VERSION SPACES: A CANDIDATE ELIMINATION APPROACH TO RULE LEARNING

Tom M. Mitchell

Heuristic Programming Project
Department of Computer Science
Stanford University
Stanford, California, 94305.

ABSTRACT*

An important research problem in artificial intelligence is the study of methods for learning general concepts or rules from a set of training instances. An approach to this problem is presented which is guaranteed to find, without backtracking, all rule versions consistent with a set of positive and negative training instances. The algorithm put forth uses a representation of the space of those rules consistent with the observed training data. This "rule version space" is modified in response to new training instances by eliminating candidate rule versions found to conflict with each new instance. The use of version spaces is discussed in the context of Meta-DENDRAL, a program which learns rules in the domain of chemical spectroscopy.

Descriptive terms: Version space, rule learning, candidate elimination, concept formation, concept learning, machine learning, Meta-DENDRAL.

1 INTRODUCTION

Rule learning and concept learning have become increasingly important goals for artificial intelligence (AI) researchers with the recent emphasis within the AI community on constructing knowledge based systems [2], [6]. A program capable of extracting general rules from training instances would be a valuable tool for the difficult task of constructing knowledge bases for use in such systems. Although there has been some success in this area [1], [9] we are still far from an understanding of efficient, reliable methods for controlling the combinatorics inherent to the task of learning.

The rule learning or concept learning problem addressed in this paper is the following. It is given that some fixed action, A, is advisable in some class of (positive) training instances, I+, but is inadvisable in some disjoint class of (negative) training instances, I-**. The task is to determine a rule of the form P ➡ A, where P is a set of conditions or constraints from some predefined language. These conditions must be satisfied for action A to be invoked. The learned rule must apply to all instances from I+, but to no instances from I-.

One popular paradigm for this rule learning task is the search paradigm. In this approach, a rule which Is consistent with the first training instance is determined, then additional training instances are considered one at a time. At each step, P is revised as needed according to a well defined set of legal alterations, so that the resulting rule version applies to exactly the correct set of instances. This search through the space of allowed patterns for the correct statement of P *** may be viewed as a concept formation problem in which the concept being learned is "the class of instances in which action A is recommended".

Two difficulties arise in the search paradigm. First, in determining the set of possible alterations to the rule in response to a given training instance (i.e. in determining the set of legal branches at a given point in the search tree), care must be taken to assure that each is consistent with correct rule performance on past training instances. Second, backtracking is sometimes required to try a different branch of the search tree when subsequent training instances reveal that an inappropriate decision has been made.

This paper proposes an approach to learning rules which falls outside the search paradigm outlined above. This candidate elimination, algorithm relies upon a method for representing and updating the space of all rule versions

* This work was supported by the Advanced Research Projects Agency under contract DAHC 15-73-C-0435 and by the National Institutes of Health under grant RR 00612-07. Computer resources were provided by the Sumex-AIM computer facility at Stanford University under NIH grant RR-00785. Bruce G. Buchanan has provided many useful suggestions throughout the course of this work. Lew G. Creary assisted in the clarification of several issues discussed in this paper.

•* In many rule induction tasks obtaining this assignment of training instances to 1+ and I- for a given action is a difficult problem in itself. See Minsky [31 for a discussion of the credit assignment problem.

*** For a general discussion of rule induction as heuristic search see Simon and Lea [7], and Buchanan [1].

consistent with the observed data. Rules are eliminated from this rule version space as they are found to conflict with observed training instances. The algorithm is guaranteed to find, without backtracking or reexamining past training instances, the set of all rules consistent with the observed data.

2 VERSION SPACES

This section proposes a candidate elimination approach to rule learning which maintains and modifies a representation of the space of all plausible rule versions (contrast this with the search paradigm discussed above which maintains and modifies a single current best hypothesis of the correct rule version). Methods for representing this rule version space and for modifying it in response to new training data are presented in this section. The candidate elimination algorithm is guaranteed to find all rule versions consistent with the observed training data. This is accomplished without backtracking and independent of the order of presentation of the training instances.

2.1 Definition and Representation

The term version space is used in this paper to refer to the set of current hypotheses of the correct statement of a rule which predicts some fixed action, A. In other words, it. is the set of those statements of the rule which cannot be ruled out on the basis of training instances observed thus far. It is easy to see that this version space contains the set of all plausible rule revisions which may be made by a search algorithm in response to some new training instance.

More exactly, assume that there is some rule R which correctly predicts action A for the class of training instances I+, but not for instances in the class I-. The rule version space of 1+ and I- is then defined as the set of all such rules. The rule version space associated with action A and the instances 1+ and I- is an equivalence class of rules with respect to their predictions on 1+ and I-. The elements of the version space are rules which predict the same action, but which differ in the patterns stated in their left hand sides.

Before writing programs which reason in terms of version spaces, we must have a compact-data representation for them. In general, the number of plausible versions can be very large (possibly infinite) when the language of patterns for rules is complex. The key to an efficient representation of version spaces lies in observing that a general-to-specific ordering is defined on the rule pattern space by the pattern matching procedure used for applying rules. The version space may be represented in terms of its maximal and minimal elements according to this ordering.

To see exactly how the general-to-specific ordering comes about, consider an example. Suppose that R1 and R2 are two rules which predict the same action. Then R1 is said to be more specific (or, equivalently, less general) than R2 if and only if it will apply to a proper subset of the instances in which R2 will apply. This definition, which relies upon the pattern matching mechanism, is simply a formalization of the intuitive ideas of "more specific" and "less general". For the remainder of this paper, the terms general and specific will be understood to have these well defined meanings.

The general-to-specific ordering will in general be a partial ordering; that is, given any two rules we cannot, always say that one is more general than the other. For instance, two rules can easily apply to some of the same instances without the requirement that one of them apply everywhere that the other applies. Therefore, when all elements of the version space are ordered according to generality, there may be several maximally general and maximally specific versions.

Version spaces can be represented by these sets of maximally general versions. MGV, and maximally specific versions. MSV. Given such a representation it is quite easy to determine whether a given rule belongs to a given version space. A rule statement belongs to the version space of 1+ and I- if and only if it is (1) less general than or equal to one of the maximally general versions, and (2) less specific than or equal to one of the maximally specific versions. Condition (1) assures that the rule cannot match any training instance in I-, while condition (2) assures that it will match every training instance in I+. Since the sets MGV and MSV are by definition complete, (1) and (?) will be necessary as well as sufficient conditions for membership of a rule statement in the version space.

2.1.1 An Example: Meta-DENDRAL

An algoriThm for representing version spaces as described above has been implemented in the Meta-DENDRAL program. Meta-DENDRAL [1] is a program which learns production rules to describe the behavior of classes of i olecules in two areas of chemical spectroscopy. The version of the program which determines rules associating substructures of molecules with data peaks in a carbon-13 nuclear magnetic resonance spectrum shall be considered here [14].

Figure 1 shows a version space represented by the program in terms of the sets of maximally specific rule versions (rule MSV1) and maximally general rule versions (rules MGV1 and MGV2). The rule pattern which expresses the conditions for application of each rule is stated in a fairly complex network language of chemical subgraphs. Each node in the subgraph represents an atom in a molecular structure. Each subgraph node has the four attributes shown, with values constrained as shown in Figure 1. Arcs between nodes in the rule

Rule Subgraph			Constraints on Subgraph Node Attributes			
Rule	subgraph	node name	atom type	number of non-hydrogen neighbors	number of hydrogen neighbors	number of unsaturated electrons
MSV1	v-w-x-y-z	v	carbon	1	3	0
		w	carbon	2	2	0
		x	carbon	2	2	0
		y	carbon	2	2	0
		z	carbon	>=1	any	0
MGV1	v-w-x	v	carbon	1	any	any
		w	any	2	any	any
		x	any	>=1	2	any
MGV2	v-w-x	v	carbon	1	any	any
		w	any	2	any	any
		x	any	2	any	any

Figure 1. A Version Space Represented by it's Extremal Sets

MSV1 is the maximally specific rule version. MGV1 and MGV2 are maximally general rule versions. Only the rule patterns (left hand sides) are shown above. All rules shown predict the same action: the appearance of a peak associated with atom "v" in a given range of the spectrum.

subgraph (shown schematically as lines between the node letters) represent chemical bonds between atoms. The definition of a match of the rule pattern (subgraph) to a training instance (molecule graph) requires that the subgraph "fit into" the molecule graph, and that the subgraph node attribute constraints be consistent with the attribute values of the corresponding node in the molecule graph. If a molecule graph contains a set of nodes which fit the pattern, then the corresponding action is predicted by the rule. In this program the classes 1+ and I- are sets of molecules for which the indicated spectral peak does and does not appear.

The version space represented in Figure 1 contains several hundred rule versions: the three versions shown plus all versions between these in the general-to-specific ordering. However, it can be represented simply by the two maximally general versions, MGV1 and MGV2, and the single maximally specific version, MSV1. The single most specific version contains every node and node attribute constraint consistent with all training instances in I+. Thus, any more specific version cannot match every element of I+. Two general versions are required in this example since neither is "above" the other in the general-to-specific partial ordering. Any rule more general than either MGV1 or MGV2 will match some element of I-. Furthermore, any rule which is between these general and specific boundaries of the version space will match all current instances in 1+ (by virtue of being more general than MSV1), and will match no current instances from I- (by virtue of being more specific than MGV1 or MGV2). Notice that the constraints stated by MSV1 are more strict than those stated by MGV1 and MGV2, and that MSV1 contains a superset of the nodes contained in the maximally general versions. This is consistent with the general-to-specific ordering discussed above, in that MSV1 will apply to a proper subset of those instances to which MGV1 and MGV2 apply.

2.2 Version Spaces and Rule Learning

Recall the rule learning task discussed earlier. A program is given examples of two classes of training instances, 1+ and I-. The program must determine some rule which will produce a given action, A, for each training instance in I+, but for no instance in I-.

The candidate elimination algorithm presented here operates on the version space of all plausible rules at each step, beginning with the space of all rule versions consistent with the first positive training instance, and modifying the version space to eliminate candidate versions

found to conflict with subsequent training instances.

The chief difference between the candidate elimination approach and the search approach discussed above is that search techniques select and modify a current best hypothesis of the form of the rule. Rather than select a single best rule version, the candidate elimination algorithm represents the space of all plausible rule versions, eliminating from consideration only those versions found to conflict with observed training instances. Thus, the candidate elimination approach separates the deductive step of determining which rule versions are plausible, from the inductive step of selecting a current-best-hypothesis. At any step, the same heuristics used by search methods to infer the current best hypothesis may be applied to infer the best-element contained in the version space. However, by refraining from committing itself to this inductive step, the candidate elimination algorithm completely avoids the need to backtrack to undo past decisions or reexamine old training instances. At the same time the algorithm is assured of finding all correct versions of the rule after all training data has been presented. These are the strongest two advantages of the candidate elimination approach to learning.

2.2.1 Meta-DENDRAL Example Revisited

The candidate elimination algorithm using version spaces has been implemented as part of the meta-DENDRAL program. Recall from the earlier example that in this program the training instance classes 1+ and l- are molecule graphs for which some action (i.e. the appearance of a peak in some region of their spectra) should or should not be predicted. The first part of Meta-DENDRAL determines several different predicted actions associated with sets of positive and negative training instances* Rule version spaces for each distinct predicted action are then generated from the training instances associated with the action. Subsequent data may be analyzed to modify the version space in a manner guaranteed to be consistent with the original data.

The candidate elimination algorithm operates on the maximally general and maximally specific sets representing the version space. The set of maximally general rule versions (MGV) is initialized to a single pattern consisting of the most general statement in the language of rule patterns (a single atom graph with no constrained node attributes). The set of maximally specific versions (MSV) is initialized to a rule which contains as its pattern the first instance in l+. The initial version space represented by these extremal sets therefore contains all rules in the language which match the first training instance.

The training instances are then considered one at a time. Each training instance is used to eliminate from the version space those rule versions which conflict with that instance. This is always accomplished by shifting the maximally specific and maximally general boundaries of the version space toward each other as shown in figure *2.*

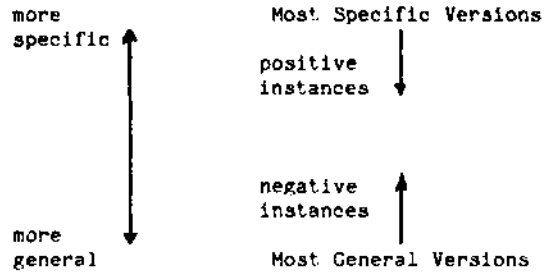

Figure 2

Effect of Positive and Negative
Training Instances on Version Space Boundaries

Positive training instances force elements of MSV to become more general, whereas negative training instances force elements of MGV to become more specific. The maximally specific set can, of course, never be replaced by a more specific set (nor the maximally general set by a more general one) since by definition, any version outside the current version space boundaries is inconsistent with previous training data. The action taken by the candidate elimination algorithm in updating the extremal sets is given below.

If the new training instance belongs to l-, then each element of MGV which matches the instance must be replaced by a set of minimally more specific versions which do not match the instance. These new versions are obtained by adding constraints taken from elements in MSV in order to ensure that they remain more general than some MSV, and thus remain consistent with previous 1+ instances. Furthermore, each element of MSV which matches the negative training instance must be eliminated from the set (since it is already maximally specific, it cannot be replaced by a more specific version).

If the new training instance belongs instead to l*, then any elements from MSV which do not match the new positive training instance are replaced by a set of minimally more general elements which do match the instance. In order to ensure that these more general versions do not match past training instances from l-, any which are not more specific than at least one element of MGV are eliminated. Elements from MGV which do not match the positive instance are eliminated.

After processing each training instance, the new maximally general and maximally specific sets

(such as those shown in Figure 1) will bound the space of fill rules consistent with the observed data. Notice that by modifying the version space in the above way, all rules (and only those rules) which conflict with the new training instance are eliminated from consideration.

3 REASONING WITH VERSION SPACRS

An explicit representation for the space of plausible rule versions appears to have uses in addition to that described above. This section discusses some limitations on the general applicability of version spaces, as well as some promising additional uses.

3.1 Applicability and Limitations

The version space approach to rule learning described above is limited in certain respects, The algorithm is based upon the assumption that the assignment of training instances to 1+ and I- is consistent (i.e. the supplied classification of training instances can be generated by at least one rule in the rule space). In some domains this may be a valid assumption, but in certain "noisy" domains the process of classifying instances may be unreliable or the training instances themselves may be inconsistent. If the set of training instances is not consistent, the algorithm will eliminate all rule versions from consideration, and backtracking will be required. This is, however, no worse than is expected from other non-statistical algorithms, all of which also require backtracking in this case. One positive point is that the collapse of the version space to the null space provides an immediate indication that something has gone wrong. Specifically, it indicates that there is no rule within the supplied rule language which can discriminate between 1+ and I- (this can occur either for noisy data, or in the case where the rule language is not sufficiently complex to represent the given dichotomy of 1+ and I-).

One method for making the candidate elimination algorithm more accommodating of noisy data might be to eliminate only candidate versions which conflict with some fixed number of training instances greater than one* The price paid in exchange for this extension would be a decrease in the rate at which the version space boundaries converge toward each other.

A second limitation on the general applicability of version spaces may lie in the nature of the partial ordering of rule versions. For simple languages of rule patterns the sizes of the maximally general and maximally specific sets of versions will be small. It appears in Meta-DENDRAL that the size of these sets may be manageable for simple molecules in spite of the complex language of rule patterns* However, it is possible that for some domains the size of these extremal sets may become quite large. In Meta-DENDRAL, information about the interdependences of the node properties is used to eliminate syntactically distinct rule statements which are semantically equivalent. Thus, the size of the extremal sets is not adversely affected by redundancy in the rule language. A second possible method for limiting the size of the maximally general and maximally specific version sets is the introduction of domain-specific constraints on allowed elements of the version space. It is common in AI programs to use task domain knowledge to constrain combinatorially explosive problems. It may be possible to incorporate such constraints in the routines which manipulate version spaces in order to dismiss a priori unlikely rule statements present in the rule language.

It appears that the practical applicability of the version space approach to other domains is limited only by the above two constraints: the requirement of reliable training data, and the danger of overly large maximally general and maximally specific sets. The only assumption critical to this approach is that a partial ordering exist over the space of rule patterns. This assumption is always satisfied since the ordering is defined in any domain by the pattern matcher employed. In general it seems that the version space approach will be most efficient in domains in which the the partial ordering over the pattern space is not extremely "branchy", but where each branch may be quite deep. This is due to the fact that only the most and least specific plausible version (if any exist) along each branch need be saved. Thus, the efficiency of the version space algorithm (and the size of the extremal sets) is unaffected by the depth of each branch, but is adversely affected by the number of branches. In contrast, the efficiency of search procedures and their need for backtracking appear to be adversely affected by both the number of branches in the partial ordering and the depth of the branches.

3.2 Other Uses for Version Spaces

Version spaces provide an explicit representation of the range of plausible rules. With this explicit representation, the program acquires the ability to reason more abstractly about its actions. The program is aware of more than the current best hypothesis - it has available the entire range of plausible choices. This view of version spaces suggests their use for tasks other than the particular rule learning task described above. Two such tasks will be suggested in this section.

3.2.1 Selecting New Training Instances

The importance of careful selection of training instances for efficient and reliable learning has been stressed by several writers [8], [10], yet few learning programs take an

active role in determining their own training instances. One notable exception is an induction program written by Popplestone [51 which itself generates training instances whose (user supplied) classification resolves among competing hypotheses. The version space representation appears well suited for allowing the program to generate its own set of critical training instances.

Since version spaces represent the range of rule versions which cannot be resolved by the current training data, they also summarize the range of unencountered training instances that will be useful in selecting among competing rule versions. By constructing a training instance which matches some, but not all, of the maximally general versions, the program may be able to determine which of several potentially important attributes should be specified in a rule. On the other hand, by constructing training instances which match a given most general version, but not its most specific counterpart, the program may determine how specific the constraint on a given attribute must be. Note (as pointed out by one of the referees of this paper) that the generation of training instances might also provide a useful tool for limiting the size of the maximally specific and maximally general sets, in that examples designed to discriminate among competing extremal versions could be generated.

3.2.2 Merging Separately Obtained Results

The construction of reliable knowledge bases for use in knowledge based programs will almost certainly require learning from a large variety and number of training instances. Merging rules learned from different data sets may therefore become a desirable capability (consider breaking a large training data set into several smaller sets to be analysed in parallel).

Version spaces allow a convenient, consistent method for merging sets of rules generated from distinct training data sets* The intersection of the version spaces of two rules formed from two sets of training data yields the version space of all rules consistent with the union of the data sets. The result obtained by intersecting version spaces derived from different data sets is therefore the same as would be obtained by running the candidate elimination algorithm once on the union of the training data.

U SUMMARY

The representation of version spaces in terms of their sets of maximally general and maximally specific versions appears well suited for learning rules from sequentially presented training instances, A candidate elimination algorithm has been shown which will find all rule versions consistent with all training instances. Backtracking is not required for noise-free training instances, and the final result is independent of the order of presentation of instances.

Version spaces provide at once a compact summary of past training instances and a representation of all plausible rule versions. Because they provide an explicit representation for the space of plausible rules, version spaces allow a program to represent "how much it doesn't know" about the correct form of the rule. This suggests the utility of the version space approach to problems such as intelligent selection of training instances and merging sets of independently generated rules.

References

1. B. G. Buchanan and T. M. Mitchell, "Model-Directed Learning of Production Rules", Proceedings of the Workshop on Pattern-Directed Inference Systems, Honolulu, Hawaii, May 1977,

2. E. A. Feigenbaum, B. G. Buchanan, and J. Lederberg, "On Generality and Problem Solving: A Case Study Using the DENDRAL Program", in Machine Intelligence j5, B. Meltzer and D. Michie, eds., American Elsevier, N. Y., 1971, pp. 165-190.

3. M. Minsky, "Steps Toward Artificial Intelligence", in Computers and Thought. E. A. Feigenbaum and J. Feldman, eds., McGraw-Hill, N. Y. , 1963, pp. 406-450.

U. T. M. Mitchell and G* M. Schwenzer, "A Computer Program for Automated Empirical 13C-NMR Rule Formation[1'], Heuristic Programming Project Working Paper HPP-77-^, Stanford University, February, 1977.

5. R. J. Popplestone, "An Experiment in Automatic Induction", in Machine Intelligence £, B. Meltzer and D. Michie, eds., Edinburgh University Press, 1969.

6. E. Shortliffe, MYCIN: Computer-Based Medical Consultations. American Elsevier, N* Y», 1976.

7. H. A. Simon and G. Lea, "Problem Solving and Rule Induction: A Unified View", in L. W. Gregg, ed., Knowledge and Cognition. Lawrence Erlbaum Associates, Potomac, Maryland, 197**, pp. 105-127.

R. G, Smith, T. M. Mitchell, R. A, Chestek, and B. G. Buchanan, "A Model for Learning Systems", Proceedings of the 5th IJCAI, Cambridge, MA, August 1977.

9. D. A. Waterman, "Adaptive Production Systems", Complex Information Processing Working Paper 285, Dept. of Psychology, CMU, December, 197*4,

10. P. H. Winston, "Learning Structural Descriptions from Examples", Ph. D. thesis, MIT AI-TR-231, September 1970,