Chapter 3

Using Version Spaces for Learning

3.1 Version Spaces and Mistake Bounds

The first learning methods we present are based on the concepts of *version spaces* and *version graphs*. These ideas are most clearly explained for the case of Boolean function learning. Given an initial hypothesis set $\mathcal{H}$ (a subset of all Boolean functions) and the values of $f(\mathbf{X})$ for each $\mathbf{X}$ in a training set, Ξ, the version space is that subset of hypotheses, $\mathcal{H}_v$, that is consistent with these values. A hypothesis, h, is *consistent* with the values of $\mathbf{X}$ in Ξ if and only if $h(\mathbf{X}) = f(\mathbf{X})$ for all $\mathbf{X}$ in Ξ. We say that the hypotheses in $\mathcal{H}$ that are not consistent with the values in the training set are *ruled out* by the training set.

We could imagine (conceptually only!) that we have devices for implementing every function in $\mathcal{H}$. An incremental training procedure could then be defined which presented each pattern in Ξ to each of these functions and then eliminated those functions whose values for that pattern did not agree with its given value. At any stage of the process we would then have left some subset of functions that are consistent with the patterns presented so far; this subset is the version space for the patterns already presented. This idea is illustrated in Fig. 3.1.

Consider the following procedure for classifying an arbitrary input pattern, $\mathbf{X}$: the pattern is put in the same class (0 or 1) as are the majority of the outputs of the functions in the version space. During the learning procedure, if this majority is not equal to the value of the pattern presented,

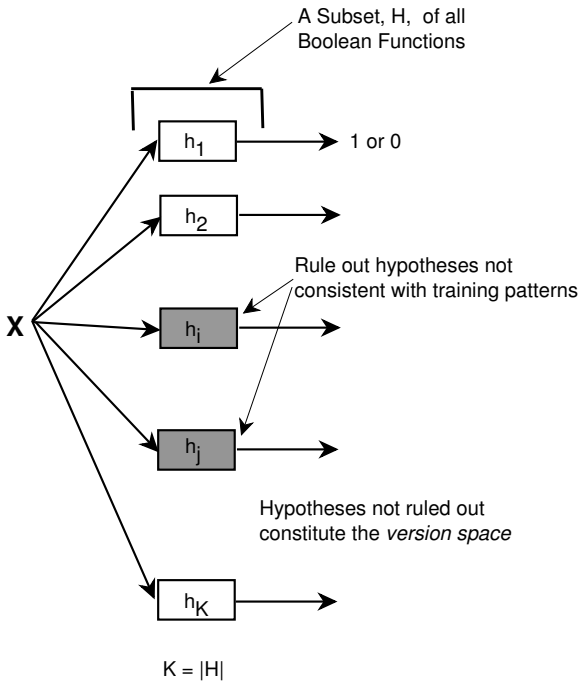

Figure 3.1: Implementing the Version Space

we say a *mistake* is made, and we revise the version space accordingly—eliminating all those (majority of the) functions voting incorrectly. Thus, whenever a mistake is made, we rule out at least half of the functions remaining in the version space.

How many mistakes can such a procedure make? Obviously, we can make no more than $\log_2(|\mathcal{H}|)$ mistakes, where $|\mathcal{H}|$ is the number of hypotheses in the original hypothesis set, $\mathcal{H}$. (Note, though, that the number of training patterns seen before this maximum number of mistakes is made might be much greater.) This theoretical (and very impractical!) result (due to [Littlestone, 1988]) is an example of a *mistake bound*—an important concept in machine learning theory. It shows that there must exist a learning procedure that makes no more mistakes than this upper bound. Later, we'll derive other mistake bounds.

As a special case, if our bias was to limit $\mathcal{H}$ to terms, we would make no more than $\log_2(3^n) = n\log_2(3) = 1.585n$ mistakes before *exhausting* the

version space. This result means that if f were a term, we would make no more than $1.585n$ mistakes before learning f, and otherwise we would make no more than that number of mistakes before being able to decide that f is not a term.

Even if we do not have sufficient training patterns to reduce the version space to a single function, it may be that there are enough training patterns to reduce the version space to a set of functions such that most of them assign the same values to most of the patterns we will see henceforth. We could select one of the remaining functions at random and be reasonably assured that it will generalize satisfactorily. We next discuss a computationally more feasible method for representing the version space.

3.2 Version Graphs

Boolean functions can be ordered by *generality*. A Boolean function, f_1, is *more general* than a function, f_2, (and f_2 is *more specific* than f_1), if f_1 has value 1 for all of the arguments for which f_2 has value 1, and $f_1 \neq f_2$. For example, x_3 is more general than x_2x_3 but is not more general than $x_3 + x_2$.

We can form a graph with the hypotheses, $\{h_i\}$, in the version space as nodes. A node in the graph, h_i, has an arc directed to node, h_j, if and only if h_j is more general than h_i. We call such a graph a *version graph*. In Fig. 3.2, we show an example of a version graph over a 3-dimensional input space for hypotheses restricted to terms (with none of them yet ruled out).

That function, denoted here by "1," which has value 1 for all inputs, corresponds to the node at the top of the graph. (It is more general than any other term.) Similarly, the function "0," is at the bottom of the graph. Just below "1," is a row of nodes corresponding to all terms having just one literal, and just below them is a row of nodes corresponding to terms having two literals, and so on. There are $3^3 = 27$ functions altogether (the function "0," included in the graph, is technically not a term). To make our portrayal of the graph less cluttered only some of the arcs are shown; each node in the actual graph has an arc directed to all of the nodes above it that are more general.

We use this same example to show how the version graph changes as we consider a set of labeled samples in a training set, Ξ. Suppose we first consider the training pattern (1, 0, 1) with value 0. Some of the functions in the version graph of Fig. 3.2 are inconsistent with this training pattern. These ruled out nodes are no longer in the version graph and are

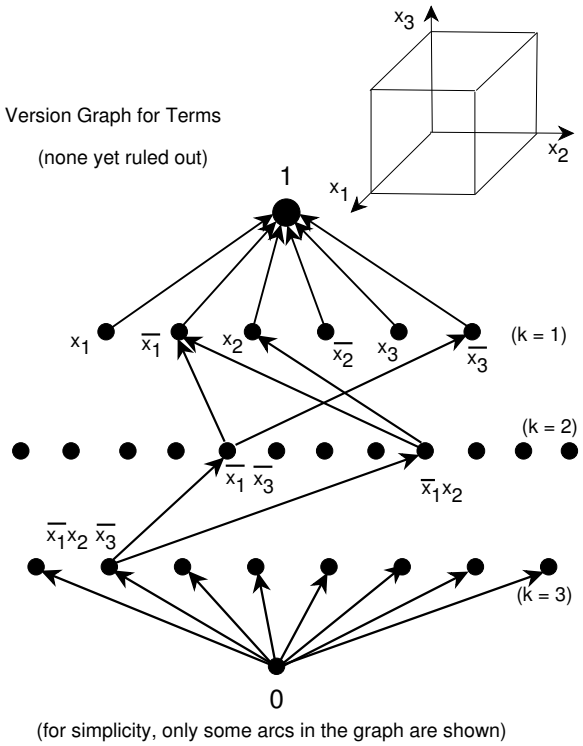

Figure 3.2: A Version Graph for Terms

shown shaded in Fig. 3.3. We also show there the three-dimensional cube representation in which the vertex (1, 0, 1) has value 0.

In a version graph, there are always a set of hypotheses that are maximally general and a set of hypotheses that are maximally specific. These are called the *general boundary set (gbs)* and the *specific boundary set (sbs)*, respectively. In Fig. 3.4, we have the version graph as it exists after learning that (1,0,1) has value 0 and (1, 0, 0) has value 1. The gbs and sbs are shown.

Boundary sets are important because they provide an alternative to representing the entire version space explicitly, which would be impractical. Given only the boundary sets, it is possible to determine whether or not any hypothesis (in the prescribed class of Boolean functions we are using)

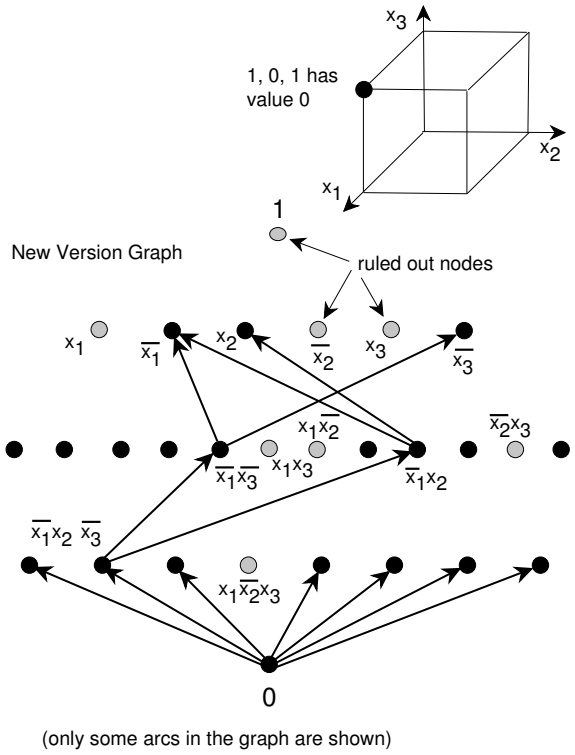

Figure 3.3: The Version Graph Upon Seeing (1, 0, 1)

is a member or not of the version space. This determination is possible because of the fact that any member of the version space (that is not a member of one of the boundary sets) is more specific than some member of the general boundary set and is more general than some member of the specific boundary set.

If we limit our Boolean functions that can be in the version space to terms, it is a simple matter to determine maximally general and maximally specific functions (assuming that there is some term that is in the version space). A maximally specific one corresponds to a subface of *minimal dimension* that contains all the members of the training set labelled by a 1 and no members labelled by a 0. A maximally general one corresponds to a subface of *maximal dimension* that contains all the members of the training

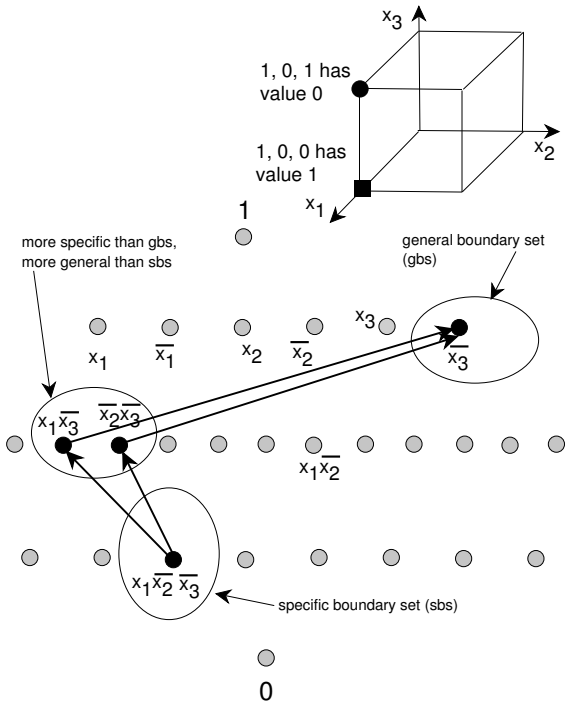

Figure 3.4: The Version Graph Upon Seeing (1, 0, 1) and (1, 0, 0)

set labelled by a 1 and no members labelled by a 0. Looking at Fig. 3.4, we see that the subface of minimal dimension that contains (1, 0, 0) but does not contain (1, 0, 1) is just the vertex (1, 0, 0) itself—corresponding to the function $x_1\overline{x_2}\ \overline{x_3}$. The subface of maximal dimension that contains (1, 0, 0) but does not contain (1, 0, 1) is the bottom face of the cube—corresponding to the function $\overline{x_3}$. In Figs. 3.2 through 3.4 the sbs is always singular. Version spaces for terms always have singular specific boundary sets. As seen in Fig. 3.3, however, the gbs of a version space for terms need not be singular.

3.3 Learning as Search of a Version Space

[To be written. Relate to term learning algorithm presented in Chapter Two. Also discuss best-first search methods. See Pat Langley's example

using "pseudo-cells" of how to generate and eliminate hypotheses.]

Selecting a hypothesis from the version space can be thought of as a search problem. One can start with a very general function and specialize it through various specialization operators until one finds a function that is consistent (or adequately so) with a set of training patterns. Such procedures are usually called *top-down* methods. Or, one can start with a very special function and generalize it—resulting in *bottom-up* methods. We shall see instances of both styles of learning in this book.

Compare this view of top-down versus bottom-up with the *divide-and-conquer* and the *covering* (or AQ) methods of decision-tree induction.

3.4 The Candidate Elimination Method

The *candidate elimination method*, is an incremental method for computing the boundary sets. Quoting from [Hirsh, 1994, page 6]:

> "The *candidate-elimination algorithm* manipulates the boundary-set representation of a version space to create boundary sets that represent a new version space consistent with all the previous instances plus the new one. For a positive exmple the algorithm generalizes the elements of the [sbs] as little as possible so that they cover the new instance yet remain consistent with past data, and removes those elements of the [gbs] that do not cover the new instance. For a negative instance the algorithm specializes elements of the [gbs] so that they no longer cover the new instance yet remain consistent with past data, and removes from the [sbs] those elements that mistakenly cover the new, negative instance."

The method uses the following definitions (adapted from [Genesereth & Nilsson, 1987]):

- a hypothesis is called *sufficient* if and only if it has value 1 for all training samples labeled by a 1,
- a hypothesis is called *necessary* if and only if it has value 0 for all training samples labeled by a 0.

Here is how to think about these definitions: A hypothesis implements a *sufficient* condition that a training sample has value 1 if the hypothesis has value 1 for all of the positive instances; a hypothesis implements a *necessary* condition that a training sample has value 1 if the hypothesis has value 0 for all of the negative instances. A hypothesis is consistent with the training

set (and thus is in the version space) if and only if it is both sufficient and necessary.

We start (before receiving any members of the training set) with the function "0" as the singleton element of the specific boundary set and with the function "1" as the singleton element of the general boundary set. Upon receiving a new labeled input vector, the boundary sets are changed as follows:

1. If the new vector is labelled with a 1:

   The new general boundary set is obtained from the previous one by excluding any elements in it that are not sufficient. (That is, we exclude any elements that have value 0 for the new vector.)

   The new specific boundary set is obtained from the previous one by replacing each element, h_i, in it by all of its *least generalizations*.

   The hypothesis h_g is a *least generalization* of h if and only if: a) h is more specific than h_g, b) h_g is sufficient, c) no function (including h) that is more specific than h_g is sufficient, and d) h_g is more specific than some member of the new general boundary set. It might be that $h_g = h$. Also, least generalizations of two different functions in the specific boundary set may be identical.

2. If the new vector is labelled with a 0:

   The new specific boundary set is obtained from the previous one by excluding any elements in it that are not necessary. (That is, we exclude any elements that have value 1 for the new vector.)

   The new general boundary set is obtained from the previous one by replacing each element, h_i, in it by all of its *least specializations*.

   The hypothesis h_s is a *least specialization* of h if and only if: a) h is more general than h_s, b) h_s is necessary, c) no function (including h) that is more general than h_s is necessary, and d) h_s is more general than some member of the new specific boundary set. Again, it might be that $h_s = h$, and least specializations of two different functions in the general boundary set may be identical.

As an example, suppose we present the vectors in the following order:

vector	label
(1, 0, 1)	0
(1, 0, 0)	1
(1, 1, 1)	0
(0, 0, 1)	0

We start with general boundary set, "1", and specific boundary set, "0." After seeing the first sample, (1, 0, 1), labeled with a 0, the specific boundary set stays at "0" (it is necessary), and we change the general boundary set to $\{\overline{x_1}, x_2, \overline{x_3}\}$. Each of the functions, $\overline{x_1}$, x_2, and $\overline{x_3}$, are least specializations of "1" (they are necessary, "1" is not, they are more general than "0", and there are no functions that are more general than they and also necessary).

Then, after seeing (1, 0, 0), labeled with a 1, the general boundary set changes to $\{\overline{x_3}\}$ (because $\overline{x_1}$ and x_2 are not sufficient), and the specific boundary set is changed to $\{x_1\overline{x_2}\ \overline{x_3}\}$. This single function is a least generalization of "0" (it is sufficient, "0" is more specific than it, no function (including "0") that is more specific than it is sufficient, and it is more specific than some member of the general boundary set.

When we see (1, 1, 1), labeled with a 0, we do not change the specific boundary set because its function is still necessary. We do not change the general boundary set either because $\overline{x_3}$ is still necessary.

Finally, when we see (0, 0, 1), labeled with a 0, we do not change the specific boundary set because its function is still necessary. We do not change the general boundary set either because $\overline{x_3}$ is still necessary.

Maybe I'll put in an example of a version graph for non-Boolean functions.

3.5 Bibliographical and Historical Remarks

The concept of version spaces and their role in learning was first investigated by Tom Mitchell [Mitchell, 1982]. Although these ideas are not used in practical machine learning procedures, they do provide insight into the nature of hypothesis selection. In order to accomodate noisy data, version spaces have been generalized by [Hirsh, 1994] to allow hypotheses that are not necessarily consistent with the training set.

More to be added.