The Computational Complexity of the Candidate-Elimination Algorithm

Haym Hirsh
Computer Science Department
Hill Center for the Mathematical Science
Busch Campus
Rutgers University
New Brunswick, NJ 08903
hirsh@cs.rutgers.edu

Abstract

Mitchell's original work on version spaces (Mitchell, 1982) presented an analysis of the computational complexity of version spaces. However, this analysis proved somewhat coarse, as it was parameterized by s and g, the maximum sizes that the S and G sets reach during learning. As has been pointed out by Haussler (1988), g can be exponential in the number of examples processed. This paper presents a more fine-grained analysis of the computational complexity of version spaces, demonstrates its equivalence to Mitchell's analysis, and instantiates it for two commonly used conjunctive concept description languages.

1 Introduction

The problem of inductive concept learning—to form general rules from data—has been well-studied in machine learning and artificial intelligence. The problem can be stated as follows:

Given:

- *Training Data*: Positive and negative examples of a concept to be identified.
- *Concept Description Language*: A language in which the final concept definition must be expressed.

Determine:

- The desired unknown concept.

Mitchell (1978; 1982) proposed an approach to this problem that maintains all elements of the concept description language that could be the desired unknown concept, namely, those that correctly classify the given training data (i.e., that are consistent with the data); this set is known as a *version space*. However, this set can be efficiently represented by only maintaining the minimal and maximal elements of the set, known as the *boundary sets* of the version space. As further data are obtained, the set of possibilities is refined until ideally only one consistent result remains. This approach has been highly influential to work in concept learning, as it formalizes inductive concept learning as a search problem—to identify some concept definition out of a space of possible definitions.

This paper presents an analysis of the computational complexity of version spaces. The next section reviews version spaces and the candidate-elimination algorithm. Section 3 then presents the complexity analysis for a single step of the candidate-elimination algorithm. This is followed in Section 4 by an analysis of the complexity for multiple instances in relation to Mitchell's own complexity analysis. Section 5 instantiates the analyses for two commonly used concept description languages. Section 6 summarizes the main results of the paper and presents some concluding remarks.

2 Background

The traditional scenario for inductive concept learning begins with a set of training data—examples classified by an unknown target concept—and a language in which the unknown concept must be expressed (which defines the space of possible generalizations concept learning will search). Mitchell defines a version space to be "the set of all concept descriptions within the given language which are consistent with those training instances" (Mitchell, 1978; page 17). Mitchell noted that the generality of concepts imposes a partial order that allows efficient representation of the version space by the boundary sets S and G representing the most specific and most general concept definitions in the space. The S and G sets delimit the set of all concept definitions consistent with the given data—the version space contains all concepts as or more general than some element in S and as or more specific than some element in G.

Given a new instance, some of the concept definitions in the version space for past data may no longer be consistent with the new instance. 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 example the algorithm generalizes the elements of the S set as little as possible so that they cover the new instance yet remain consistent with past data, and removes those elements of the G set that do not cover the new instance. For a negative instance the algorithm specializes elements of the G set so that they no longer cover the new instance yet remain consistent with past data, and removes from the S set those elements that mistakenly cover the new, negative instance. The unknown concept is determined when the version space has only one element, which in the boundary set representation is when the S and G sets have the same single element.

The process in more detail is as follows. If the instance is positive,

1. Remove those elements of the G set that do not cover the new positive instance.

2. (a) Generalize the elements of the S set as little as possible (if at all) so that they cover the new positive instance.

   (b) Remove from the S set those elements that are not most specific (i.e., remove those more general than some other new S-set element). Also remove those concept definitions that are no longer covered by some element of the G set (i.e., remove those generalizations that now cover some negative instance).

The process for negative instances is symmetric:

1. Remove those elements of the S set that cover the new negative instance.

2. (a) Specialize the elements of the G set as little as possible (if at all) so that they no longer cover the new negative instance.

   (b) Remove from the G set those elements that are not most general and those that no longer cover some element of the S set.

The concept definition is determined when the version space has only one element, which in the boundary set representation is when the S- and G sets have the same single element.

3 Computational Complexity for a Single Update

Upon obtaining each instance, the candidate-elimination algorithm updates the boundary sets for the current version space to reflect the new instance. This section presents the computational complexity of this updating process.

Three fundamental computations are required by the candidate-elimination algorithm. The first tests whether one description is more general than another. This computation is used both to test the relative generality of two concept definitions, as well as to test whether a concept definition covers an instance.[1] The computational complexity of this step will be labeled $c_{above?}$.

The second necessary computation is to determine the minimal generalizations of a description that covers an example. The computational complexity of this step will be labeled c_{gen}. Since thus far there are no constraints on the nature of the description language, there may be more than one such minimal generalization; f will designate the maximum number of such generalizations.

Finally, the third necessary computation is to compute the minimal specializations of a description to exclude an instance but remain above another description. The computational complexity of this step will be labeled c_{spec}. Again, there may be more than one such specialization, and b will designate the maximum number of such specializations.

How an instance is processed by the candidate-elimination algorithm depends on whether the instance is positive or negative. For positive data, removing elements of a G set that don't cover an instance takes $O(|G|c_{above?})$ time, since each G-set element must be tested to check whether it

[1]The analyses of this paper assume that learning uses a concept description language for which the single-representation trick (Dietterich *et al.*, 1982) holds, that is, that for each instance there is a concept definition that covers only that example and no others. While the results presented here do not strictly require this assumption and could be generalized, it would be at the cost of additional complexity in exposition.

covers the given instance. Generalizing elements of the S set to include the new instance takes $O(|S|c_{gen})$, since each S-set element must be generalized to cover the description. The resulting S set will have at most $|S|f$ elements, since every S-set element can be generalized at most f ways. Each pair of these elements must be compared so that nonminimal elements can be pruned, and this takes $O((|S|f)^2 c_{above?})$ time. Finally, each element must be compared to each G-set element, requiring an additional cost of $O(|S||G|f c_{above?})$. Thus the total time is $O(|G|c_{above?} + |S|c_{gen} + (|S|f)^2 c_{above?} + |S||G|f c_{above?})$, or, since the first term can be dropped since it is always less than or equal to the final term, $O(|S|c_{gen} + (|S|f)^2 c_{above?} + |S||G|f c_{above?})$.

The analysis is similar for negative data. Each element of the G set must be specialized to exclude the given instance. Removing S-set elements that cover an instance takes $O(|S|c_{above?})$ time. Specializing G-set elements takes $O(|G|c_{spec})$. The resulting G set will have at most $|G|b$ elements. Pruning nonmaximal elements by pairwise comparisons requires $O((|G|b)^2 c_{above?})$ time. Finally, each element must be compared to each S-set element, requiring additional cost $O(|S||G|b c_{above?})$. The total time is therefore $O(|G|c_{spec} + (|G|b)^2 c_{above?} + |S||G|b c_{above?})$.

4 Computational Complexity for Multiple Instances

The preceding section analyzed the complexity of processing a single instance. This section considers what happens to the computational complexity across multiple examples, and shows that the more fine-grained analysis given here is equivalent to Mitchell's (1982) original analysis.

Mitchell presents the complexity of version spaces as $O(\mathsf{sg}(\mathsf{p}+\mathsf{n}) + \mathsf{s}^2\mathsf{p} + \mathsf{g}^2\mathsf{n})$, where s and g are the largest sizes reached by the S and G sets, and p and n are the number of positive and negative instances processed, using the assumption that $c_{above?}$, $c_{above?}$, and $c_{above?}$ are all constant-time operations. The $\mathsf{sg}(\mathsf{p}+\mathsf{n})$ term arises from comparing every S-set element to every G-set element for each positive and negative instance. The $\mathsf{s}^2\mathsf{p}$ arises for positive data by comparing every element of a new S set to the other new S-set elements so that non-minimal elements can be removed. Similarly, the $\mathsf{g}^2\mathsf{n}$ arises from negative data by comparing pairs of G-set elements.

Mitchell's analysis assumed the cost of comparisons, generalization, and specialization were constant time, and if this were applied to the analysis of the previous section all occurrences of $c_{above?}$, c_{gen}, and c_{spec} can be dropped. This means that the complexity for positive data can be simplified to $O((|S|f)^2 + |S||G|f)$, and for negative data to $O((|G|b)^2 + |S||G|b)$. Furthermore, since the $|S|f$ and $|G|b$ terms are bounded by the maximum size of the new S and G sets after processing a single example, they can be replaced by the bounds s and g, respectively, yielding complexity $O(\mathsf{s}^2 + \mathsf{s}|G|)$ for positive data and $O(\mathsf{g}^2 + |S|\mathsf{g})$ for negative data. Finally, since $|S|$ and $|G|$ are also always bounded by s and g, they too can be similarly replaced, yielding $O(\mathsf{s}^2 + \mathsf{sg})$ for positive data and $O(\mathsf{g}^2 + \mathsf{sg})$ for negative data. If there are p positive instances and n negative instances the total complexity will therefore be $O(\mathsf{p}[\mathsf{s}^2+\mathsf{sg}]+\mathsf{n}[\mathsf{g}^2+\mathsf{sg}]) = O(\mathsf{sg}(\mathsf{p}+\mathsf{n})+\mathsf{s}^2\mathsf{p}+\mathsf{g}^2\mathsf{n})$. This is the same bound given by Mitchell.

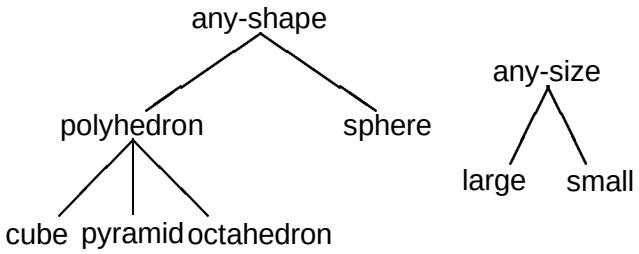

Figure 1: Generalization Hierarchies.

5 Generalization Hierarchies and Ranges

In general, a more detailed analysis of computational complexity can only be made in the context of a particular concept description language—the previous sections did not place constraints on the description languages. This section instantiates the analyses of Section 3 for conjunctive languages over two types of features hierarchies: tree-structured (an example of such a generalization hierarchy is given in Figure 1), and ranges of the form $a \leq x < b$, where a and b must be chosen from a set of prespecified values (e.g., x may range over reals between 0 and 1000, and a and b over integers between 0 and 1000).[2] Throughout this analysis k will be used to designate the number of features.

Features of these forms have two properties that aid in learning, as becomes evident in the following complexity analysis. First, both share the property that any two feature values have a single minimal generalization (i.e., they are upper semi-lattices), and thus $f = 1$, and furthermore if the S set starts singleton it always remains singleton (i.e., $|S| = 1$). Second, there is always only one minimal specialization of a feature to remain above one value but exclude a second, and thus $b = k$. These simplify the complexity analyses of Section 3 to $O(c_{gen} + |G|c_{above?})$ for positive data and $O(|G|c_{spec} + (|G|k)^2 c_{above?})$ for negative data.

Furthermore, $c_{above?}$, c_{gen}, and c_{spec} can all be specified more precisely for both tree-structured and range-valued features. For tree-structured features comparing two feature values for relative generality takes time $O(d)$, where d is the maximum depth of any feature's hierarchy, so $c_{above?}$ is $O(kd)$. For ranges, all that need be tested for each feature is whether the given value is within the corresponding range of the concept definition, and, assuming this takes constant time, $c_{above?}$ is $O(k)$. c_{gen} (computing the minimal generalization of a description to cover an instance) requires time proportional to kd for tree-structured features, and $O(k)$ for ranges. Finally, c_{spec} (computing the minimal specialization of a description to exclude an instance but remain above some other description) again requires $O(kd)$ time for tree-structured features and $O(k)$ time for ranges.

The result of these more refined complexity assessments is that for tree-structured features processing a positive example requires $O(kd + |G|kd) = O(|G|kd)$ time and processing a negative example requires $O(|G|kd + (|G|k)^2 kd) = O(|G|^2 k^3 d)$ time. For ranges, positive data require

[2]Features are of the form $a \leq x < b$ rather than $a < x < b$ or $a \leq x \leq b$ to guarantee that ranges do not overlap, but cover the space of possible values.

$O(k + |G|k) = O(|G|k)$ time and negative data $O(|G|k + (|G|k)^2 k) = O(|G|^2 k^3)$ time.

6 Concluding Remarks

Table 1 summarizes the analyses presented in this paper. A few general lessons can be easily extracted from these results.

1. The computational complexity sorely depends on the size of G sets, especially for the two common languages analyzed here (tree-structured hierarchies and ranges). If the G set is large, a single update will be expensive for both positive and (even more so) negative data.

2. The computational complexity for tree-structured hierarchies and ranges depend on the number of features, only linearly for positive data, but cubically for negative data; again negative data exhibits worse behavior.

3. In the general case, for conjunctive languages complexity is most severely affected by the number of minimal generalizations of two feature values: if the number of minimal generalizations of two feature values is greater than one, f will depend exponentially on the number of features k. This can be contrasted with the effect of the number of maximal specializations of a feature to exclude a value yet remain above another, on which b depends linearly. Thus for feature hierarchies that are not upper semi-lattices learning can depend exponentially on the number of features; moreover, this exponential complexity would appear in processing positive data, in contrast to the previous two cases.

	General Case	Hierarchies	Ranges
Positive Data	$\|S\|c_{gen} + (\|S\|f)^2 c_{above?} + \|S\|\|G\|f c_{above?}$	$\|G\|kd$	$\|G\|k$
Negative Data	$\|G\|c_{spec} + (\|G\|b)^2 c_{above?} + \|S\|\|G\|b c_{above?}$	$\|G\|^2 k^3 d$	$\|G\|^2 k^3$

Table 1: Summary of Results

This paper has focused primarily on the complexity of a single step of the candidate-elimination algorithm, discussing the complexity for a set of instances only in the context of Mitchell's original analysis. The main reason for this is that the behavior of version spaces across multiple instances is already well-known, namely, the size of the G set can grow exponentially in the number of examples even for simple boolean features, as shown using an example set of data by Haussler (1988). The significance of this exponential size is perhaps clarified further from the analyses given in this paper, since it highlights exactly how dependent computational complexity is on G-set size. However, it is interesting to contrast the candidate-elimination algorithm with the INBF version-space algorithm

developed by Smith and Rosenbloom (1990) in light of this. They present an algorithm that keeps the G set singleton throughout learning by saving negative data and only processing a negative instance when it becomes a "near miss" that will not cause the G set to fragment and increase in size. Their algorithm will even keep the G set singleton in cases where Haussler's data is a subset of the training data, as long as the version space for the full data set will ultimately converge. It is an open question as to whether it is possible to process data in general to guarantee that the G set never grows exponentially in size if at the end of learning the final G set has size polynomial in the number of examples. More recent work (Hirsh, 1992) shows that the INBF approach can be taken even further, namely that the list of all negative data can replace the use of the G set for most desired version-space operations and yield guaranteed polynomial-time learning.

Acknowledgments

William Cohen provided helpful comments on an earlier draft of this paper.

References

1. T. G. Dietterich, B. London, K. Clarkson, and G. Dromey. Learning and inductive inference. In P. Cohen and E. A. Feigenbaum, editors, *The Handbook of Artificial Intelligence, Volume III*. William Kaufmann, Los Altos, CA, 1982.

2. D. Haussler. Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence*, 26(2):177–221, Sept. 1988.

3. H. Hirsh. Polynomial-time learning with version spaces. In *Proceedings of the National Conference on Artificial Intelligence*, San Jose, CA, July 1992.

4. T. M. Mitchell. *Version Spaces: An Approach to Concept Learning*. PhD thesis, Stanford University, December 1978.

5. T. M. Mitchell. Generalization as search. *Artificial Intelligence*, 18(2):203–226, March 1982.

6. B. D. Smith and P. S. Rosenbloom. Incremental non-backtracking focusing: A polynomially bounded generalization algorithm for version spaces. In *Proceedings of the National Conference on Artificial Intelligence*, pages 848–853, Boston, MA, August 1990.