Module-1
Introduction

1. Introduction

- Machine learning, describing a variety of learning paradigms, algorithms, theoretical results, and applications.
- Machine learning is inherently a multidisciplinary field.
- It draws on results from artificial intelligence, probability and statistics, computational complexity theory, control theory, information theory, philosophy, psychology, neurobiology, and other fields.

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.

Applications of machine learning

- **Learning to recognize spoken words.** All of the most successful speech recognition systems employ machine learning in some form. For example, the SPHINX system (e.g., Lee 1989) learns speaker-specific strategies for recognizing the primitive sounds (phonemes) and words from the observed speech signal. Neural network learning methods (e.g., Waibel et al. 1989) and methods for learning hidden Markov models (e.g., Lee 1989) 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 other cars. Similar techniques have possible applications in many sensor-based control problems.
- **Learning to classify new 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 have been used by NASA to learn how to classify celestial objects from the second Palomar

Observatory Sky Survey (Fayyad et al. 1995). This system is now used to automatically classify all objects in the Sky Survey, which consists of three terra bytes of image data.

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

  The most successful computer programs for playing games such as backgammon
  are based on machine learning algorithms. For example, the world's top computer program for backgammon,
  TD-GAMMON (Tesauro 1992, 1995). learned its strategy by playing over one million practice games against itself. It now plays at a level competitive with the human world champion. Similar techniques have applications in many practical problems where very large search spaces must be examined efficiently.

Some disciplines and examples of their influence on machine learning.

- **Artificial intelligence**

  Learning symbolic representations of concepts. Machine learning as a search problem. Learning as an approach to improving problem solving. Using prior knowledge together with training data to guide learning.

- **Bayesian methods**

  Bayes' theorem as the basis for calculating probabilities of hypotheses. The naive Bayes classifier Algorithms for estimating values of unobserved variables.

- **Computational complexity theory**

  Theoretical bounds on the inherent complexity of different learning tasks, measured in terms of the computational effort, number of training examples, number of mistakes, etc. required in order to learn.

- **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.

- **Information theory**

  Measures of entropy and information content. Minimum description length approaches to learning. Optimal codes and their relationship to optimal training sequences for encoding a hypothesis.

- **Philosophy**

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

- **Psychology and neurobiology**

The power law of practice, which states that over a very broad range of learning problems, people's response time improves with practice according to a power law. Neurobiological studies motivating artificial neural network models of learning.

- **Statistics**

  Characterization of errors (e.g., bias and variance) that occur when estimating the accuracy of a hypothesis based on a limited sample of data. Confidence intervals, statistical tests.

1.2 Designing a Learning System

Basic design issues and approaches to machine learning with various example.

1) Choosing the Training Experience
2) Choosing the Target Function
3) Choosing a Representation for the Target Function
4) Choosing a Function Approximation Algorithm
5) The Final Design

1.2.1 Choosing the Training Experience

- 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.
- 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 system performance P must be measured. In general, learning is most reliable when the training examples follow a distribution similar to that of future test examples.

Example : A checkers learning problem:

- 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
  - The exact type of knowledge to be, learned

- a representation for this target knowledge
- a learning mechanism.

1.2.2 Choosing the Target Function

- For Example, 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.
- Learning task is representative of a large class of tasks for which the legal moves that define some large search space are known a priori, but for which the best search strategy is not known.
- Many optimization problems fall into this class, such as the problems of scheduling and controlling manufacturing processes where the available manufacturing steps are well understood, but the best strategy for sequencing them is not.
- we will find it useful to define one particular target function V among the many that produce optimal play. As we shall see, this will make it easier to design a training algorithm. 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, then V(b) = V(bl), 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).

1.2.3 Choosing a Representation for the Target Function:

- we must choose a representation that the learning program will use to describe the function V that it will learn.
- for any given board state, the function V will be calculated as a linear combination of the following board features

  xl: the number of black pieces on the board
  x2: the number of red pieces on the board
  x3: the number of black kings on the board
  x4: the number of red kings on the board
  x5: the number of black pieces threatened by red (i.e. which can be captured on red's next turn)
  x6: the number of red pieces threatened by black.

- Thus, our learning program will represent c(b) 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, whereas the weight wo will provide an additive constant to the board value.

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 + 8
- Target function representation.

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

The first three items above correspond to the specification of the learning task, whereas the final two items constitute design choices for the implementation of the learning program.

1.2.4 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 $V_{train}(b)$ for b. In other words, each training example is an ordered pair of the form (b, Vtrain,(b)).

1.2.4.1 Estimating Training Values

- 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.
- This rule for estimating training values can be summarized as
  Rule for estimating training values.

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

1.2.4.2 Adjusting the Weights

- All that remains is to specify the learning algorithm for choosing the weights w_i to best fit the set of training examples $\{(b, V_{train}(b))\}$. As a first step we must 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 square error E between the training values and the values predicted by the hypothesis V.

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

- Several algorithms are known for finding weights of a linear function that minimize E in that LMS algorithm is one. The LMS algorithm is defined as follows:
- LMS weight update rule.
  - For each training example $(b, V_{train}(b))$
  - Use the current weights to calculate ?(b)
  - For each weight w_i, update it as

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

- when the error ($V_{train}(b)$ - c(b)) is zero, no weights are changed. When ($V_{train}(b)$ - e(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 V^(b), reducing the error.

1.2.5 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.
- 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.

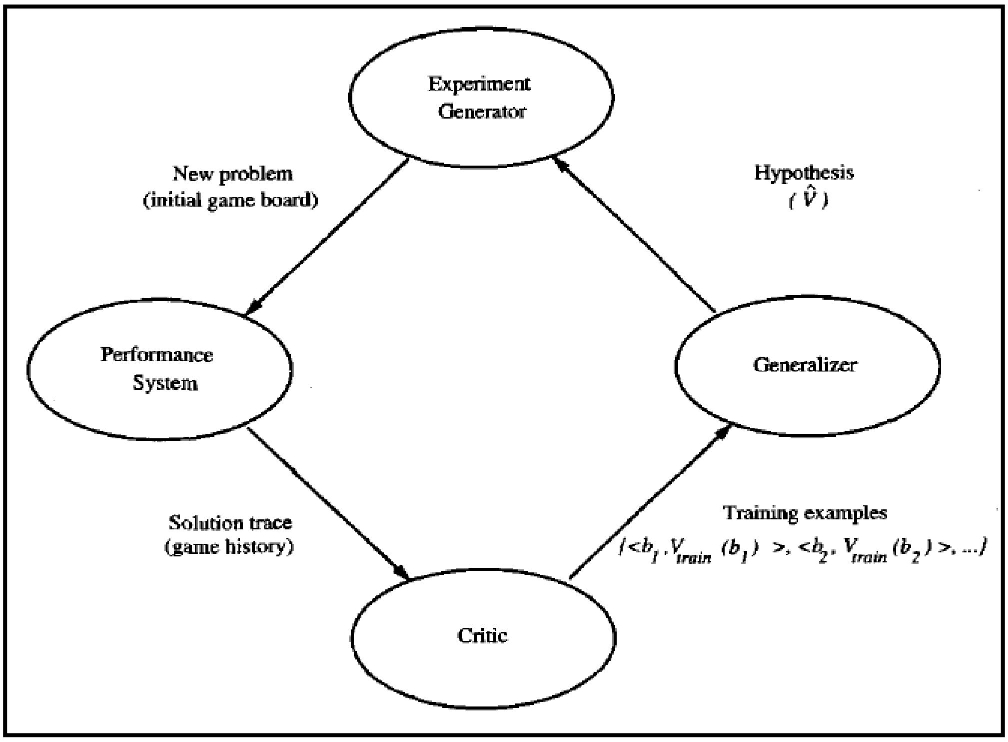


Figure 1.1: Final design of the checkers learning program

The strategy used by the Performance System to select its next move at each step is determined by the learned V evaluation function.

- 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 V_{train} of the target function value for this example.
- 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.
- 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.
- The sequence of design choices made for the checkers program is summarized in Figure 1.2.

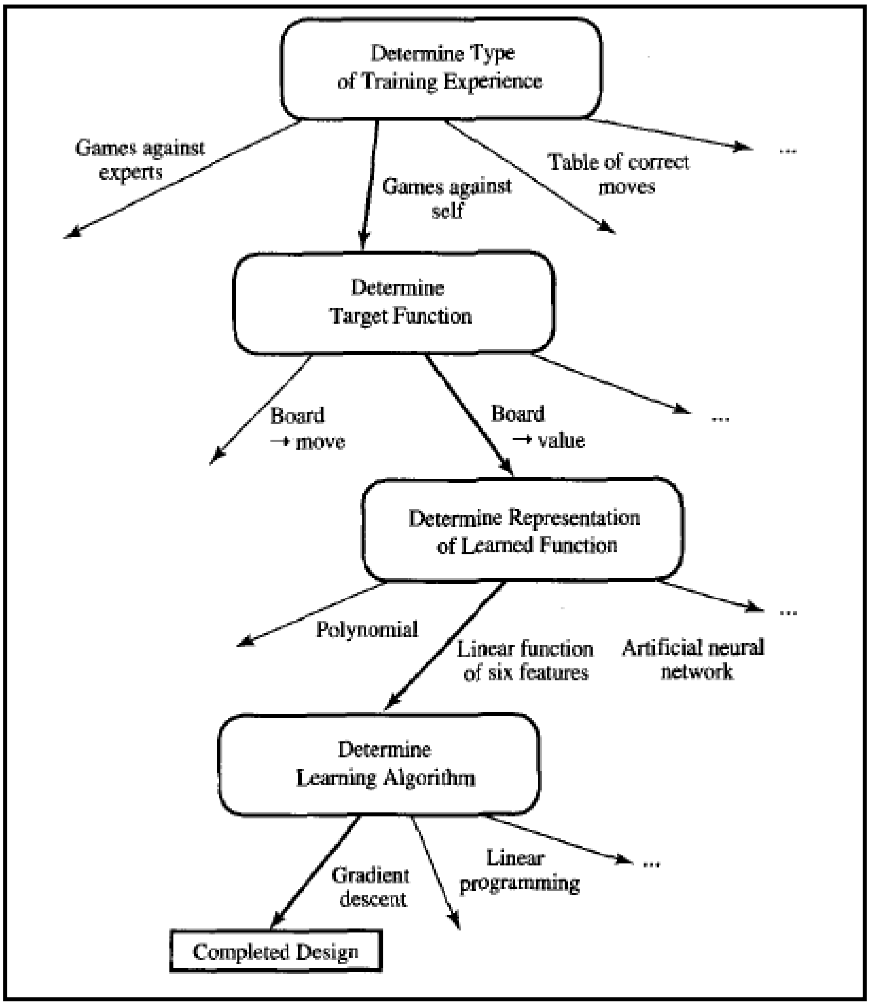


Figure 1.2: Summary of choices in designing the checkers learning program.

- Restricted the type of knowledge that can be acquired to a single linear evaluation function. Furthermore, we have constrained this evaluation function to depend on only the six specific board features provided. If the true target function V can indeed be represented by a linear combination of these particular features.

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.
- 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.
- How much training data is sufficient?

- When and how can prior knowledge held by the learner guide the process of generalizing from examples.
- When and how can prior knowledge held by the learner guide the process of generalizing from examples.
- What is the best strategy for choosing a useful next training experience? 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?

1.4 Concept Learning

- Inducing general functions from specific training examples is a main issue of machine learning.
- Concept learning - a learning task in which 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.
- Concept learning - also known as category learning, concept attainment, and concept formation.
- Concept Learning: Acquiring the definition of a general category from given sample of positive and negative training examples of the category.

A Formal Definition for Concept Learning:

- Inferring a Boolean-valued function from training examples of its input and output. Learn the definition of a concept from examples.
- An example for concept-learning is the learning of bird-concept from the given examples of birds (positive examples) and non-birds (negative examples).
- The concept of a bird is the subset of all objects (i.e., the set of all things or all animals) that belong to the category of bird.
- Each concept is 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 every other animal.

Every concept has two components:

- **Attributes:** These are features of a stimulus that one must look for todecide if that stimulus is a positive instance of the concept.

- **A rule:** This statement that specifies which attributes must be present or absent for a stimulus to qualify as a positive instance of the concept.

  The simplest rules refer to the presence or absence of a single attribute. For example, a "vertebrate" animal is defined as an animal with a backbone. Which of these stimuli are positive instances?
  - This rule is called affirmation. It says at a stimulus must possess a single specified attribute to qualify as a positive instance of a concept.
  - The opposite or "complement" of affirmation is negation. To qualify as a positive instance, a stimulus must lack a single specified attribute.
  - An invertebrate animal is one that lacks a backbone. These are the positive and negative instances when the negation rule is applied.

A Concept Learning Task – EnjoySport Training

Table 1.1: Positive and Negative training examples for the target concept EnjoySport

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

A set of example days, and each is described by six attributes.

- The task is to learn to predict the value of Enjoy Sport for arbitrary day, based on the values of its attribute values -*Target concept*.
- **Goal**: To infer the "best" concept-description from the set of all possible hypotheses.
- Each hypothesis consists of a conjunction of constraints on the instance attributes.
- Each hypothesis will be a vector of six constraints, specifying the values of the six attributes
- (*Sky, AirTemp, Humidity, Wind, Water, and Forecast*).
- Each attribute will be:
  - *Indicate by a "**?**" that any value is acceptable for this attribute.*
  - *Specify a "**single**" required value (eg: Warm) for the attribute, or*
  - *Indicated by a "∅" that no value is acceptabl*e
- hypothesis:
- Sky, AirTemp, Humidity, Wind, Water, Forecast

- *Most General Hypothesis*: Every day is a good day for water sports <?, ?, ?, ?, ?, ?> (Positive example)
- *Most Specific Hypothesis:* No day is a good day for water sports <0, 0, 0, 0, 0, 0> No day is Positive example)
- EnjoySport concept learning task requires learning the sets of days for which EnjoySport = yes, describing this set by a conjunction of constraints over the instance attributes.

1.4.1 Notation

Given:

- Instances X: Set of all Possible days, each described by the attributes
  - Sky (Sunny, Cloudy, and Rainy)
  - Temp (Warm and Cold)
  - Humidity (Normal and High)
  - Wind (Strong and Weak)
  - Water (Warm and Cool)
  - Forecast (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 along with their target concept value c(x).

Determine: A hypothesis h in H such that h(x) = c(x) for all x in X.

When learning the target concept, the learner is presented a set of ***training examples,*** each consisting of an instance ***x*** from X, along with its target concept value ***c(x)*** (e.g., the training examples in Table 1.1). 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. We will often write the ordered pair ***(x, c(x))*** to describe the training example consisting of the instance **x** and its target concept value ***c(x).*** We use the symbol ***D*** to denote the set of available training examples.

- Given a set of training examples of the target concept ***c*,** the problem faced by the learner is to hypothesize, or estimate, ***c*.**
- We use the symbol H to denote the set of ***all possible hypotheses*** that the learner may consider regarding the identity of the target concept. Usually H is determined by the human designer's choice of hypothesis representation. In general, each hypothesis ***h*** in H represents a Boolean-valued function defined over X; that is, ***h*** : X → {O, 1).
- The goal of the learner is to find a hypothesis ***h*** such that ***h(x)*** = ***c(x)*** for a ***x*** in X.

1.4.2 The Inductive Learning Hypothesis

- 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.

1.5 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.
- The hypothesis space has a general-to-specific ordering of hypotheses, and the search can be efficiently organized by taking advantage of a naturally occurring structure over the hypothesis space.
- 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.

Example: EnjoySport Hypothesis Space

- ✓ Sky has 3 possible values, and other 5 attributes have 2 possible values.
- ✓ There are 96 (= 3.2.2.2.2.2) distinct instances in X.
- ✓ There are 5120 (=5.4.4.4.4.4) syntactically distinct hypotheses in H.

  – Two more values for attributes: ? and ∅
- ✓ Every hypothesis containing one or more ∅ symbols represents the empty set of instances. that is, it classifies every instance as negative.

- ✓ There are 973 (= 1 + 4.3.3.3.3.3) semantically distinct hypotheses in H.
  - Only one more value for attributes: ∅
- ✓ one hypothesis representing empty set of instances.
- ✓ Although EnjoySport has small, finite hypothesis
- ✓ space, most learning tasks have much larger(even infinite) hypothesis spaces.
  - We need efficient search algorithms on the hypothesis spaces.

1.5.1 General-to-Specific Ordering of Hypotheses

- The hypothesis space has a general-to-specific ordering of hypotheses, and the search can be efficiently organized.
- Now consider the sets of instances that are classified positive by h_l and by h_2.Because h_2 imposes fewer constraints on the instance, it classifies more instances as positive. In fact, any instance classified positive by h_l will also be classified positive by h_2. Therefore, we say that h_2 is more general than h_l.
- This intuitive "*more general than*" relationship between hypotheses can be defined more precisely as follows. First, for any instance x in X and hypothesis h in H, we say that x satisfies h if and only if h(x) = 1. We now define the ***more_general_than_or equal_to relat***ion in terms of the sets of instances that satisfy the two hypotheses: Given hypotheses h_j and h_k, h_j is more-general-than-equal do h_k if and only if any instance that satisfies hk also satisfies ***hi.***

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

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

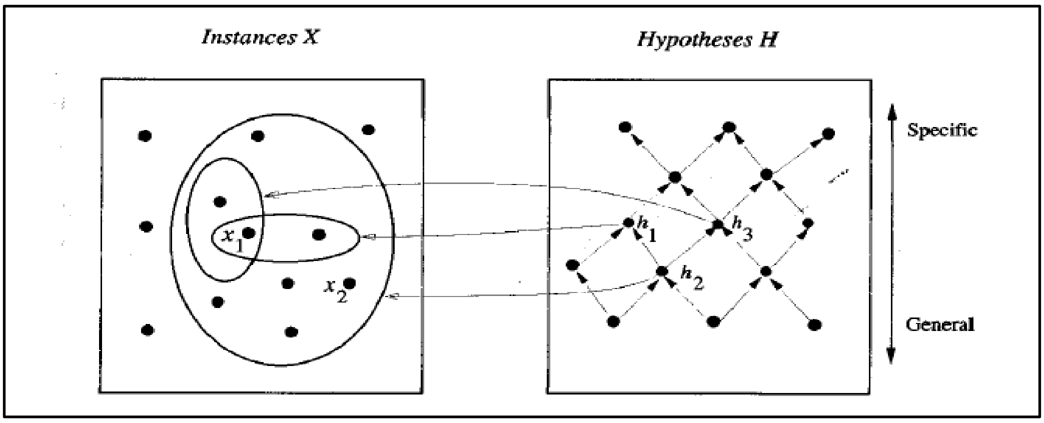
Example:

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

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

- Every instance that are classified as positive by h1 will also be classified as positive by h2 in our example data set. Therefore, h2 is more general than h1.
- We also use the ideas of “strictly” -more-general-than, and more-specific-than [Mitchell].

More general than relation



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, ?*>

Figure 1.3: Instances, hypotheses, and the *m o r e - g e n e r a l - t h a n* relation. 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 *m o r e - g e n e r a l - t h a n* relation, with the arrow pointing toward the less general hypothesis. Note the subset of instances characterized by *h2* subsumes the subset characterized by h_l, hence h_2 is *m o r e - g e n e r a l - t h a n* h_l.

To illustrate these definitions, consider the three hypotheses ***hl, h2,*** and ***h3*** from our ***Enjoysport*** example, shown in Figure 2.1. How are these three hypotheses related by the ***p,*** relation? As noted earlier, hypothesis ***h2*** is more general than ***hl*** because every instance that satisfies ***hl*** also satisfies ***h2.***

Similarly,***h2*** is more general than ***h3.*** Note that neither ***hl*** nor ***h3*** is more general than the other; although the instances satisfied by these two hypotheses intersect,neither set subsumes the other. Notice also that the ***p,*** and >, relations are defined independent of the target concept. They depend only on which instances satisfy the two hypotheses and not on the classification of those instances according to the target concept.

Formally, the ***p,*** relation defines a partial order over the hypothesis space H (the relation is reflexive, antisymmetric, 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 ***hl*** and ***h3,*** such that ***hl not greater than equal to h3*** and *h3 not greater than equal to* ***hl.***

The $\geq_g$ relation is important because it provides a useful structure over the hypothesis space H for ***any*** concept learning problem. The following sections present concept learning algorithms that take advantage of this partial order to efficiently organize the search for hypotheses that fit the training data**.**

1.6 Find-S: Finding a Maximally Specific Hypothesis

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*, in *h*

   If the constraint *a*, is satisfied by *x*

   Then do nothing

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

The first step of FIND- S is to initialize h to the most specific hypothesis in H

h🡨(Ø, Ø, Ø, Ø, Ø, Ø)

none of the "Ø" constraints in *h* are satisfied by this example, so each is replaced by the next more general constraint {h at fits the example; namely, the attribute values for this training example.

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

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 FIND-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.

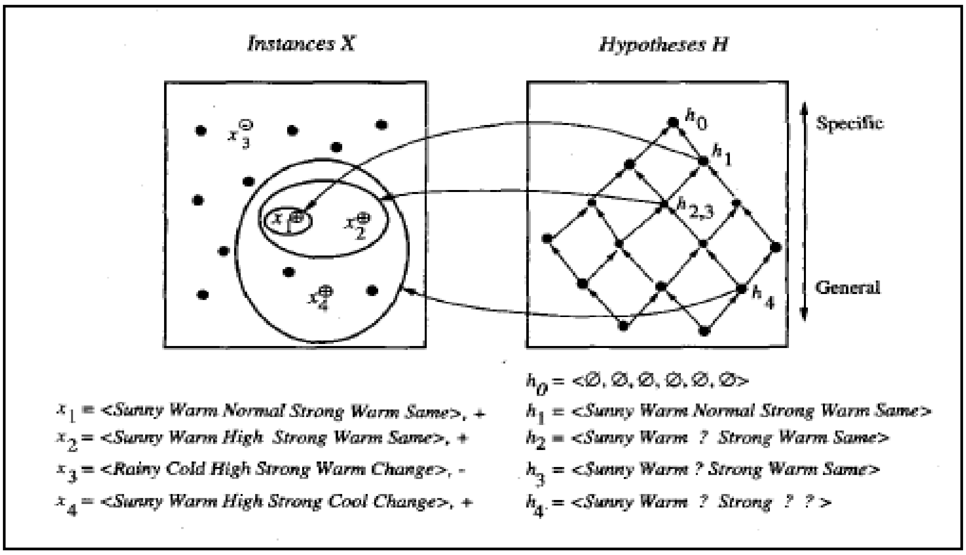


Figure 1.4: illustrates this search in terms of the instance and hypothesis spaces. At each step, the hypothesis is generalized only as far as necessary to cover the new positive example. Therefore, at each stage the hypothesis is the most specific hypothesis consistent with the training examples observed up to this point (hence the name FIND-S).

x1=<Sunny Warm Normal Strong Warm Same>, +

x2 = <Sunny Warm High Strong Warm Same>, +

x3= <Rainy Cold High Strong Warm Change>, -

x4 = <Sunny Warm High Strong Cool Change>, +

h1=<Sunny Warm Normal Strong Warm same>

h2 = <Sunny Warm ? Strong Warm Same>

h3= <Sunny Warm ? Strong Warm Same>

h4= <Sunny Warm ? Strong ? ? >

The hypothesis space search performed by FINDS. The search begins (ho) with the most specific hypothesis in H, then considers increasingly general hypotheses (hl through h4) as mandated by the training examples. In the instance space diagram, positive training examples are denoted by "+," negative by "-," and instances that have not been presented as training examples are denoted by a solid circle.

Issues in Find-S algorithm:

- Has the learner converged to the correct target?
- Why prefer the most specific hypothesis
- Are the training examples consistent (training examples contain some errors)?
- Finally getting one specific hypothesis in Find-S if it is more than one.

1.7 Version Spaces and the Candidate Elimination Algorithm

1.7.1 Representation

The CANDIDATE-ELIMINATION algorithm finds all describable hypotheses that are consistent with the observed training examples. In order to define this algorithm precisely, we begin with a few basic definitions. First, let us say that a hypothesis is `consistent` with the training examples if it correctly classifies these 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 (x, c(x)) in D.

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

An example x is said to satisfy hypothesis h when h(x) = 1, regardless of whether x is a positive or negative example of the target concept. However, whether such an example is consistent with h depends on the target concept, and in particular, whether h(x) = c(x).

The CANDIDATE-ELIMINATION algorithm represents the set of all hypotheses consistent with the observed training examples. This subset of all hypotheses is called the version space with respect to the hypothesis space H and the training examples D, because it contains all plausible versions of the target concept.

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

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

1.7.2 The LIST-THEN-ELIMINATE Algorithm

The LIST-THEN-ELIMINATE algorithm 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, until ideally just one hypothesis remains that is consistent with all the observed examples. The LIST-THEN-ELIMINATE algorithm can be applied whenever the hypothesis space H is finite. It has many advantages, including the fact that it is guaranteed to output all hypotheses consistent with the training data.

1.7.3 A More Compact Representation for Version Spaces

The version space is represented by its most general and least general members.

The LIST-THEN-ELIMINATE Algorithm

1. VersionSpace c a list containing every hypothesis in H
2. For each training example, (x, c(x))

    remove from VersionSpace any hypothesis h for which h(x) # c(x)
3. Output the list of hypotheses in VersionSpace

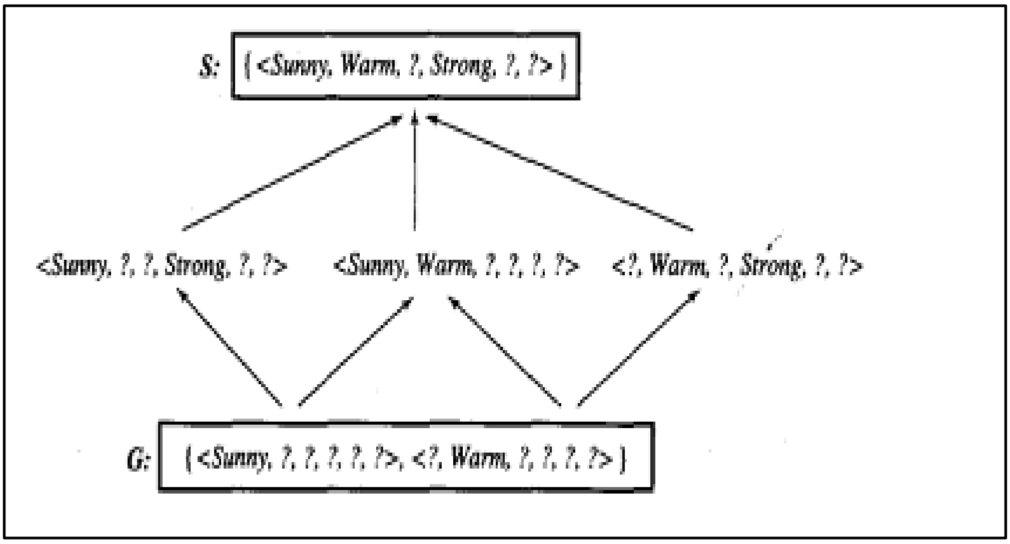


Figure 1.5: A version space with its general and specific boundary sets. The version space includes all six hypotheses shown here, but can be represented more simply by S and G. Arrows indicate instances of the more-general-than relation. This is the version space for the Enjoy sport concept learning problem and training examples.

To illustrate this representation for version spaces, consider again the En*joysport* concept learning problem described in Table 1.1. Recall that given the four training examples from Table 2.1, FIND-S outputs the hypothesis

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

In fact, this is just one of six different hypotheses from H that are consistent with these training examples. All six hypotheses are shown in Figure 1.5. They constitute the version space relative to this set of data and this hypothesis representation.

The arrows among these six hypotheses in Figure 1.5 indicate instances of the *more-general-than* relation. The CANDIDATE-ELIMINTION Algorithm represents the version space by storing only its most general members and its most specific (labeled *S* in the figure). Given only these two sets *S* and G, it is possible to enumerate all members of the version space as needed by generating the hypotheses that lie between these two sets in the general-to-specific partial ordering over hypotheses.

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.

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

Theorem 2.1. Version space representation theorem. Let X be an arbitrary set of instances and let H be a set of boolean-valued hypotheses defined over X. Let c : X → {O, 1) be an arbitrary target concept defined over X, and let D be an arbitrary set of training examples {(x, c(x))). For all X, H, c, and D such that S and G are well defined,

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

Proof.

- To prove the theorem it suffices to show that (1) every h satisfying the right- hand side of the above expression is in $VS_{H,D}$.
- (2) every member of VS_{HD}, satisfies the right-hand side of the expression. To show (1) let g be an arbitrary member of G, s be an arbitrary member of S, and h be an arbitrary member of H, such that $g \geq_g h \geq_g s$.

- Then by the definition of S, s must be satisfied by all positive examples in D. Because $h \geq_g s$ h must also be satisfied by all positive examples in D.
- Similarly, by the definition of G, g cannot be satisfied by any negative example in D, and because $g \geq_g h$, h cannot be satisfied by any negative example in D.
- Because h is satisfied by all positive examples in D and by no negative examples in D, h is consistent with D, and therefore h is a member of $VS_{H,D}$. This proves step (1). The argument for (2) is a bit more complex. It can be proven by assuming some h in $VS_{H,D}$ that does not satisfy the right-hand side of the expression, then showing that this leads to an inconsistency.

1.7.4 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.

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

- And initializing the S boundary set to contain the most specific (least general)hypothesis.

$$S_0 \leftarrow \{\langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle\}$$

- These two boundary sets delimit the entire hypothesis space, because every other hypothesis in H is both more general than So and more specific than Go.

Algorithm:

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 another 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

1.7.5 An Illustrative Example.

The CANDIDATE-ELIMINATION algorithm applied to the first two training examples from Table 1.1.Figure 1.6 traces the CANDIDATE- ELIMINATION algorithm applied to the first two training examples from Table 1.1. As described above, the boundary sets are first initialized to Go and So, the most general and most specific hypotheses in H, respectively.

When the first training example is presented (a positive example in this case), the CANDIDATE-ELIMINATION Algorithm checks the *S* boundary and finds that it is overly specific-it fails to cover the positive example. The boundary is therefore revised by moving it to the least more general hypothesis that covers this new example. This revised boundary is shown as S_1 in Figure 1.6. No update of the G boundary is needed in response to this training example because Go correctly covers this example. When the second training example (also positive) is observed, it has a similar effect of generalizing ***S*** further to ***S2,*** leaving G again unchanged (i.e., G_2 = $\mathbf{G_1}$ = G_O).

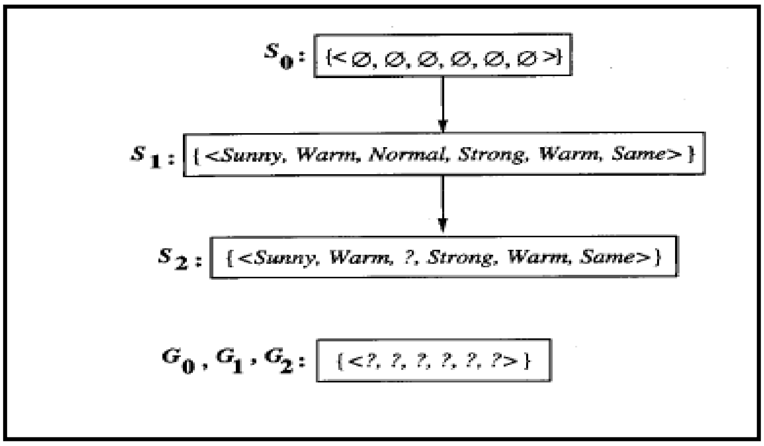


Figure 1.6: CANDIDATE-ELIMINATION Algorithm Trace1. S_o and G_o are the initial boundary sets corresponding to the most specific and most general hypotheses. Training examples 1 and 2 force the *S* boundary to become more general, as in the FIND-S algorithm. They have no effect on the *G* boundary.

Notice the processing of these first two positive examples is very similar to the processing performed by the FIND-S algorithm.

As illustrated by these first two steps, positive training examples may force the **S** boundary of the version space to become increasingly general. Negative training examples play the complimentary role of forcing the **G** boundary to become increasingly specific. Consider the third training example, shown in Figure 1.8.

This negative example reveals that the **G** boundary of the version space is overly general; that is, the hypothesis in **G** incorrectly predicts that this new example is a positive example. The hypothesis in the **G** boundary must therefore be specialized until it correctly classifies this new negative example. As shown in Figure 1.7, there are several alternative minimally more specific hypotheses. All of these become members of the new **G3** boundary set.

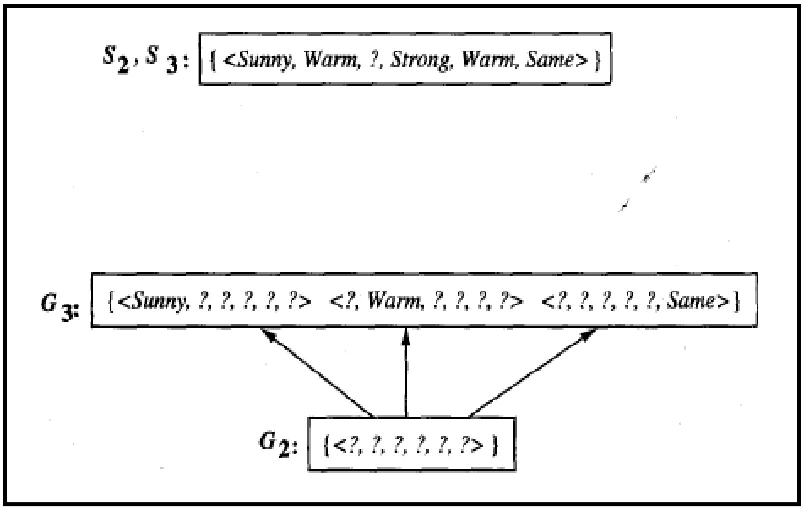


Figure 1.7: CANDIDATE-ELIMINATION Trace 2. Training example 3 is a negative example that forces the G_2 boundary to be specialized to G_3. Note several alternative maximally general hypotheses are included in G_3.

Given that there are six attributes that could be specified to specialize $\mathbf{G_2}$**,**why are there only three new hypotheses in $\mathbf{G_3}$**?** For example, the hypothesis ***h*** = (?, ?, ***Normal,*** ?, ?, ?) is a minimal specialization of $\mathbf{G_2}$ that correctly labels the new example as a negative example, but it is not included in $\mathbf{G_3}$**.** The reason this hypothesis is excluded is that it is inconsistent with the previously encountered positive examples. The algorithm determines this simply by noting that ***h*** is not more general than the current specific boundary, $\mathbf{S_2}$**.** In fact, the **S** boundary of the version space forms a summary of the previously encountered positive examples that can be used to determine whether any given hypothesis is consistent with these examples.

Any hypothesis more general than ***S*** will, by definition, cover any example that ***S*** covers and thus will cover any past positive example. In a dual fashion, the G boundary summarizes the information from previously encountered negative examples. Any hypothesis more specific than G is assured to be consistent with past negative examples. This is true because any such hypothesis, by definition, cannot cover examples that G does not cover.

The fourth training example, as shown in Figure 2.0, further generalizes the S boundary of the version space. It also results in removing one member of the G boundary, because this member fails to cover the new positive example. This last action results from the first step under the condition "If d is a positive example" shown in candidate elimination algorithm. To understand the rationale for this step, it is useful to consider why the offending hypothesis must be removed from G. Notice it cannot be specialized, because specializing it would not make it cover the new example. It also cannot be generalized, because by the definition of G, any more general hypothesis will cover at least one negative training example. Therefore, the hypothesis must be dropped from the G boundary, thereby removing an entire branch of the partial ordering from the version space of hypotheses remaining under consideration.

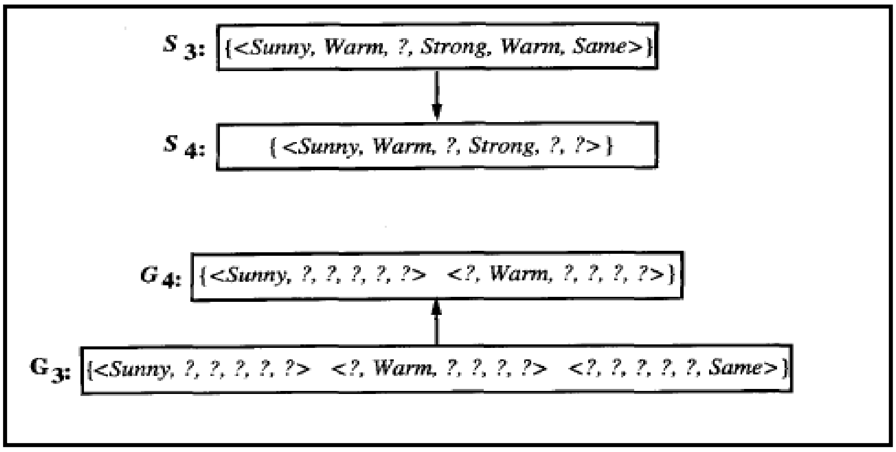


Figure 1.8: CANDIDATE-ELIMINATION Trace 3 . The positive training example generalizes the *S* boundary, from *S3* to *S4.* One member of *Gg* must also be deleted, because it is no longer more general than the *S4* boundary.

- No up-date of the G boundary is needed in response to this training example because Go correctly covers this example.
- When the second training example (also positive) is observed, it has a similar effect of generalizing S further to S2, leaving G again unchanged (i.e., G2 = GI = GO). Notice the

processing of these first two positive examples is very similar to the processing performed by the FIND-S algorithm.

- Negative training examples play the complimentary role of forcing the G boundary to become increasingly specific. Consider the third training example,
- The fourth training example, as shown in Figure 1.9, further generalizes the S boundary of the version space. It also results in removing one member of the G boundary, because this member fails to cover the new positive example. This last action results from the first step under the condition.
- After processing these four examples, the boundary setsS4 and G4 delimit the version space of all hypotheses consistent with the set of incrementally observed training examples. The entire version space, including those hypotheses bounded by S4 and G4, is shown in Figure 1.9
- This learned version space is independent of the sequence in which the training examples are presented (be-cause in the end it contains all hypotheses consistent with the set of examples).

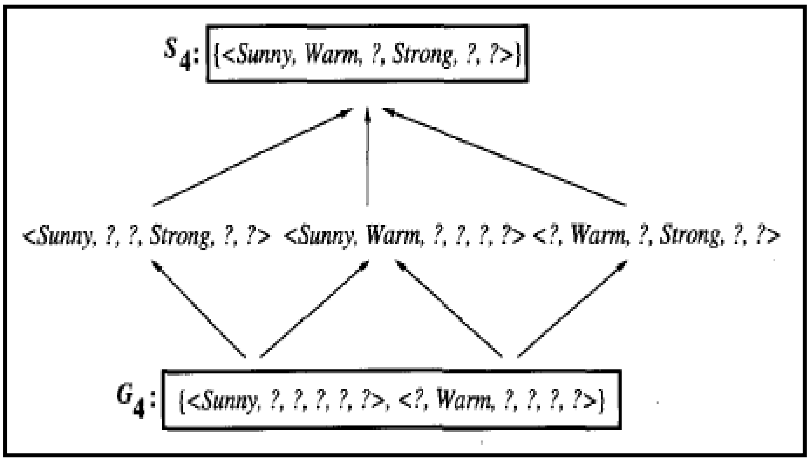


Figure 1.9: The final version space for the *EnjoySport* concept learning problem and training examples described earlier.

- As further training data is encountered, the S and G boundaries will move monotonically closer to each other, delimiting a smaller and smaller version space of candidate hypotheses.

1.8 REMARKS ON VERSION SPACES AND CANDIDATE-ELIMINATION

1.8.1 Will the CANDIDATE-ELIMINATION Algorithm Converge to the Correct Hypothesis?

The version space learned by the **CANDIDATE-ELIMINATION Algorithm** will converge toward the hypothesis that correctly describes the target concept, provided

(1) There are no errors in the training examples

(2) There is some hypothesis in H that correctly describes the target concept.

In fact, as new training examples are observed, the version space can be monitored to determine the remaining ambiguity regarding the true target concept and to determine when sufficient training examples have been observed to unambiguously identify the target concept. The target concept is exactly learned when the S and ***G*** boundary sets converge to a single, identical, hypothesis.

1.8.2 What Training Example Should the Learner Request Next?

Consider again the version space learned from the four training examples of the **Enjoysport** ***concept***. Clearly, the learner should attempt to discriminate among the alternative competing hypotheses in its current version space. Therefore, it should choose an instance that would be classified positive by some of these hypotheses, but negative by others.

One such instance is

(Sunny, Warm, Normal, Light, Warm, Same)

Note that this instance satisfies three of the six hypotheses in the current version space.

If the trainer classifies this instance as a positive example, the ***S*** boundary of the version space can then be generalized.

Alternatively, if the trainer indicates that this is a negative example, the G boundary can then be specialized. Either way, the learner will succeed in learning more about the true identity of the target concept, shrinking the version space from six hypotheses to half this number.

1.8.3 How Can Partially Learned Concepts Be Used?

Instance A was not among the training examples, it is classified as a positive instance by ***every*** hypothesis in the current version space .Because the hypotheses in the version space unanimously agree that this is a positive instance, the learner can classify instance A as positive with the same confidence it would have if it had already converged to the single ,correct target concept. Regardless of which hypothesis in the version space is eventually found to be the correct target concept, it is already clear that it will classify instance A as a positive example. Notice furthermore that we need not enumerate every hypothesis in the version space in order to test whether each classifies the instance as positive. This condition will be met if and only if the instance satisfies every member of S (why?). The reason is that every other hypothesis in the version space is at least as general as some member of S. By our definition of ***more-general_than i***f the new instance satisfies **all** members of S it must also satisfy each of these more general hypotheses.

- Similarly, instance B is classified as a negative instance by every hypothesis in the version space. This instance can therefore be safely classified as negative, given the partially learned concept. An efficient test for this condition is that the instance satisfies none of the members of G.

Table 1.2: New Instances to be classified

Instance	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
A	Sunny	Warm	Normal	Strong	Cool	Change	?
B	Rainy	Cold	Normal	Light	Warm	Same	?
C	Sunny	Warm	Normal	Light	Warm	Same	?
D	Sunny	Cold	Normal	Strong	Warm	Same	?

- Instance C presents a different situation. Half of the version space hypotheses classify it as positive and half classify it as negative. Thus, the learner cannot classify this example with confidence until further training examples are available. Notice that instance C is the same instance presented in the previous section as an optimal experimental query for the learner.
- Finally, instance D is classified as positive by two of the version space hypotheses and negative by the other four hypotheses. In this case we have less confidence in the classification than in the unambiguous cases of instances A and B. Still, the vote is in favor of a negative classification, and one approach we could take would be to output the majority vote, perhaps with a confidence rating indicating how close the vote was.

1.9 INDUCTIVE BIAS

1.9.1 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 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.

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	yes

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

S_2 : (?, ***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.

1.9.2 An Unbiased Learner

In the ***EnjoySport*** learning task, for example, the size of the instance space X of days described by the six available attributes is 96. How many possible concepts can be defined over this set of instances? In other words, how large is the power set of X? In general, the number of distinct subsets that can be defined over a set X containing 1x1 elements (i.e., the size of the power set of X) is 21'1. Thus, there are **296,** or approximately distinct target concepts that could be defined over this instance space and that our learner might be called upon to learn.

Let us reformulate the ***Enjoysport*** learning task in an unbiased way by defining a new hypothesis space ***H'*** that can represent every subset of instances; that is, let H' correspond to the power set of X. One way to define such an H' is to allow arbitrary disjunctions, conjunctions, and negations of our earlier hypotheses.
For instance, the target concept ***"Sky*** = ***Sunny*** or ***Sky*** = ***Cloudy"*** could then be described as

(Sunny, ?, ?, ?, ?, ?) v ***(Cloudy,*** ?, ?, ?, ?, ?)

Given this hypothesis space, we can safely use the CANDIDATE-ELIMINATION algorithm without worrying that the target concept might not be expressible. However, while this hypothesis space eliminates any problems of expressibility, it unfortunately raises a new, equally difficult problem: our concept learning algorithm is now completely unable to generalize beyond the observed examples! 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.

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

because this is the most specific possible hypothesis that covers these three examples. Similarly, the G boundary will consist of the hypothesis that rules out only the observed negative examples

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

It might at first seem that we could avoid this difficulty by simply using the partially learned version space and by taking a vote among the members of the version space

To see the reason, note that when H is the power set of X and **x** is some previously unobserved instance, then for any hypothesis h in the version space that covers **x,** there will be another hypothesis *h'* in the power set that is identical to h except for its classification of x. And of course if *h* is in the version space, then h' will be as well, because it agrees with *h* on all the observed training examples.

1.9.3 The Futility of Bias-Free Learning

Because inductive learning requires some form of prior assumptions, or inductive bias, we will find it useful to characterize different learning approaches by the inductive bias they employ. Let us define this notion of inductive bias more precisely. The key idea we wish to capture here is the policy by which the learner generalizes beyond the observed training data, to infer the classification of new instances. Therefore, consider the general setting in which an arbitrary learning algorithm L is provided an arbitrary set of training data D, = {(x, c(x))} of some arbitrary target concept c. After training, L is asked to classify a new instance xi. Let L(xi, D,) denote the classification (e.g., positive or negative) that L assigns to xi after learning from the training data D,. We can describe this inductive inference step performed by L as follows

$$(D_c \wedge x_i) > L(x_i, D_c)$$

where the notation *y* > z indicates that z is inductively inferred from *y*. For example, if we take L to be the CANDIDATE-ELIMINATION Algorithm, D, to be the training data, and x_i to be the first instance from Table 2.6,then the inductive inference performed in this case concludes that

$L(x_i, Dc) = (EnjoySport = yes)$

Because L is an inductive learning algorithm, the result L(xi, D,) that it infers will not in general be provably correct; that is, the classification L(xi, D,) need not follow deductively from the training data D, and the description of the new instance xi. However, it is interesting to ask what additional assumptions could be added to D, r\xi so that L(xi, D,) would follow deductively. We define the inductive bias of ***L*** as this set of additional assumptions. More precisely, we define the inductive bias of ***L*** to be the set of assumptions ***B*** such that for all new instances

xi (B A ***D, A xi)*** ├ ***L(xi, D,)***

where the notation ***y*** ├ *z* indicates that *z* follows deductively from y (i.e., that z is provable from ***y).*** Thus, we define the inductive bias of a learner as the set of additional assumptions B sufficient to justify its inductive inferences as deductive inferences.

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,*** = ***((x , c(x))}*** be an arbitrary set of training examples of c. Let ***L(xi, Dc)*** denote the classification assigned to the instance ***xi*** by ***L*** after training on the data ***Dc.*** The **inductive bias** of ***L*** is any minimal set of assertions ***B*** such that for any target concept c and corresponding training examples ***Dc***

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

****