

CHAPTER 2

CONCEPT LEARNING AND THE GENERAL-TO-SPECIFIC ORDERING

The problem of inducing general functions from specific training examples is central to learning. This chapter considers concept learning: acquiring the definition of a general category given a sample of positive and negative training examples of the category. Concept learning can be formulated as a problem of searching through a predefined space of potential hypotheses for the hypothesis that best fits the training examples. In many cases this search can be efficiently organized by taking advantage of a naturally occurring structure over the hypothesis space—a general-to-specific ordering of hypotheses. This chapter presents several learning algorithms and considers situations under which they converge to the correct hypothesis. We also examine the nature of inductive learning and the justification by which any program may successfully generalize beyond the observed training data.

2.1 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,” “situations in which I should study more in order to pass the exam,” etc. Each such concept can be viewed as describing some subset of objects or events defined over a larger set (e.g., the subset of animals that constitute

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

In this chapter we consider the problem of automatically inferring the general definition of some concept, given examples labeled as members or nonmembers of the concept. This task is commonly referred to as *concept learning*, or approximating a boolean-valued function from examples.

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

2.2 A CONCEPT LEARNING TASK

To ground our discussion of concept learning, consider the example task of learning the target concept “days on which my friend Aldo enjoys his favorite water sport.” Table 2.1 describes a set of example days, each represented by a set of *attributes*. The attribute *EnjoySport* indicates whether or not Aldo enjoys his favorite water sport on this day. The task is to learn to predict the value of *EnjoySport* for an arbitrary day, based on the values of its other attributes.

What hypothesis representation shall we provide to the learner in this case? 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 “∅” 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	<i>Sky</i>	<i>AirTemp</i>	<i>Humidity</i>	<i>Wind</i>	<i>Water</i>	<i>Forecast</i>	<i>EnjoySport</i>
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

$$(\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$$

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. The definition of the *EnjoySport* concept learning task in this general form is given in Table 2.2.

2.2.1 Notation

Throughout this book, we employ the following terminology when discussing concept learning problems. The set of items over which the concept is defined is called the set of *instances*, which we denote by X . In the current example, X is the set of all possible days, each represented by the attributes *Sky*, *AirTemp*, *Humidity*, *Wind*, *Water*, and *Forecast*. The concept or function to be learned is called the *target concept*, which we denote by c . In general, c can be any boolean-valued function defined over the instances X ; that is, $c : X \rightarrow \{0, 1\}$. In the current example, the target concept corresponds to the value of the attribute *EnjoySport* (i.e., $c(x) = 1$ if *EnjoySport* = *Yes*, and $c(x) = 0$ if *EnjoySport* = *No*).

• 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), “ \emptyset ” (no value is acceptable), or a specific value.
- Target concept c : *EnjoySport* : $X \rightarrow \{0, 1\}$
- Training examples D : Positive and negative examples of the target function (see Table 2.1).

• Determine:

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

TABLE 2.2

The *EnjoySport* concept learning task.

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 2.1). 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 nonmembers of the target concept. We will often write the ordered pair $\langle x, c(x) \rangle$ 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, $h : X \rightarrow \{0, 1\}$. The goal of the learner is to find a hypothesis h such that $h(x) = c(x)$ for all x in X .

2.2.2 The Inductive Learning Hypothesis

Notice that although the learning task is to determine a hypothesis h identical to the target concept c over the entire set of instances X , the only information available about c is its value over the training examples. Therefore, inductive learning algorithms can at best guarantee that the output hypothesis fits the target concept over the training data. Lacking any further information, our assumption is that the best hypothesis regarding unseen instances is the hypothesis that best fits the observed training data. This is the fundamental assumption of inductive learning, and we will have much more to say about it throughout this book. We state it here informally and will revisit and analyze this assumption more formally and more quantitatively in Chapters 5, 6, and 7.

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

2.3 CONCEPT LEARNING AS SEARCH

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

$3 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 = 96$ distinct instances. A similar calculation shows that there are $5 \cdot 4 \cdot 4 \cdot 4 \cdot 4 \cdot 4 = 5120$ *syntactically distinct* hypotheses within H . Notice, however, that every hypothesis containing one or more “ \emptyset ” symbols represents the empty set of instances; that is, it classifies every instance as negative. Therefore, the number of *semantically distinct* hypotheses is only $1 + (4 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3) = 973$. Our *EnjoySport* example is a very simple learning task, with a relatively small, finite hypothesis space. Most practical learning tasks involve much larger, sometimes infinite, hypothesis spaces.

If we view learning as a search problem, then it is natural that our study of learning algorithms will examine different strategies for searching the hypothesis space. We will be particularly interested in algorithms capable of efficiently searching very large or infinite hypothesis spaces, to find the hypotheses that best fit the training data.

2.3.1 General-to-Specific Ordering of Hypotheses

Many algorithms for concept learning organize the search through the hypothesis space by relying on a very useful structure that exists for any concept learning problem: 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

$$h_1 = \langle \text{Sunny}, ?, ?, \text{Strong}, ?, ? \rangle$$

$$h_2 = \langle \text{Sunny}, ?, ?, ?, ?, ? \rangle$$

Now consider the sets of instances that are classified positive by h_1 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_1 will also be classified positive by h_2 . Therefore, we say that h_2 is more general than h_1 .

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* relation 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-or-equal-to* h_k if and only if any instance that satisfies h_k also satisfies h_j .

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

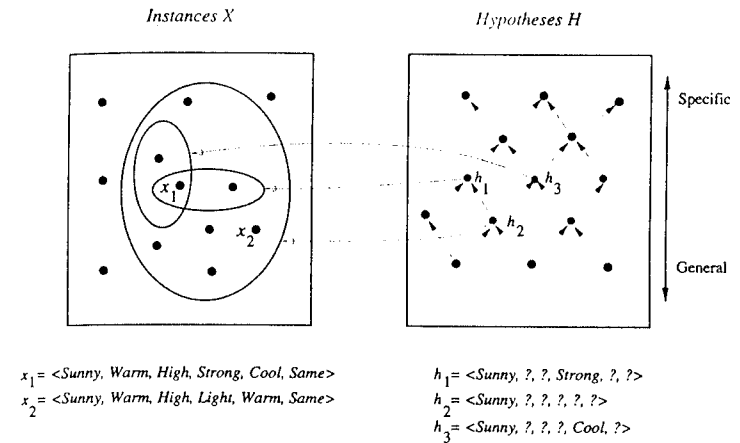


FIGURE 2.1

Instances, hypotheses, and the *more-general-than* 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 *more-general-than* relation, with the arrow pointing toward the less general hypothesis. Note the subset of instances characterized by h_2 subsumes the subset characterized by h_1 , hence h_2 is *more-general-than* h_1 .

h_k (written $h_j >_g h_k$) if and only if $(h_j \geq_g h_k) \wedge (h_k \not\geq_g h_j)$. Finally, we will sometimes find the inverse useful and will say that h_j is *more-specific-than* h_k when h_k is *more-general-than* h_j .

To illustrate these definitions, consider the three hypotheses h_1 , h_2 , and h_3 from our *EnjoySport* example, shown in Figure 2.1. How are these three hypotheses related by the \geq_g relation? As noted earlier, hypothesis h_2 is more general than h_1 because every instance that satisfies h_1 also satisfies h_2 . Similarly, h_2 is more general than h_3 . Note that neither h_1 nor h_3 is more general than the other; although the instances satisfied by these two hypotheses intersect, neither set subsumes the other. Notice also that the \geq_g and $>_g$ 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 \geq_g 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 h_1 and h_3 , such that $h_1 \not\geq_g h_3$ and $h_3 \not\geq_g h_1$.

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. 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.

2.4 FIND-S: FINDING A MAXIMALLY SPECIFIC HYPOTHESIS

How can we use the *more_general_than* partial ordering to organize the search for a hypothesis consistent with the observed training examples? One way is to begin with the most specific possible hypothesis in H , then generalize this hypothesis each time it fails to cover an observed positive training example. (We say that a hypothesis “covers” a positive example if it correctly classifies the example as positive.) To be more precise about how the partial ordering is used, consider the FIND-S algorithm defined in Table 2.3.

To illustrate this algorithm, assume the learner is given the sequence of training examples from Table 2.1 for the *EnjoySport* task. The first step of FIND-S is to initialize h to the most specific hypothesis in H

$$h \leftarrow \langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$$

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 “ \emptyset ” 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 values for this training example.

$$h \leftarrow \langle \text{Sunny, Warm, Normal, Strong, Warm, Same} \rangle$$

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

$$h \leftarrow \langle \text{Sunny, Warm, ?, Strong, Warm, Same} \rangle$$

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!* While this may at first seem strange, notice that in the current case our hypothesis h is already consistent with the new negative example (i.e., h correctly classifies this example as negative), and hence no revision

is needed. In the general case, as long as we assume that the hypothesis space H contains a hypothesis that describes the true target concept c and that the training data contains no errors, then the current hypothesis h can never require a revision in response to a negative example. To see why, recall that the current hypothesis h is the most specific hypothesis in H consistent with the observed positive examples. Because the target concept c is also assumed to be in H and to be consistent with the positive training examples, c must be *more_general_than_or_equal_to* h . But the target concept c will never cover a negative example, thus neither will h (by the definition of *more_general_than*). Therefore, no revision to h will be required in response to any negative example.

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

$$h \leftarrow \langle \text{Sunny, Warm, ?, Strong, ?, ?} \rangle$$

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 2.2 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). The literature on concept learning is

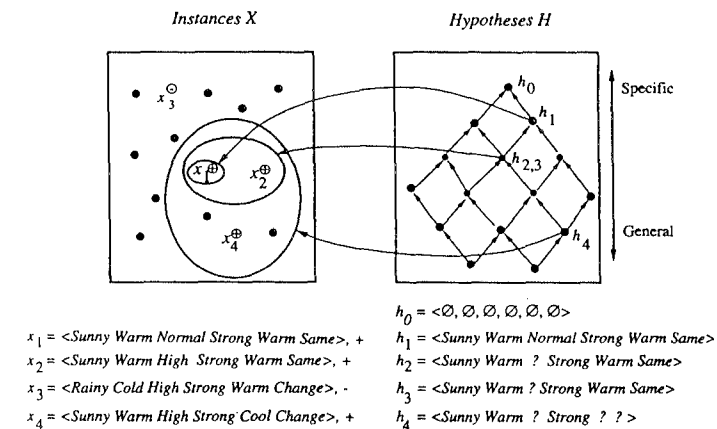


FIGURE 2.2
The hypothesis space search performed by FIND-S. The search begins (h_0) with the most specific hypothesis in H , then considers increasingly general hypotheses (h_1 through h_4) as mandated by the training examples. In the instance space diagram, positive training examples are denoted by “+,” negative by “-,” and instances that have not been presented as training examples by a solid circle.

populated by many different algorithms that utilize this same *more-general-than* partial ordering to organize the search in one fashion or another. A number of such algorithms are discussed in this chapter, and several others are presented in Chapter 10.

The key property of the FIND-S algorithm is that for hypothesis spaces described by conjunctions of attribute constraints (such as H for the *EnjoySport* task), FIND-S is guaranteed to output the most specific hypothesis within H that is consistent with the positive training examples. Its final hypothesis will also be consistent with the negative examples provided the correct target concept is contained in H , and provided the training examples are correct. However, there are several questions still left unanswered by this learning algorithm, such as:

- 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. We would prefer a learning algorithm that could determine whether it had converged and, if not, at least characterize its uncertainty regarding the true identity of the target concept.
- 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. We would prefer an algorithm that could at least detect when the training data is inconsistent and, preferably, accommodate such errors.
- What if there are several maximally specific consistent hypotheses? In the hypothesis language H for the *EnjoySport* task, there is always a unique, most specific hypothesis consistent with any set of positive examples. However, for other hypothesis spaces (discussed later) 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 of how to generalize the hypothesis, to accommodate the possibility that the target concept lies along a different branch of the partial ordering than the branch it has selected. Furthermore, we can define hypothesis spaces for which there is no maximally specific consistent hypothesis, although this is more of a theoretical issue than a practical one (see Exercise 2.7).

2.5 VERSION SPACES AND THE CANDIDATE-ELIMINATION ALGORITHM

This section describes a second approach to concept learning, the CANDIDATE-ELIMINATION algorithm, that addresses several of the limitations of FIND-S. Notice that although FIND-S outputs a hypothesis from H that is consistent with the training examples, this is just one of many hypotheses from H that might fit the training data equally well. The key idea in the CANDIDATE-ELIMINATION algorithm is to output a description of the set of *all hypotheses consistent with the training examples*. Surprisingly, the CANDIDATE-ELIMINATION algorithm computes the description of this set without explicitly enumerating all of its members. This is accomplished by again using the *more-general-than* partial ordering, this time to maintain a compact representation of the set of consistent hypotheses and to incrementally refine this representation as each new training example is encountered.

The CANDIDATE-ELIMINATION algorithm has been applied to problems such as learning regularities in chemical mass spectroscopy (Mitchell 1979) and learning control rules for heuristic search (Mitchell et al. 1983). Nevertheless, practical applications of the CANDIDATE-ELIMINATION and FIND-S algorithms are limited by the fact that they both perform poorly when given noisy training data. More importantly for our purposes here, the CANDIDATE-ELIMINATION algorithm provides a useful conceptual framework for introducing several fundamental issues in machine learning. In the remainder of this chapter we present the algorithm and discuss these issues. Beginning with the next chapter, we will examine learning algorithms that are used more frequently with noisy training data.

2.5.1 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 $\langle x, c(x) \rangle$ in D .

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

Notice the key difference between this definition of *consistent* and our earlier definition of *satisfies*. 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 $VS_{H,D}$, with respect to hypothesis space H and training examples D , is the subset of hypotheses from H consistent with the training examples in D .

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

2.5.2 The LIST-THEN-ELIMINATE Algorithm

One obvious way to represent the version space is simply to list all of its members. This leads to a simple learning algorithm, which we might call the LIST-THEN-ELIMINATE algorithm, defined in Table 2.4.

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. This, presumably, is the desired target concept. If insufficient data is available to narrow the version space to a single hypothesis, then the algorithm can output the entire set of hypotheses consistent with the observed data.

In principle, 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. Unfortunately, it requires exhaustively enumerating all hypotheses in H —an unrealistic requirement for all but the most trivial hypothesis spaces.

2.5.3 A More Compact Representation for Version Spaces

The CANDIDATE-ELIMINATION algorithm works on the same principle as the above LIST-THEN-ELIMINATE algorithm. However, it employs a much more compact representation of the version space. In particular, the version space is represented by its most general and least general members. These members form general and specific boundary sets that delimit the version space within the partially ordered hypothesis space.

The LIST-THEN-ELIMINATE Algorithm

1. *VersionSpace* \leftarrow a list containing every hypothesis in H
2. For each training example, $(x, c(x))$
 remove from *VersionSpace* any hypothesis h for which $h(x) \neq c(x)$
3. Output the list of hypotheses in *VersionSpace*

TABLE 2.4

The LIST-THEN-ELIMINATE algorithm.

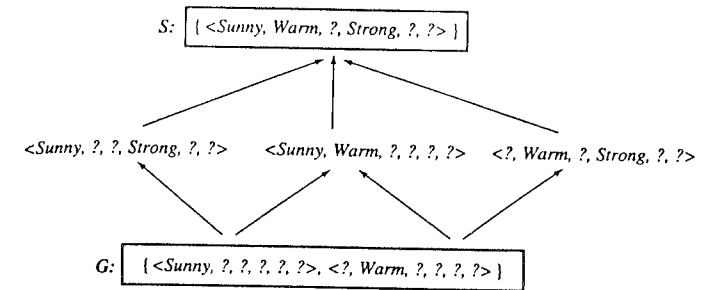


FIGURE 2.3

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.

To illustrate this representation for version spaces, consider again the *EnjoySport* concept learning problem described in Table 2.2. Recall that given the four training examples from Table 2.1, FIND-S outputs the hypothesis

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

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 2.3. They constitute the version space relative to this set of data and this hypothesis representation. The arrows among these six hypotheses in Figure 2.3 indicate instances of the *more-general-than* relation. The CANDIDATE-ELIMINATION algorithm represents the version space by storing only its most general members (labeled G in Figure 2.3) 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.

It is intuitively plausible that we can represent the version space in terms of its most specific and most general members. Below we define the boundary sets G and S precisely and prove that these sets do in fact represent the version space.

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 .

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

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 .

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

As long as the sets G and S are well defined (see Exercise 2.7), they completely specify the version space. In particular, we can show that the version space is precisely the set of hypotheses contained in G , plus those contained in S , plus those that lie between G and S in the partially ordered hypothesis space. This is stated precisely in Theorem 2.1.

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

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

Proof. To prove the theorem it suffices to show that (1) every h satisfying the right-hand side of the above expression is in $VS_{H,D}$ and (2) every member of $VS_{H,D}$ 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. (See Exercise 2.6.) \square

2.5.4 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

$$G_0 \leftarrow \{(\?, \?, \?, \?, \?, \?)\}$$

and initializing the S boundary set to contain the most specific (least general) hypothesis

$$S_0 \leftarrow \{(\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset)\}$$

These two boundary sets delimit the entire hypothesis space, because every other hypothesis in H is both more general than S_0 and more specific than G_0 . 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. After all examples have been processed, the computed version space contains all the hypotheses consistent with these examples and only these hypotheses. This algorithm is summarized in Table 2.5.

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

TABLE 2.5

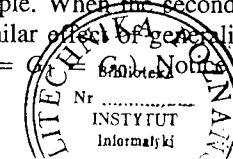
CANDIDATE-ELIMINATION algorithm using version spaces. Notice the duality in how positive and negative examples influence S and G .

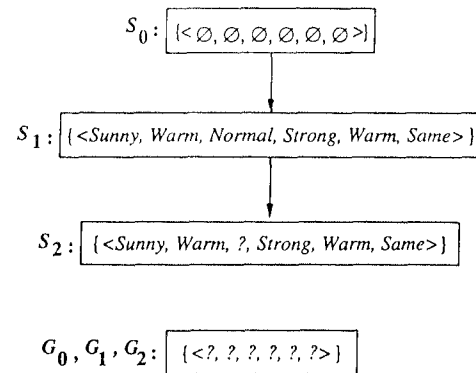
Notice that the algorithm is specified in terms of operations such as computing minimal generalizations and specializations of given hypotheses, and identifying nonminimal and nonmaximal hypotheses. The detailed implementation of these operations will depend, of course, on the specific representations for instances and hypotheses. However, the algorithm itself can be applied to any concept learning task and hypothesis space for which these operations are well-defined. In the following example trace of this algorithm, we see how such operations can be implemented for the representations used in the *EnjoySport* example problem.

2.5.5 An Illustrative Example

Figure 2.4 traces the CANDIDATE-ELIMINATION algorithm applied to the first two training examples from Table 2.1. As described above, the boundary sets are first initialized to G_0 and S_0 , 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 2.4. No update of the G boundary is needed in response to this training example because G_0 correctly covers this example. When the second training example (also positive) is observed, it has a similar effect of generalizing S further to S_2 , leaving G again unchanged (i.e., $G_2 = G_0$). Notice the processing of these first





Training examples:

1. $\langle \text{Sunny}, \text{Warm}, \text{Normal}, \text{Strong}, \text{Warm}, \text{Same} \rangle, \text{Enjoy Sport} = \text{Yes}$
2. $\langle \text{Sunny}, \text{Warm}, \text{High}, \text{Strong}, \text{Warm}, \text{Same} \rangle, \text{Enjoy Sport} = \text{Yes}$

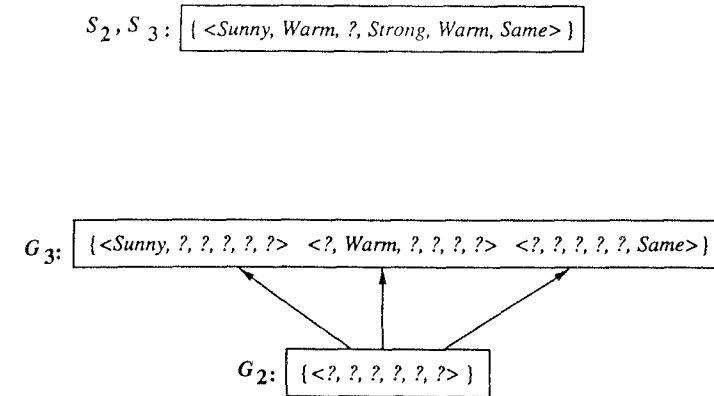
FIGURE 2.4

CANDIDATE-ELIMINATION Trace 1. S_0 and G_0 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.

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 2.5. 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 2.5, there are several alternative minimally more specific hypotheses. All of these become members of the new G_3 boundary set.

Given that there are six attributes that could be specified to specialize G_2 , why are there only three new hypotheses in G_3 ? For example, the hypothesis $h = \{?, ?, \text{Normal}, ?, ?, ?\}$ is a minimal specialization of G_2 that correctly labels the new example as a negative example, but it is not included in 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, 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



Training Example:

3. $\langle \text{Rainy}, \text{Cold}, \text{High}, \text{Strong}, \text{Warm}, \text{Change} \rangle, \text{EnjoySport} = \text{No}$

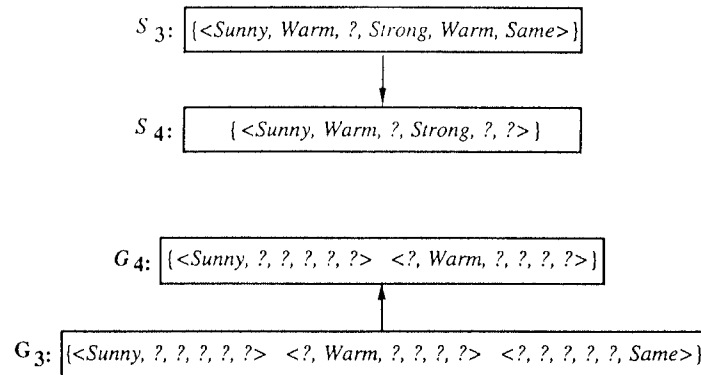
FIGURE 2.5

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 .

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 2.6, 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" in the algorithm shown in Table 2.5. 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.

After processing these four examples, the boundary sets S_4 and G_4 delimit the version space of *all* hypotheses consistent with the set of incrementally observed training examples. The entire version space, including those hypotheses



Training Example:

4. <Sunny, Warm, High, Strong, Cool, Change>, EnjoySport = Yes

FIGURE 2.6 CANDIDATE-ELIMINATION Trace 3. The positive training example generalizes the S boundary, from S_3 to S_4 . One member of G_3 must also be deleted, because it is no longer more general than the S_4 boundary.

bounded by S_4 and G_4 , is shown in Figure 2.7. This learned version space is independent of the sequence in which the training examples are presented (because in the end it contains all hypotheses consistent with the set of examples). 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.

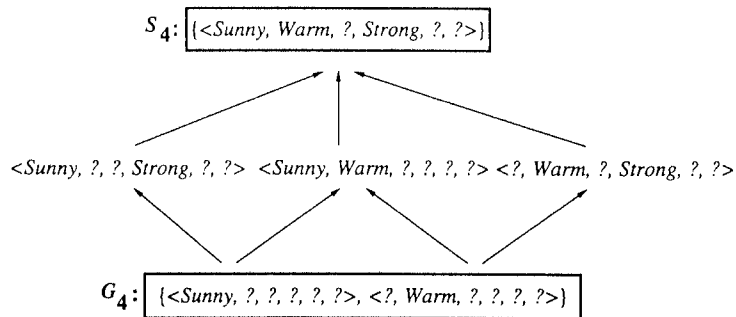


FIGURE 2.7 The final version space for the *EnjoySport* concept learning problem and training examples described earlier.

2.6 REMARKS ON VERSION SPACES AND CANDIDATE-ELIMINATION

2.6.1 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, and (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.

What will happen if the training data contains errors? Suppose, for example, that the second training example above is incorrectly presented as a negative example instead of a positive example. Unfortunately, in this case the algorithm is certain to remove the correct target concept from the version space! Because it will remove every hypothesis that is inconsistent with each training example, it will eliminate the true target concept from the version space as soon as this false negative example is encountered. Of course, given sufficient additional training data the learner will eventually detect an inconsistency by noticing that the S and G boundary sets eventually converge to an empty version space. Such an empty version space indicates that there is *no* hypothesis in H consistent with all observed training examples. A similar symptom will appear when the training examples are correct, but the target concept cannot be described in the hypothesis representation (e.g., if the target concept is a disjunction of feature attributes and the hypothesis space supports only conjunctive descriptions). We will consider such eventualities in greater detail later. For now, we consider only the case in which the training examples are correct and the true target concept is present in the hypothesis space.

2.6.2 What Training Example Should the Learner Request Next?

Up to this point we have assumed that training examples are provided to the learner by some external teacher. Suppose instead that the learner is allowed to conduct experiments in which it chooses the next instance, then obtains the correct classification for this instance from an external oracle (e.g., nature or a teacher). This scenario covers situations in which the learner may conduct experiments in nature (e.g., build new bridges and allow nature to classify them as stable or unstable), or in which a teacher is available to provide the correct classification (e.g., propose a new bridge and allow the teacher to suggest whether or not it will be stable). We use the term *query* to refer to such instances constructed by the learner, which are then classified by an external oracle.

Consider again the version space learned from the four training examples of the *EnjoySport* concept and illustrated in Figure 2.3. What would be a good query for the learner to pose at this point? What is a good query strategy in

general? 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 (Figure 2.3). 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.

In general, the optimal query strategy for a concept learner is to generate instances that satisfy exactly half the hypotheses in the current version space. When this is possible, the size of the version space is reduced by half with each new example, and the correct target concept can therefore be found with only $\lceil \log_2 |VS| \rceil$ experiments. The situation is analogous to playing the game twenty questions, in which the goal is to ask yes-no questions to determine the correct hypothesis. The optimal strategy for playing twenty questions is to ask questions that evenly split the candidate hypotheses into sets that predict yes and no. While we have seen that it is possible to generate an instance that satisfies precisely half the hypotheses in the version space of Figure 2.3, in general it may not be possible to construct an instance that matches precisely half the hypotheses. In such cases, a larger number of queries may be required than $\lceil \log_2 |VS| \rceil$.

2.6.3 How Can Partially Learned Concepts Be Used?

Suppose that no additional training examples are available beyond the four in our example above, but that the learner is now required to classify new instances that it has not yet observed. Even though the version space of Figure 2.3 still contains multiple hypotheses, indicating that the target concept has not yet been fully learned, it is possible to classify certain examples with the same degree of confidence as if the target concept had been uniquely identified. To illustrate, suppose the learner is asked to classify the four new instances shown in Table 2.6.

Note that although instance A was not among the training examples, it is classified as a positive instance by *every* hypothesis in the current version space (shown in Figure 2.3). 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

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	?

TABLE 2.6
New instances to be classified.

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*, if 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 (why?).

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. This is to be expected, because those instances whose classification is most ambiguous are precisely the instances whose true classification would provide the most new information for refining the version space.

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. As we will discuss in Chapter 6, if we assume that all hypotheses in H are equally probable a priori, then such a vote provides the most probable classification of this new instance. Furthermore, the proportion of hypotheses voting positive can be interpreted as the probability that this instance is positive given the training data.

2.7 INDUCTIVE BIAS

As discussed above, 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. What if the target concept is not contained in the hypothesis space? Can we avoid this difficulty by using a hypothesis space that includes every possible hypothesis? How does the

size of this hypothesis space influence the ability of the algorithm to generalize to unobserved instances? How does the size of the hypothesis space influence the number of training examples that must be observed? These are fundamental questions for inductive inference in general. Here we examine them in the context of the CANDIDATE-ELIMINATION algorithm. As we shall see, though, the conclusions we draw from this analysis will apply to *any* concept learning system that outputs *any* hypothesis consistent with the training data.

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 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	<i>Sky</i>	<i>AirTemp</i>	<i>Humidity</i>	<i>Wind</i>	<i>Water</i>	<i>Forecast</i>	<i>EnjoySport</i>
1	Sunny	Warm	Normal	Strong	Cool	Change	Yes
2	Cloudy	Warm	Normal	Strong	Cool	Change	Yes
3	Rainy	Warm	Normal	Strong	Cool	Change	No

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

$$S_2 : \langle ?, \text{Warm}, \text{Normal}, \text{Strong}, \text{Cool}, \text{Change} \rangle$$

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.

2.7.2 An Unbiased Learner

The obvious solution to the problem of assuring that the target concept is in the hypothesis space *H* is to provide a hypothesis space capable of representing *every teachable concept*; that is, it is capable of representing every possible subset of the instances *X*. In general, the set of all subsets of a set *X* is called the *power set* of *X*.

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 $|X|$ elements (i.e., the size of the power set of *X*) is $2^{|X|}$. Thus, there are 2^{96} , or approximately 10^{28} distinct target concepts that could be defined over this instance space and that our learner might be called upon to learn. Recall from Section 2.3 that our conjunctive hypothesis space is able to represent only 973 of these—a very biased hypothesis space indeed!

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

$$\langle \text{Sunny}, ?, ?, ?, ?, ? \rangle \vee \langle \text{Cloudy}, ?, ?, ?, ?, ? \rangle$$

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 (x_1, x_2, x_3) and two negative examples (x_4, x_5) 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

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

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

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

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!

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 as discussed in Section 2.6.3. Unfortunately, the only instances that will produce a unanimous vote are the previously observed training examples. For all the other instances, taking a vote will be futile: each unobserved instance will be classified positive by *precisely half* the hypotheses in the version space and will be classified negative by the other half (why?). 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.

2.7.3 The Futility of Bias-Free Learning

The above discussion illustrates a fundamental property of inductive inference: *a learner that makes no a priori assumptions regarding the identity of the target concept has no rational basis for classifying any unseen instances.* In fact, the only reason that the CANDIDATE-ELIMINATION algorithm was able to generalize beyond the observed training examples in our original formulation of the *EnjoySport* task is that it was biased by the implicit assumption that the target concept could be represented by a conjunction of attribute values. In cases where this assumption is correct (and the training examples are error-free), its classification of new instances will also be correct. If this assumption is incorrect, however, it is certain that the CANDIDATE-ELIMINATION algorithm will misclassify at least some instances from X .

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 $D_c = \{(x, c(x))\}$ of some arbitrary target concept c . After training, L is asked to classify a new instance x_i . Let $L(x_i, D_c)$ denote the classification (e.g., positive or negative) that L assigns to x_i after learning from the training data D_c . We can describe this inductive inference step performed by L as follows

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

where the notation $y \succ z$ indicates that z is inductively inferred from y . For example, if we take L to be the CANDIDATE-ELIMINATION algorithm, D_c to be the training data from Table 2.1, and x_i to be the first instance from Table 2.6, then the inductive inference performed in this case concludes that $L(x_i, D_c) = (\text{EnjoySport} = \text{yes})$.

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

[†]The term *inductive bias* here is not to be confused with the term *estimation bias* commonly used in statistics. Estimation bias will be discussed in Chapter 5.

inductive bias of L to be the set of assumptions B such that for all new instances x_i

$$(B \wedge D_c \wedge x_i) \vdash L(x_i, D_c)$$

where the notation $y \vdash z$ 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. To summarize,

Definition: Consider a concept learning algorithm L for the set of instances X . Let c be an arbitrary concept defined over X , and let $D_c = \{(x, c(x))\}$ 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

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

What, then, is the inductive bias of the CANDIDATE-ELIMINATION algorithm? To answer this, let us specify $L(x_i, D_c)$ exactly for this algorithm: given a set of data D_c , the CANDIDATE-ELIMINATION algorithm will first compute the version space VS_{H, D_c} , then classify the new instance x_i by a vote among hypotheses in this version space. Here let us assume that it will output a classification for x_i only if this vote among version space hypotheses is unanimously positive or negative and that it will not output a classification otherwise. Given this definition of $L(x_i, D_c)$ for the CANDIDATE-ELIMINATION algorithm, what is its inductive bias? It is simply the assumption $c \in H$. Given this assumption, each inductive inference performed by the CANDIDATE-ELIMINATION algorithm can be justified deductively.

To see why the classification $L(x_i, D_c)$ follows deductively from $B = \{c \in H\}$, together with the data D_c and description of the instance x_i , consider the following argument. First, notice that if we assume $c \in H$ then it follows deductively that $c \in VS_{H, D_c}$. This follows from $c \in H$, from the definition of the version space VS_{H, D_c} as the set of all hypotheses in H that are consistent with D_c , and from our definition of $D_c = \{(x, c(x))\}$ as training data consistent with the target concept c . Second, recall that we defined the classification $L(x_i, D_c)$ to be the unanimous vote of all hypotheses in the version space. Thus, if L outputs the classification $L(x_i, D_c)$, it must be the case the every hypothesis in VS_{H, D_c} also produces this classification, including the hypothesis $c \in VS_{H, D_c}$. Therefore $c(x_i) = L(x_i, D_c)$. To summarize, the CANDIDATE-ELIMINATION algorithm defined in this fashion can be characterized by the following bias

Inductive bias of CANDIDATE-ELIMINATION algorithm. The target concept c is contained in the given hypothesis space H .

Figure 2.8 summarizes the situation schematically. The inductive CANDIDATE-ELIMINATION algorithm at the top of the figure takes two inputs: the training examples and a new instance to be classified. At the bottom of the figure, a deductive

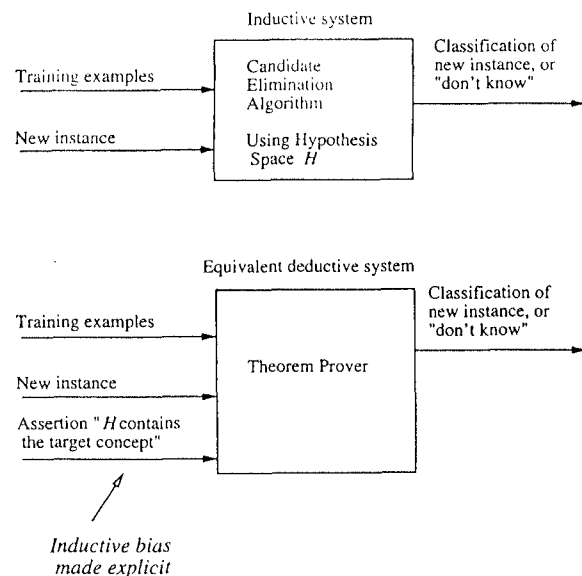


FIGURE 2.8

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 assertion " 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 a way to compare inductive systems according to their policies for generalizing beyond the observed training data.

theorem prover is given these same two inputs plus the assertion " H contains the target concept." These two systems will in principle produce identical outputs for every possible input set of training examples and every possible new instance in X . Of course the inductive bias that is explicitly input to the theorem prover is only implicit in the code of the CANDIDATE-ELIMINATION algorithm. In a sense, it exists only in the eye of us beholders. Nevertheless, it is a perfectly well-defined set of assertions.

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.

1. **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.

2. **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.
3. **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: that all instances are negative instances unless the opposite is entailed by its other knowledge.[†]

As we examine other inductive inference methods, it is useful to keep in mind this means of characterizing them and the strength of their inductive bias. More strongly biased methods make more inductive leaps, classifying a greater proportion of unseen instances. Some inductive biases correspond to categorical assumptions that completely rule out certain concepts, such as the bias "the hypothesis space H includes the target concept." Other inductive biases merely rank order the hypotheses by stating preferences such as "more specific hypotheses are preferred over more general hypotheses." Some biases are implicit in the learner and are unchangeable by the learner, such as the ones we have considered here. In Chapters 11 and 12 we will see other systems whose bias is made explicit as a set of assertions represented and manipulated by the learner.

2.8 SUMMARY AND FURTHER READING

The main points of this chapter include:

- Concept learning can be cast as a problem of searching through a large predefined space of potential hypotheses.
- The general-to-specific partial ordering of hypotheses, which can be defined for any concept learning problem, provides a useful structure for organizing the search through the hypothesis space.

[†]Notice this last inductive bias assumption involves a kind of default, or nonmonotonic reasoning.

- The FIND-S algorithm utilizes this general-to-specific ordering, performing a specific-to-general search through the hypothesis space along one branch of the partial ordering, to find the most specific hypothesis consistent with the training examples.
- The CANDIDATE-ELIMINATION algorithm utilizes this general-to-specific ordering to compute the version space (the set of all hypotheses consistent with the training data) by incrementally computing the sets of maximally specific (S) and maximally general (G) hypotheses.
- Because the S and G sets delimit the entire set of hypotheses consistent with the data, they provide the learner with a description of its uncertainty regarding the exact identity of the target concept. This version space of alternative hypotheses can be examined to determine whether the learner has converged to the target concept, to determine when the training data are inconsistent, to generate informative queries to further refine the version space, and to determine which unseen instances can be unambiguously classified based on the partially learned concept.
- Version spaces and the CANDIDATE-ELIMINATION algorithm provide a useful conceptual framework for studying concept learning. However, this learning algorithm is not robust to noisy data or to situations in which the unknown target concept is not expressible in the provided hypothesis space. Chapter 10 describes several concept learning algorithms based on the general-to-specific ordering, which are robust to noisy data.
- Inductive learning algorithms are able to classify unseen examples only because of their implicit inductive bias for selecting one consistent hypothesis over another. The bias associated with the CANDIDATE-ELIMINATION algorithm is that the target concept can be found in the provided hypothesis space ($c \in H$). The output hypotheses and classifications of subsequent instances follow *deductively* from this assumption together with the observed training data.
- If the hypothesis space is enriched to the point where there is a hypothesis corresponding to every possible subset of instances (the power set of the instances), this will remove any inductive bias from the CANDIDATE-ELIMINATION algorithm. Unfortunately, this also removes the ability to classify any instance beyond the observed training examples. An unbiased learner cannot make inductive leaps to classify unseen examples.

The idea of concept learning and using the general-to-specific ordering have been studied for quite some time. Bruner et al. (1957) provided an early study of concept learning in humans, and Hunt and Hovland (1963) an early effort to automate it. Winston's (1970) widely known Ph.D. dissertation cast concept learning as a search involving generalization and specialization operators. Plotkin (1970, 1971) provided an early formalization of the *more-general-than* relation, as well as the related notion of θ -subsumption (discussed in Chapter 10). Simon and Lea (1973) give an early account of learning as search through a hypothesis

space. Other early concept learning systems include (Poplestone 1969; Michalski 1973; Buchanan 1974; Vere 1975; Hayes-Roth 1974). A very large number of algorithms have since been developed for concept learning based on symbolic representations. Chapter 10 describes several more recent algorithms for concept learning, including algorithms that learn concepts represented in first-order logic, algorithms that are robust to noisy training data, and algorithms whose performance degrades gracefully if the target concept is not representable in the hypothesis space considered by the learner.

Version spaces and the CANDIDATE-ELIMINATION algorithm were introduced by Mitchell (1977, 1982). The application of this algorithm to inferring rules of mass spectroscopy is described in (Mitchell 1979), and its application to learning search control rules is presented in (Mitchell et al. 1983). Haussler (1988) shows that the size of the general boundary can grow exponentially in the number of training examples, even when the hypothesis space consists of simple conjunctions of features. Smith and Rosenbloom (1990) show a simple change to the representation of the G set that can improve complexity in certain cases, and Hirsh (1992) shows that learning can be polynomial in the number of examples in some cases when the G set is not stored at all. Subramanian and Feigenbaum (1986) discuss a method that can generate efficient queries in certain cases by factoring the version space. One of the greatest practical limitations of the CANDIDATE-ELIMINATION algorithm is that it requires noise-free training data. Mitchell (1979) describes an extension that can handle a bounded, predetermined number of misclassified examples, and Hirsh (1990, 1994) describes an elegant extension for handling bounded noise in real-valued attributes that describe the training examples. Hirsh (1990) describes an INCREMENTAL VERSION SPACE MERGING algorithm that generalizes the CANDIDATE-ELIMINATION algorithm to handle situations in which training information can be different types of constraints represented using version spaces. The information from each constraint is represented by a version space and the constraints are then combined by intersecting the version spaces. Sebag (1994, 1996) presents what she calls a disjunctive version space approach to learning disjunctive concepts from noisy data. A separate version space is learned for each positive training example, then new instances are classified by combining the votes of these different version spaces. She reports experiments in several problem domains demonstrating that her approach is competitive with other widely used induction methods such as decision tree learning and k -NEAREST NEIGHBOR.

EXERCISES

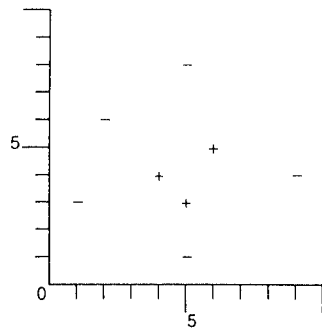
- 2.1. Explain why the size of the hypothesis space in the *EnjoySport* learning task is 973. How would the number of possible instances and possible hypotheses increase with the addition of the attribute *WaterCurrent*, which can take on the values *Light*, *Moderate*, or *Strong*? More generally, how does the number of possible instances and hypotheses grow with the addition of a new attribute A that takes on k possible values?

- 2.2. Give the sequence of S and G boundary sets computed by the CANDIDATE-ELIMINATION algorithm if it is given the sequence of training examples from Table 2.1 in reverse order. Although the final version space will be the same regardless of the sequence of examples (why?), the sets S and G computed at intermediate stages will, of course, depend on this sequence. Can you come up with ideas for ordering the training examples to minimize the sum of the sizes of these intermediate S and G sets for the H used in the *EnjoySport* example?
- 2.3. Consider again the *EnjoySport* learning task and the hypothesis space H described in Section 2.2. Let us define a new hypothesis space H' that consists of all pairwise disjunctions of the hypotheses in H . For example, a typical hypothesis in H' is

$$(\text{?}, \text{Cold}, \text{High}, \text{?}, \text{?}, \text{?}) \vee (\text{Sunny}, \text{?}, \text{High}, \text{?}, \text{?}, \text{Same})$$

Trace the CANDIDATE-ELIMINATION algorithm for the hypothesis space H' given the sequence of training examples from Table 2.1 (i.e., show the sequence of S and G boundary sets.)

- 2.4. Consider the instance space consisting of integer points in the x, y plane and the set of hypotheses H consisting of rectangles. More precisely, hypotheses are of the form $a \leq x \leq b, c \leq y \leq d$, where a, b, c , and d can be any integers.
- (a) Consider the version space with respect to the set of positive (+) and negative (-) training examples shown below. What is the S boundary of the version space in this case? Write out the hypotheses and draw them in on the diagram.



- (b) What is the G boundary of this version space? Write out the hypotheses and draw them in.
- (c) Suppose the learner may now suggest a new x, y instance and ask the trainer for its classification. Suggest a query guaranteed to reduce the size of the version space, regardless of how the trainer classifies it. Suggest one that will not.
- (d) Now assume you are a teacher, attempting to teach a particular target concept (e.g., $3 \leq x \leq 5, 2 \leq y \leq 9$). What is the smallest number of training examples you can provide so that the CANDIDATE-ELIMINATION algorithm will perfectly learn the target concept?
- 2.5. Consider the following sequence of positive and negative training examples describing the concept "pairs of people who live in the same house." Each training example describes an ordered pair of people, with each person described by their sex, hair

color (black, brown, or blonde), height (tall, medium, or short), and nationality (US, French, German, Irish, Indian, Japanese, or Portuguese).

- + $\langle\langle \text{male brown tall US} \rangle\rangle \langle\langle \text{female black short US} \rangle\rangle$
- + $\langle\langle \text{male brown short French} \rangle\rangle \langle\langle \text{female black short US} \rangle\rangle$
- $\langle\langle \text{female brown tall German} \rangle\rangle \langle\langle \text{female black short Indian} \rangle\rangle$
- + $\langle\langle \text{male brown tall Irish} \rangle\rangle \langle\langle \text{female brown short Irish} \rangle\rangle$

Consider a hypothesis space defined over these instances, in which each hypothesis is represented by a pair of 4-tuples, and where each attribute constraint may be a specific value, "?," or " \emptyset ," just as in the *EnjoySport* hypothesis representation. For example, the hypothesis

$$\langle\langle \text{male ? tall ?} \rangle\rangle \langle\langle \text{female ? ? Japanese} \rangle\rangle$$

represents the set of all pairs of people where the first is a tall male (of any nationality and hair color), and the second is a Japanese female (of any hair color and height).

- (a) Provide a hand trace of the CANDIDATE-ELIMINATION algorithm learning from the above training examples and hypothesis language. In particular, show the specific and general boundaries of the version space after it has processed the first training example, then the second training example, etc.
- (b) How many distinct hypotheses from the given hypothesis space are consistent with the following single positive training example?

$$+ \langle\langle \text{male black short Portuguese} \rangle\rangle \langle\langle \text{female blonde tall Indian} \rangle\rangle$$

- (c) Assume the learner has encountered only the positive example from part (b), and that it is now allowed to query the trainer by generating any instance and asking the trainer to classify it. Give a specific sequence of queries that assures the learner will converge to the single correct hypothesis, whatever it may be (assuming that the target concept is describable within the given hypothesis language). Give the shortest sequence of queries you can find. How does the length of this sequence relate to your answer to question (b)?
- (d) Note that this hypothesis language cannot express all concepts that can be defined over the instances (i.e., we can define sets of positive and negative examples for which there is no corresponding describable hypothesis). If we were to enrich the language so that it could express all concepts that can be defined over the instance language, then how would your answer to (c) change?
- 2.6. Complete the proof of the version space representation theorem (Theorem 2.1).
- 2.7. Consider a concept learning problem in which each instance is a real number, and in which each hypothesis is an interval over the reals. More precisely, each hypothesis in the hypothesis space H is of the form $a < x < b$, where a and b are any real constants, and x refers to the instance. For example, the hypothesis $4.5 < x < 6.1$ classifies instances between 4.5 and 6.1 as positive, and others as negative. Explain informally why there cannot be a maximally specific consistent hypothesis for any set of positive training examples. Suggest a slight modification to the hypothesis representation so that there will be.

- 2.8. In this chapter, we commented that given an unbiased hypothesis space (the power set of the instances), the learner would find that each unobserved instance would match exactly half the current members of the version space, regardless of which training examples had been observed. Prove this. In particular, prove that for any instance space X , any set of training examples D , and any instance $x \in X$ not present in D , that if H is the power set of X , then exactly half the hypotheses in $VS_{H,D}$ will classify x as positive and half will classify it as negative.
- 2.9. Consider a learning problem where each instance is described by a conjunction of n boolean attributes $a_1 \dots a_n$. Thus, a typical instance would be

$$(a_1 = T) \wedge (a_2 = F) \wedge \dots \wedge (a_n = T)$$

Now consider a hypothesis space H in which each hypothesis is a *disjunction* of constraints over these attributes. For example, a typical hypothesis would be

$$(a_1 = T) \vee (a_5 = F) \vee (a_7 = T)$$

Propose an algorithm that accepts a sequence of training examples and outputs a consistent hypothesis if one exists. Your algorithm should run in time that is polynomial in n and in the number of training examples.

- 2.10. Implement the FIND-S algorithm. First verify that it successfully produces the trace in Section 2.4 for the *EnjoySport* example. Now use this program to study the number of random training examples required to exactly learn the target concept. Implement a training example generator that generates random instances, then classifies them according to the target concept:

$$\langle \text{Sunny}, \text{Warm}, ?, ?, ?, ? \rangle$$

Consider training your FIND-S program on randomly generated examples and measuring the number of examples required before the program's hypothesis is identical to the target concept. Can you predict the average number of examples required? Run the experiment at least 20 times and report the mean number of examples required. How do you expect this number to vary with the number of "?"s in the target concept? How would it vary with the number of attributes used to describe instances and hypotheses?

REFERENCES

- Bruner, J. S., Goodnow, J. J., & Austin, G. A. (1957). *A study of thinking*. New York: John Wiley & Sons.
- Buchanan, B. G. (1974). Scientific theory formation by computer. In J. C. Simon (Ed.), *Computer Oriented Learning Processes*. Leyden: Noordhoff.
- Gunter, C. A., Ngair, T., Panangaden, P., & Subramanian, D. (1991). The common order-theoretic structure of version spaces and ATMS's. *Proceedings of the National Conference on Artificial Intelligence* (pp. 500-505). Anaheim.
- Hausssler, D. (1988). Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence*, 36, 177-221.
- Hayes-Roth, F. (1974). Schematic classification problems and their solution. *Pattern Recognition*, 6, 105-113.
- Hirsh, H. (1990). Incremental version space merging: A general framework for concept learning. Boston: Kluwer.

- Hirsh, H. (1991). Theoretical underpinnings of version spaces. *Proceedings of the 12th IJCAI* (pp. 665-670). Sydney.
- Hirsh, H. (1994). Generalizing version spaces. *Machine Learning*, 17(1), 5-46.
- Hunt, E. G., & Hovland, D. I. (1963). Programming a model of human concept formation. In E. Feigenbaum & J. Feldman (Eds.), *Computers and thought* (pp. 310-325). New York: McGraw Hill.
- Michalski, R. S. (1973). AQUAL/1: Computer implementation of a variable valued logic system VL1 and examples of its application to pattern recognition. *Proceedings of the 1st International Joint Conference on Pattern Recognition* (pp. 3-17).
- Mitchell, T. M. (1977). Version spaces: A candidate elimination approach to rule learning. *Fifth International Joint Conference on AI* (pp. 305-310). Cambridge, MA: MIT Press.
- Mitchell, T. M. (1979). *Version spaces: An approach to concept learning*, (Ph.D. dissertation). Electrical Engineering Dept., Stanford University, Stanford, CA.
- Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence*, 18(2), 203-226.
- Mitchell, T. M., Utgoff, P. E., & Banerji, R. (1983). Learning by experimentation: Acquiring and modifying problem-solving heuristics. In Michalski, Carbonell, & Mitchell (Eds.), *Machine Learning* (Vol. 1, pp. 163-190). Tioga Press.
- Plotkin, G. D. (1970). A note on inductive generalization. In Meltzer & Michie (Eds.), *Machine Intelligence 5* (pp. 153-163). Edinburgh University Press.
- Plotkin, G. D. (1971). A further note on inductive generalization. In Meltzer & Michie (Eds.), *Machine Intelligence 6* (pp. 104-124). Edinburgh University Press.
- Popplestone, R. J. (1969). An experiment in automatic induction. In Meltzer & Michie (Eds.), *Machine Intelligence 5* (pp. 204-215). Edinburgh University Press.
- Sebag, M. (1994). Using constraints to build version spaces. *Proceedings of the 1994 European Conference on Machine Learning*. Springer-Verlag.
- Sebag, M. (1996). Delaying the choice of bias: A disjunctive version space approach. *Proceedings of the 13th International Conference on Machine Learning* (pp. 444-452). San Francisco: Morgan Kaufmann.
- Simon, H. A., & Lea, G. (1973). Problem solving and rule induction: A unified view. In Gregg (Ed.), *Knowledge and Cognition* (pp. 105-127). New Jersey: Lawrence Erlbaum Associates.
- Smith, B. D., & Rosenbloom, P. (1990). Incremental non-backtracking focusing: A polynomially bounded generalization algorithm for version spaces. *Proceedings of the 1990 National Conference on Artificial Intelligence* (pp. 848-853). Boston.
- Subramanian, D., & Feigenbaum, J. (1986). Factorization in experiment generation. *Proceedings of the 1986 National Conference on Artificial Intelligence* (pp. 518-522). Morgan Kaufmann.
- Vere, S. A. (1975). Induction of concepts in the predicate calculus. *Fourth International Joint Conference on AI* (pp. 281-287). Tbilisi, USSR.
- Winston, P. H. (1970). *Learning structural descriptions from examples*, (Ph.D. dissertation). [MIT Technical Report AI-TR-231].