process they are controlling.

5. Information theory: Measures of entropy and information content.

6. Philosophy: Occam's razor, suggesting that the simplest hypothesis is the best. Analysis of the justification for generalizing beyond observed data.

7. Psychology and neurobiology: Neurobiological studies motivating artificial neural network models of learning.

8. Statistics: Characterization of errors (e.g., bias and variance)and Confidence intervals, statistical tests.

DESIGNING A LEARNING SYSTEM

In order to illustrate some of the basic design issues and approaches to machine learning, let us consider designing a program to learn to play checkers, with the goal of entering it in the world checkers tournament. We adopt the obvious performance measure: the percent of games it wins in this world tournament.

Choosing the Training Expe rience

- The first design choice to choose the type of training experience from which our system will learn.
- The type of training experience available can have a significant impact on success or failure of the learner.
- One key attribute is whether the training experience provides direct or indirect feedback regarding the choices made by the performance system.
- For example, 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. Alternatively, it might have available only indirect information consisting of the move sequences and final outcomes of various games played.
- A second important attribute of the training experience is the degree to which the learner controls the sequence of training examples
- A third important attribute of the training experience is how well it represents the distribution of examples over which the final s ystem performance P must be measured.

A checkers learning proble m:

Task T: playing checkers

Performance measure P: percent of games won in the world tournament

Training experience E: games played against itself

In order to complete the design of the learning system, we must now choose

1. The exact type of knowledge to be learned
2. A representation for this target knowledge
3. A learning mechanism

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. 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.

Let us call this function ChooseMove and use the notation ChooseMove : B->M to 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. It is useful to reduce the problem of improving performance P at task T to the problem of learning some particular arget function such as Choose Move. The choice of the target function will therefore be a key design choice.

.Let us therefore 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, the n 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 (assuming the opponent plays optimally, as well).

We say that it is a nonoperational definition. The goal of learning in this case is to discover an operational description of V that is a description that can be used by the checkers-playing program to evaluate states and select moves within realistic time bounds.

In fact, we often expect learning algorithms to acquire only some approximation to the target function, and for this reason the process of learning the target function is often called function approximation. In the current discussion we will use the symbol $\hat{V}$ to refer to the function that is actually learned by our program, to distinguish it from the ideal target function V.

Choosing a Representation for the Target Function

Now that we have specified the ideal target function V, we must choose a representation that the learning program will use to describe the function c that it will learn.

To keep the discussion brief, 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 white pieces on the board

x_3 : the number of black kings on the board

x4: the number of white kings on the board

x5: the number of black pieces threatened by white (i.e., which can be captured on white's next turn)

x_6 : the number of white pieces threatened by black

Thus, our learning program will represent c(b) as a linear function of the form

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:

Target function representation

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 for b.

In other words, each training example is an ordered pair of the form (b,).

For instance, the following training example describes a board state b in which black has won the game (note x2 = 0 indicates that red has no remaining pieces) and for which the target function value Vtrain (b) is therefore +100.

<<x1=3, x2=0, x3=1, x4=0, x5=0, x6=0>, +100)

Estimating Training Values

Recall that according to our formulation of the learning problem, the only training information available to our learner is whether the game was eventually won or lost.

On the other hand, we require training examples that assign specific scores to specific board states. While it is easy to assign a value to board states that correspond to the end of the game, it is less obvious how to assign training values to the more numerous intermediate board states that occur before the game's end. For example, even if the program loses the game, it may still be the case that board states occurring early in the game should be rated very highly and that the cause of the loss was a subsequent poor move.

Despite the ambiguity inherent in estimating training values for intermediate boa rd states, one simple approach has been found to be surprisingly successful. This approach is to assign the training value of V_{train} (b) for any intermediate board state b to be

(successor (b)),where is the learner's current approximation to V and where S uccessor (b) denotes the next board state following b for which it is again the program's turn to move

.This rule for estimating training values can be summarized as

Adjusting the Weights

The algorithm can be viewed as performing a stochastic grad ient-descent search through the space of possible hypotheses (weight values) to minimize the squared error E.

The LMS algorithm is defined as follows:

LMS weight update rule.

• For each training example (b, (b))

• Use the current weights to calculate (b)

• For each weight, update it.

Here η is a small constant (e.g., 0.1) that moderates the size of the weight update. Notice that when the error (Vtrain(b)- (b)) is zero, no weights are changed. When (Vtrain(b) - (b)) is positive (i.e., when f(b) is too low), then each weight is increased in proportion to the value of its corresponding feature. This will raise the value of (b), reducing the error.

The Final Design

The final design of our checkers learning system can be naturally described by four distinct program modules that represent the central components in many learning systems. These four modules, summarized in Figure 1.1, are as follows:

• The Performance System is the module that must solve the given performance task, in this case playing checkers, 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.

• 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. As shown in the diagram, each training example in this case corresponds to some game state in the trace, along with an estimate Vtrain,f the target function value for this example.

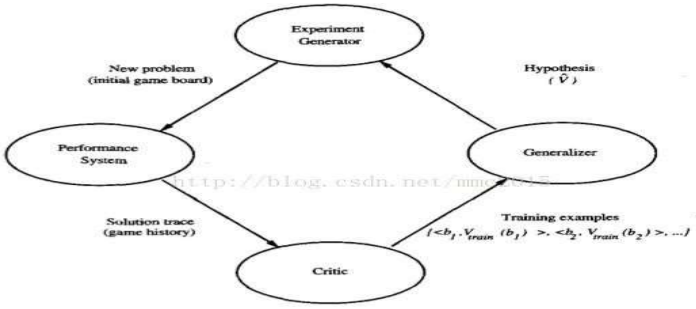


FIGURE 1.1
Final design of the checkers learning program.

- 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. In our example, the generalize corresponds to the LMS algorithm, and the output hypothesis is the function f described by the learned weights wo, . . . , W6.
- The Experiment Generator takes as input the current hypothesis (currently learned function) 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.
- In our example, the Experiment Generator follows a very simple strategy. It always proposes the same initial game board to begin a new game.

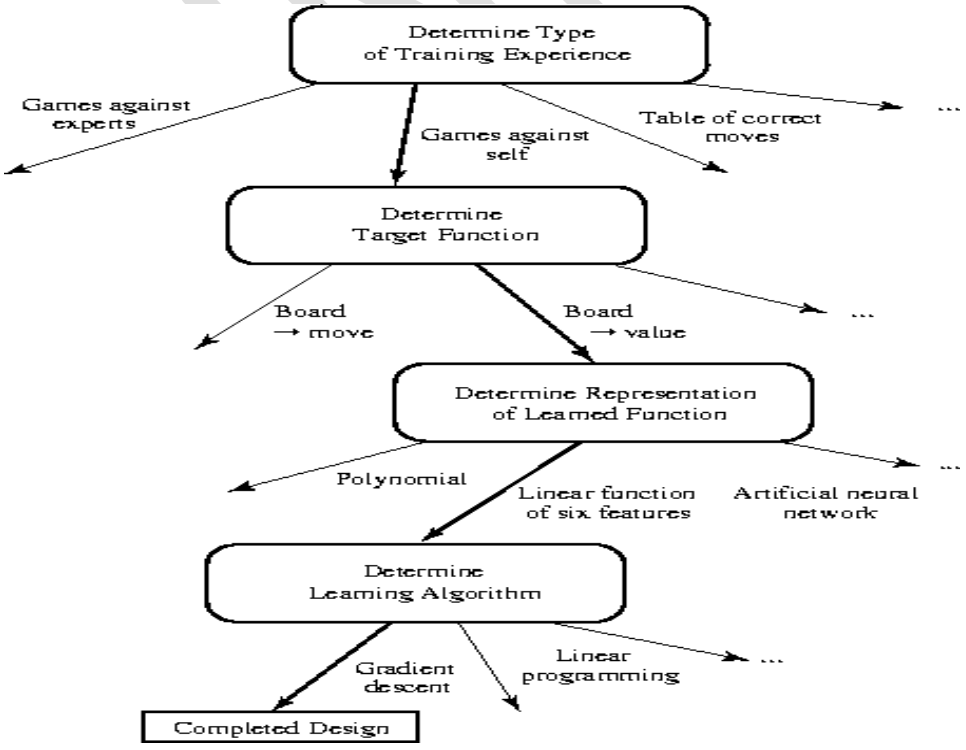


Figure 1.2 Summary of choices in designing the checkers learning program

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.
- The hypothesis space consists of all evaluatio n functions that can be represented by some choice of values for the weights w_o 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.

1.3.1 Issues in Machine Learning

- 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?
- 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?
- How can the learner automatically alter its representation to improve its ability to represent and learn the target function?

Chapter 2: Concept Learning

Introduction
Concept learning task,
Concept learning as search,
FIND-S algorithm,
Version space Candidate Elimination algorithm,
2.7 Inductive Bias.

INTRODUCTION

- Much of learning involves acquiring general concepts from specific training examples. People, for example, continually learn general concepts or categories such as "bird", "car", etc.
- Each such 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).
- 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 TASK

- Consider the example task of learning the target concept "days on which a person enjoys his favourite 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 a person enjoys his favourite 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.
- Let us begin by considering a simple representation in which each hypothesis consists of a conjunction of constraints on the instance attributes.
- In particular, 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" phi* 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).

To illustrate, the hypothesis that Aldo enjoys his favorite sport only on cold days with high humidity (independent of the values of the other attributes) is represented by the expression

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

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*.

- The most general hypothesis-that every day is a positive example- is represented by

  <?, ?, ?, ?, ?, ?>

- And the most specific possible hypothesis-that no day is a positive example-is represented by **<Φ, Φ, Φ, Φ, Φ, Φ>**

To summarize, the EnjoySport concept learning task requires learning the set of days for which EnjoySport = yes, describing this set by a conjunction of constraints over the instance attributes.

In general, any concept learning task can be described by the set of instances over which the target function is defined, the target function, the set of candidate hypotheses considered by the learner, and the set of available training examples.

Notation

The EnjoySport concept learning task

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), "0 (no value is acceptable), or a specific value.

Target concept c: EnjoySport: X → {0,l)}

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.

The Inductive Learning Hypothesis

The inductive learning hypothesis is any hypothesis found to approximate the target function well over a sufficiently large set of training examples will also approximate the target function well ove r other unobserved examples.

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. It is important to note that by selecting a hypothesis representation, the designer of the learning algorithm implicitly defines the space of all hypotheses that the program can ever represent and therefore can ever learn.
- Consider, for example, the instances X and hypotheses H in the EnjoySport learning task.
- Given that the attribute Sky has three possible values, and that AirTemp, Humidity, Wind, Water, and Forecast each have two possible values, the instance space X contains exactly

  3.2 .2 .2 2 .2 = 96 distinct instances.
- A similar calculation shows that there are

  5.4.4.4.4.4 = 5120 syntactically distinct hypotheses within H.
- However, that every hypothesis containing one or more "Φ" symbols represents the empty set of instances; that is, it classifies every instance as negative. Therefore, the number of semantically distinct hypotheses is only 1 + (4.3.3.3.3.3) = 973.

General-to-Specific Ordering of Hypotheses

• A general-to-specific ordering of hypotheses. By taking advantage of this naturally occurring structure over the hypothesis space, we can design learning algorithms that exhaustively search even infinite hypothesis spaces without explicitly enumerating every hypothesis.

• To illustrate the general-to-specific ordering, consider the two hypotheses

h1 = <Sunny, ?, ?, Strong, ?, ?>

h2 = <Sunny, ?, ?, ?, ?, ?>

• Now consider the sets of instances that are classified positive by hl and by h2. Because h2 imposes fewer constraints on the instance, it classifies more instances as positive.

In fact, any instance classified positive by hl will also be classified positive by h2. Therefore, we say that h2 is more general than h1.

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 $h_j \geq_g h_k$) if and only if

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

We will also find it useful to consider cases where one hypothesis is strictly more general than the other.

Therefore, we will say that $\mathbf{h_j}$ is (strictly) more_general_than $\mathbf{h_k}$ (written $\mathbf{h_j} >_g \mathbf{h_k}$) if and only if $(\mathbf{h_j} \geq_g \mathbf{h_k}) \wedge (\mathbf{h_k} \not\geq_g \mathbf{h_j})$.

Finally, we will sometimes find the inverse useful and will say that **hj** is **more_specific_than** $\mathbf{h_k}$ when $\mathbf{h_k}$ is **more_general-than** $\mathbf{h_j}$.

To illustrate these definitions, consider the three hypotheses h_l, h_2, and h_3 from our ***Enjoysport*** example, shown in Figure 2.1.

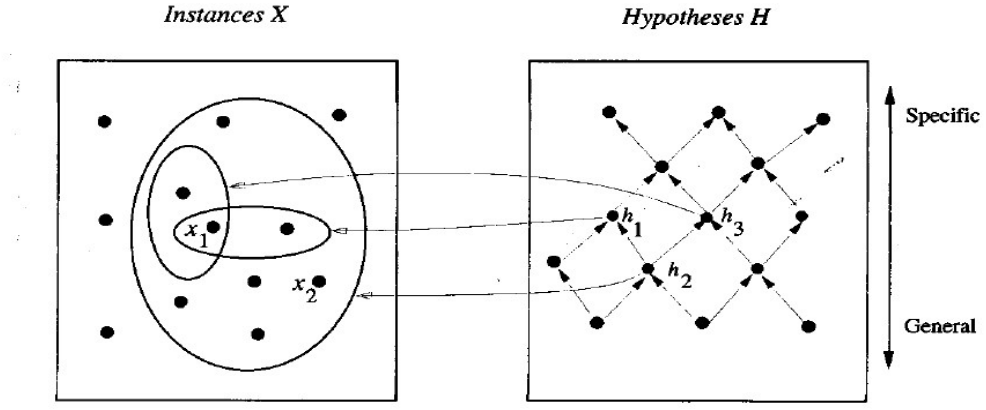


FIGURE 2.1

Hypothesis $\mathbf{h_2}$ is more general than $\mathbf{h_l}$ because every instance that satisfies $\mathbf{h_l}$ also satisfies $\mathbf{h_2}$. Similarly, $\mathbf{h_2}$ is more general than $\mathbf{h_3}$. Note that neither $\mathbf{h_l}$ nor $\mathbf{h_3}$ is more general than the other; although the instances satisfied by these two hypotheses intersect, neither set subsumes the other.

Formally, the $\geq_g$, relation defines a partial order over the hypothesis space H (the relation is reflexive, anti-symmetric, and transitive). Informally, when we say the structure is a partial (as opposed to total) order, we mean there may be pairs of hypotheses such as $\mathbf{h_1}$ and $\mathbf{h_3}$, such that $\mathbf{h_1} \not\geq_g \mathbf{h_3}$ and $\mathbf{h_3} \not\geq_g \mathbf{h_1}$.

The $\geq_g$ relation is important because it provides a useful structure over the hypothesis space H for *any* concept learning problem.

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.)

• FIND-S algorithm defined above, assume the learner is given the sequence of training examples from for the EnjoySport task.

• The first step of FIND-S is to initialize h to the most specific hypothesis in H Upon observing the first training example from Table 2.1, which happens to be a positive example, it becomes clear that our hypothesis is too specific.

• In particular, none of the "Φ " constraints in h are satisfied by this example, so each is replaced by the next more general constraint that fits the example; namely, the attribute

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

TABLE 2.3
FIND-S Algorithm.

values for this training example.

h (Sunny, Warm, Normal, Strong, Warm, Same)

• This h is still very specific; it asserts that all instances are negative except for the single positive training example we have observed. Next, the second training example (also positive in this case) 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. The refined hypothesis in this case is

h *(Sunny, Warm, ?, Strong, Warm, Same)*

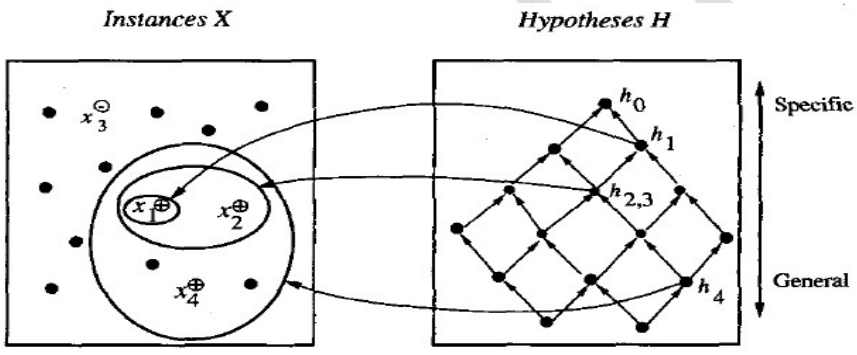
• Upon encountering the third training example- in this case a negative example the algorithm makes no change to h. In fact, the FIND-S algorithm simply ignores every negative example!

• To complete our trace of FIND-S, the fourth (positive) example leads to a further generalization of h

h *(Sunny, Warm, ?, Strong, ?, ?)*

• The FIN D-S algorithm illustrates one way in which the more-general- than partial ordering can be used to organize the search for an acceptable hypothesis.

• The search moves from hypothesis to hypothesis, searching from the most specific to progressively more general hypotheses along one chain of the partial ordering.



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 ? ? >

FIGURE 2.2: The hypothesis space search performed by FIND-S.

The search begins (h_o) with the most specific hypothesis in H, then considers increasingly general hypotheses (h_l through h_4) as mandated by the training examples. In the instance space diagram, positive training examples are denoted by "+" and negative by "-," and instances that have not been presented as training examples are denoted by a solid circle.

Complaints about Find-S

- Has the learner converged to the correct target concept? Although FIND-S will find a hypothesis consistent with the training data, it has no way to determine whether it has found the ***only*** hypothesis in ***H*** consistent with the data (i.e., the correct target concept), or whether there are many other consistent hypotheses as well.

- Why prefer the most specific hypothesis? In case there are multiple hypotheses consistent with the training examples, FIND-S will find the most specific. It is unclear whether we should prefer this hypothesis over, say, the most general, or some other hypothesis of intermediate generality.
- Are the training examples consistent? In most practical learning problems there is some chance that the training examples will contain at least some errors or noise. Such inconsistent sets of training examples can severely mislead FIND-S, given the fact that it ignores negative examples.
- What if there are several maximally specific consistent hypotheses? There can be several maximally specific hypotheses consistent with the data. In this case, FIND-S must be extended to allow it to backtrack on its choices.

VERSION SPACES AND THECANDIDATE-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 ofall 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.

Representation

The CANDIDATE-ELIMINATION ALGORITHM finds all desirable hypotheses that are consistent with the observed training examples.

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

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

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

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

The LIST-THEN-ELIMINATE Algorithm

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 $VersionSpace$

- First initializes the version space to contain all hypotheses in H, then eliminates any hypothesis found inconsistent with any training example. The version space of candidate hypotheses thus shrinks as more examples are observed.
- Unfortunately, it requires exhaustively enumerating all hypotheses in H-an unrealistic requirement for all but the most trivial hypothesis spaces.

A More Compact Representation for Version Spaces

To illustrate this representation for version spaces, consider again the Enjoysport concept learning problem described in "**Notations**". Recall that given the four training examples from Table 2.1, FIND-S outputs the hypothesis

h = <Sunny, Warm, ?, Strong, ?, ?>

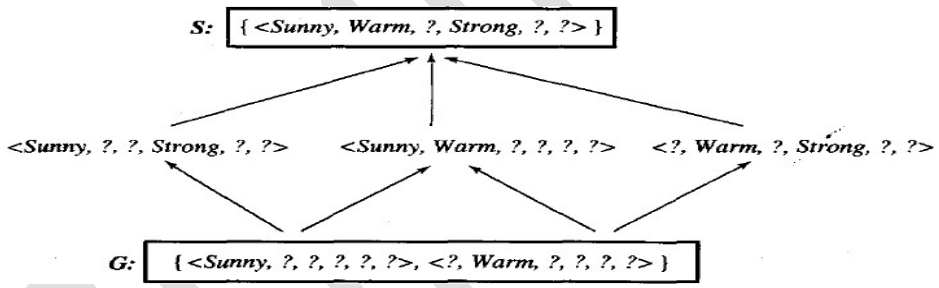


FIGURE 2.3: A version space with its general and specific boundary sets.

The version space includes all six hypotheses shown here, but can be represented more simply by S and G. Arrows indicate instances of the more- general-than relation. This is the version space for the Enjoysport concept learning problem and training examples described in Table 2.1.

Definition: The general boundary G, with respect to hypothesis space H and training data D, is the set of maximally general members of H consistent with D.

$$G \equiv \{g \in H | Consistent(g, D) \wedge (\neg \exists g' \in H)[(g' >_g g) \wedge Consistent(g', D)]\}$$

Definition: The **specific boundary** S, with respect to hypothesis space *H* and training data *D,* is the set of minimally general (i.e., maximally specific) members of *H* consistent with *D.*

$$S \equiv \{s \in H | Consistent(s, D) \wedge (\neg \exists s' \in H)[(s >_g s') \wedge Consistent(s', D)]\}$$

CANDIDATE-ELIMINATION LEARNING ALGORITHM

- The CANDIDATE-ELIMINATION ALGORITHM computes the version space containing all hypotheses from H that are consistent with an observed sequence of training examples.
- It begins by initializing the version space to the set of all hypotheses in H; that is, by initializing the G boundary set to contain the most general hypothesis in H

  G_o {(?, ?, ?, ?, ?, ?)}

  And initializing the S boundary set to contain the most specific (least general hypothesis S_o {0,0,0,0, 0,0)

These two boundary sets delimit the entire hypothesis space, because every other hypothesis in H is both more general than S_o and more specific than G_o. As each training example is considered, the S and G boundary sets are generalized and specialized, respectively, to eliminate from the version space any hypotheses found inconsistent with the new training example.

CANDIDATE-ELIMINATION Algorithm using version space

Initialize G to the set of maximally general hypotheses in H

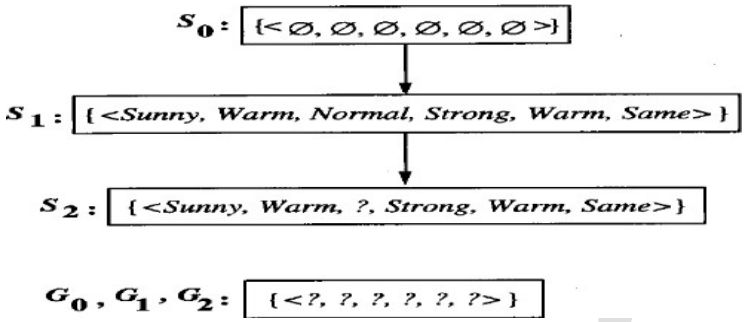
Initialize S to the set of maximally specific hypotheses in H

For each training example d, do

- If d is a positive example
  - Remove from G any hypothesis inconsistent with d,
  - For each hypothesis s in S that is not consistent 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 anot her hypothesis in S
- If d is a negative example
    - Remove from S any hypothesis inconsistent with d
    - For each hypothesis g in G that is not consistent 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

AN ILLUSTRATIVE EXAMPLE

CANDIDATE-ELIMINATION Trace 1: S_o and G_o are the initial boundary sets corresponding to the most specific and most general hypotheses. Training examples 1 and 2 force the S boundary to become more general, as in the FIND-S algorithm. They have no effect on the G boundary.



Training examples:

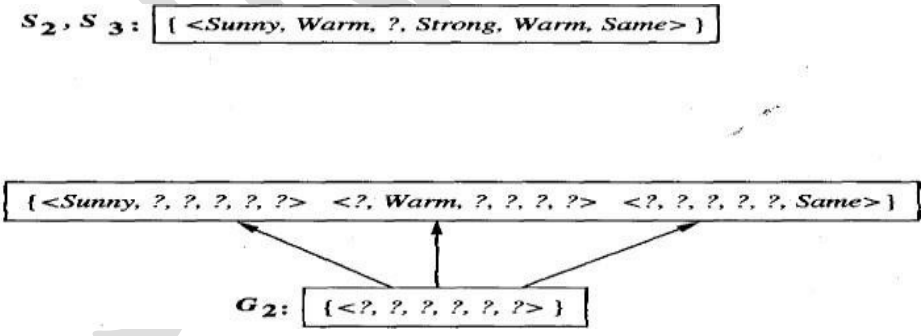
1 <Sunny, Warm, Normal, Strong, Warm, Same>, Enjoy Sport = Yes

2 <Sunny, Warm, High, Strong, Warm, Same>, Enjoy Sport = Yes

Training Example:

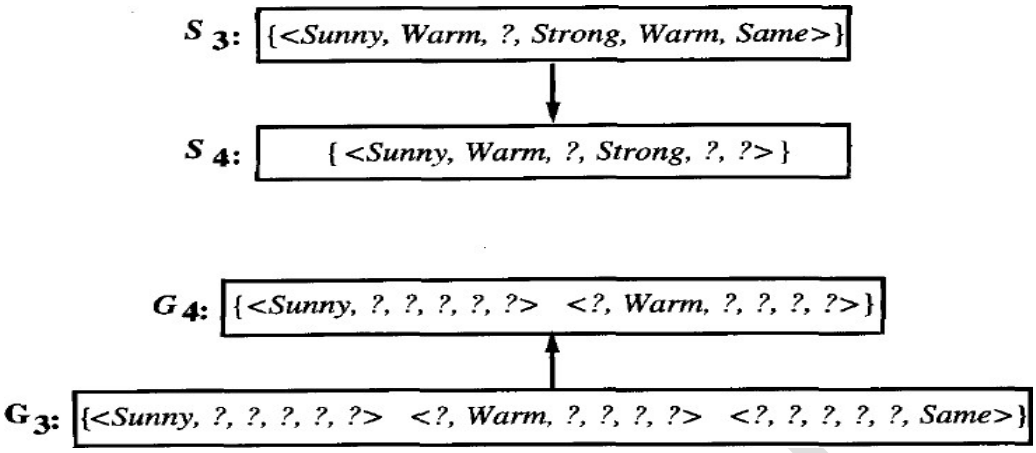
3 <Rainy, Cold, High, Strong, Warm, Change>, Enjoy Sport = No

CANDIDATE-ELMN ATION Trace 2: Training example 3 is a negative example that forces the G2 boundary to be specialized to G3. Note several alternative maximally general hypotheses are included in G_3.



Training Example: 4. <Sunny, Warm, High, Strong, Cool, Change>, Enjoy Sport = Yes

CANDIDATE-ELIMINATIO Trace 3: The positive training example generalizes the S boundary, from S3 to S4. O ne member of G3 must also be deleted, because it is no longer more general than the S4 boundary.



2.7 INDUCTIVE BIAS

The CANDIDATE-ELIMINATION Algorithm will converge toward the true target concept provided it is given accurate training examples and provided its initial hypothesis space contains the target concept.

2.7.1 A Biased Hypothesis Space

• Suppose we wish to assure that the hypothesis space contains the unknown target concept. The obvious solution is to enrich the hypothesis space to include every possible hypothesis.

• To illustrate, consider again the EnjoySport example in which we restricted the hypothesis space to include only conjunctions of attribute values. Because of this restriction, the

Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Cool	Change	Yes
2	Cloudy	Warm	Normal	Strong	Cool	Change	Yes
3	Rainy	Warm	Normal	Strong	Cool	Change	No

hypothesis space is unable to represent even simple disjunctive target concepts such as "Sky = Sunny or Sky = Cloudy." In fact, given the following three training examples of this disjunctive hypothesis, our algorithm would find that there are zero hypotheses in the version space.

• To see why there are no hypotheses consistent with these three examples, note that the most specific hypothesis consistent with the first two examples and representable in the given hypothesis space H is

S2 : (?, Warm, Normal, Strong, Cool, Change)

• This hypothesis, although it is the maximally specific hypothesis from H that is consistent with the first two examples, is already overly general: it incorrectly covers the third (negative) training example.

• The problem is that we have biased the learner to consider only conjunctive hypotheses. In this case we require a more expressive hypothesis space.

An UN-Biased Learner

• Idea: Choose H that expresses every teachable concept (i.e., H is the power set of X)

• Consider = disjunctions, conjunctions, negations over previous H.

E.g.,

$$\langle Sunny\ Warm\ Normal\ ?\ ?\ ?\rangle \vee \neg\langle ?\ ?\ ?\ ?\ ?\ Change\rangle$$

• To see why, suppose we present three positive examples (xl,x2, x3)and two negative examples(x4, x5) to the learner. At this point, the S boundary of the version space will contain the hypothesis which is just the disjunction of the positive examples because this is the most specific possible hypothesis that covers these three examples.

$$S : \{(x_1 \vee x_2 \vee x_3)\}$$

Similarly, the G boundary will consist of the hypothesis that rules out only the observed negative examples.

$$G : \{\neg(x_4 \vee x_5)\}$$

• The problem here is that with this very expressive hypothesis representation, the S boundary will always be simply the disjunction of the observed positive examples, while the G boundary will always be the negated disjunction of the observed negative examples.

• Therefore, the only examples that will be unambiguously classified by S and G are the observed training examples themselves. In order to converge to a single, final target concept, we will have to present every single instance in X as a training example.

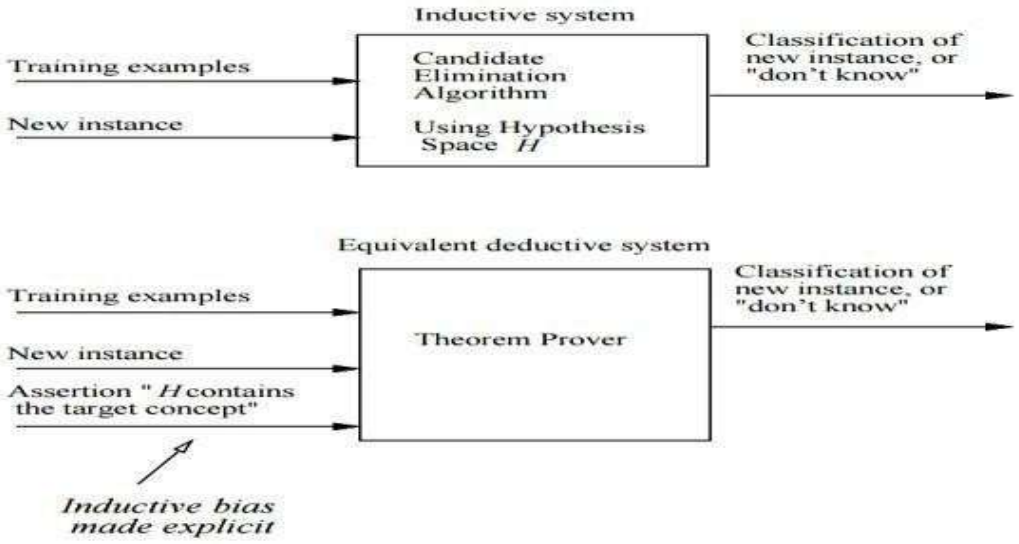
Definition: Consider a concept learning algorithm L for the set of instances X. Let c be an arbitrary concept defined over X, and let $D_c = \{\langle x, c(x)\rangle\}$ be an arbitrary set of training examples of c. Let $L(x_i, D_c)$ denote the classification assigned to the instance x_i by L after training on the data D_c. The **inductive bias** of L is any minimal set of assertions B such that for any target concept c and corresponding training examples D_c

$$(\forall x_i \in X)[(B \wedge D_c \wedge x_i) \vdash L(x_i, D_c)] \qquad (2.1)$$

Inductive Systems and Equivalent Deductive Systems

• Modeling inductive systems by equivalent deductive systems. The input-output behavior of the CANDIDATE-ELIMINATION ALGORITHM using a hypothesis space H is identical to that of a deductive theorem prover utilizing the assertio n "H contains the target concept."

• This assertion is therefore called the inductive bias of the CANDIDATE- ELIMINATION ALGORITHM. Characterizing inductive systems by their inductive bias allows modeling them by their equivalent deductive systems. This provides away to compare inductive systems according to their policies for generalizing beyond the observed training data.



• One advantage of viewing inductive inference systems in terms of their inductive bias is that it provides a nonprocedural means of characterizing their policy for generalizing beyond the observed data.

• A second advantage is that it allows comparison of different learners according to the strength of the inductive bias they employ. Consider, for example, the following three learning algorithms, which are listed from weakest to strongest bias.

• ROTE-LEARNER: Learning corresponds simply to storing each observed training example in memory. Subsequent instances are classified by looking them up in memory. If the instance is found in memory, the stored classification is returned. Otherwise, the system refuses to classify the new instance.

• CANDIDATE-ELIMINATION ALGORITHM: New instances are classified only in the case where all members of the current version space agree on the classification. Otherwise, the system refuses to classify the new instance.

• FIND-S: This algorithm, described earlier, finds the most specific hypothesis consistent with the training examples. It then uses this hypothesis to classify all subsequent instances.

• The ROTE-LEARNER has no inductive bias. The classifications it provides for new instances follow deductively from the observed training examples, with no additional assumptions required.

• The CANDIDATE-ELIMINATION ALGORITHM has a stronger inductive bias: that the target concept can be represented in its hypothesis space. Because it has a stronger bias, it will classify some instances that the ROTE-LEARNER will not.

• Of course the correctness of such classifications will depend completely on the correctness of this inductive bias. The FIND-S algorithm has an even stronger inductive bias. In addition to the assumption that the target concept can be described in its hypothesis space, it has an additional inductive bias assumption.