NP-Completeness of Searches for Smallest Possible Feature Sets

Scott Davies and Stuart Russell
Computer Science Division
University of California
Berkeley, CA 94720
{sdavies,russell}@cs.berkeley.edu

Abstract

In many learning problems, the learning system is presented with values for features that are actually irrelevant to the concept it is trying to learn. The FOCUS algorithm, due to Almuallim and Dietterich, performs an explicit search for the smallest possible input feature set S that permits a consistent mapping from the features in S to the output feature. The FOCUS algorithm can also be seen as an algorithm for learning determinations or functional dependencies, as suggested in [6]. Another algorithm for learning determinations appears in [7]. The FOCUS algorithm has superpolynomial runtime, but Almuallim and Dietterich leave open the question of tractability of the underlying problem. In this paper, the problem is shown to be NP-complete. We also describe briefly some experiments that demonstrate the benefits of determination learning, and show that finding lowest-cardinality determinations is easier in practice than finding minimal determinations.

Proof of NP-Completeness

Define the *MIN-FEATURES* problem as follows: given a set X of examples (which are each composed of a a binary value specifying the value of the target feature and a vector of binary values specifying the values of the other features) and a number n, determine whether or not there exists some feature set S such that:

- S is a subset of the set of all input features.
- S has cardinality n.
- There exist no two examples in X that have identical values for all the features in S but have different values for the target feature.

We show that *MIN-FEATURES* is NP-complete by reducing *VERTEX-COVER* to *MIN-FEATURES*.[1] The *VERTEX-COVER* problem may be stated as the question: given a graph G with vertices V and edges E, is there a subset V' of V, of size m, such that each edge in E is connected to at least one vertex in V'? We may reduce an instance of *VERTEX-COVER* to an instance of *MIN-FEATURES* by mapping each edge in E to an example in X, with one input feature for every vertex in V.

[1] In [8], a "proof" is reported for this result by reduction *to* set covering. The proof therefore fails to show NP-completeness.

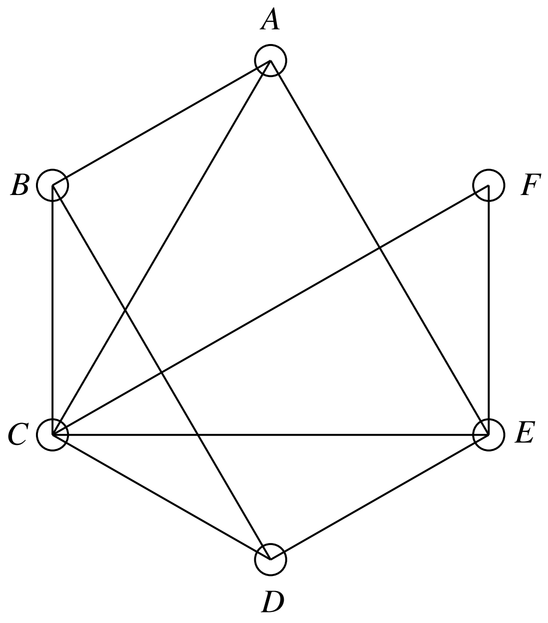

Figure 1: An example graph for *VERTEX-COVER*.

Start with one example with 0's for all input features—one input for every vertex in V—and a 0 for an output. Call this the "null example." Then, for each edge E_i in E, add one example that has a value of 1 for the output and 0's for all input features except the two inputs corresponding to the two vertices to which E_i is connected; for these two inputs, set the input values to 1's. Call these examples the "edge-generated" examples. Finally, use the set size parameter m in the *VERTEX-COVER* problem as the set size parameter n for the *MIN-FEATURES* problem. Figure 1 shows a a graph with labelled vertices, and Figure 2 shows the corresponding *MIN-FEATURES* problem.

If there is an input set S of size n that satisfies the generated *MIN-FEATURES* problem, then there is a vertex cover V' of size $m = n$ that solves the *VERTEX-COVER* problem: simply let V' contain exactly those vertices that correspond with the input features contained in S. This works because for any edge E_i in the graph, S must contain at least one of the two input features corresponding to those vertices to which E_i is connected, since this constraint must have been met in order to prevent the example generated by E_i from matching the "null example" on all the features in S.

Conversely, if there is a vertex set V' of size m that solves an instance of *VERTEX-COVER* problem, then

A	B	C	D	E	F	X	
0	0	0	0	0	0	0	null example
1	1	0	0	0	0	1	
0	1	1	0	0	0	1	
0	0	1	1	0	0	1	
0	0	0	1	1	0	1	
0	0	0	0	1	1	1	edge−generated examples using adjacency matrix
1	0	1	0	0	0	1	
0	0	1	0	1	0	1	
1	0	0	0	1	0	1	
0	1	0	1	0	0	1	
0	0	1	0	0	1	1	

Figure 2: The examples for the corresponding *MIN-FEATURES*.

there is a solution S of size $n = m$ to the generated *MIN-FEATURES* problem—namely, the input set S that contains exactly those input features corresponding to the vertices in V'. Since V' is a solution, there must be at least one vertex in V' that is connected to any given edge in the graph G; this means that, for any "edge-generated" example in the *MIN-FEATURES* problem, at least one of the input features that have values of 1 must be in S. This prevents any "edge-generated" examples from matching the "null example" on all inputs in S, and thus ensures that S is a solution to the the *MIN-FEATURES* problem.

Since the generated *MIN-FEATURES* problem has a solution if and only if the original *VERTEX COVER* problem has a solution, *VERTEX-COVER* has been reduced to *MIN-FEATURES*; the reduction described above is computable in polynomial time. Thus, *MIN-FEATURES* is NP-hard.

MIN-FEATURES is obviously in NP, since we can tell whether or not any given input feature set S is a solution in polynomial time by hashing: for each example X_i, use the values of the input features of X_i that are in S to produce a unique key for the table entry, and store the value of X_i's output feature as the data for the table entry. If two entries have the same key but different data, S is not solution; otherwise, it is a solution.

Therefore, *MIN-FEATURES* is NP-complete. Not only does the problem of finding the smallest relevant feature set S appear intractable, but we cannot even determine the minimum size of such a set.

It is important to note that the proof that *MIN-FEATURES* is NP-complete relies on the generation of problem instances in which all features are relevant. In general, the search for the smallest possible set will take time non-polynomial (unless P = NP) in the number of input features; however, if it turns out that only a few inputs are actually relevant, our reduction does not say that any algorithm must take time that is non-polynomial in the *total* number of features. Hence, while the general problem is intractable, it may still be feasible to use a search along the lines we have been describing if we suspect that only a logarithmic fraction of the inputs are relevant—we can order the search from smaller sets to larger sets, and abort the search if we were mistaken in our suspicions.

The Importance of Relevance

Our algorithm for identifying and using relevant features is as follows. As the examples are processed, we maintain a list L of all the possible feature sets S, of the lowest possible cardinality n, such that any examples seen so far that match on the features in S have identical output feature values. Every time we encounter a new example, we check the validity of each set S in L (by using a hash table as described above), throwing out those sets that are no longer valid. If the list consequently becomes empty, we increase n by 1, and add to the list all feature sets S of the new cardinality $n' = n + 1$ that are now valid.

When asked to predict an output for a given input vector, an algorithm M maintaining the list L may pick a random set S from L, pass on the features in S to any other machine learning algorithm M', and return the value returned by M'. Alternatively, it may call M' once for every set S in L and return the value predicted most often, or return a distribution based on the values predicted. This takes more time, however, and although the resulting learning curves are more predictable, they often have the same averages.

Using a basic implementation of Quinlan's ID3 decision-tree learning algorithm [5] for M', we tested the above algorithm on random boolean functions of 16 boolean variables, only 5 of which were relevant. Three trials were performed, each with a different random function; for each trial, up to 150 of the 3200 examples generated were used for training, while the rest were used for testing. Figure 3 shows the resulting learning curve, and contrasts it with the learning curve that was generated when ID3 was used directly on all of the feature values of the same three data sets. It is clear that the system learned much more quickly when we used a FOCUS-like algorithm to filter out features which could conceivably be irrelevant.

Some papers ([7], for example) have put forward algorithms for determining all feature sets S that are "minimal" in the sense that any proper subset of S would be insufficient to permit a consistent mapping from the features in the proper subset to the output feature. The list

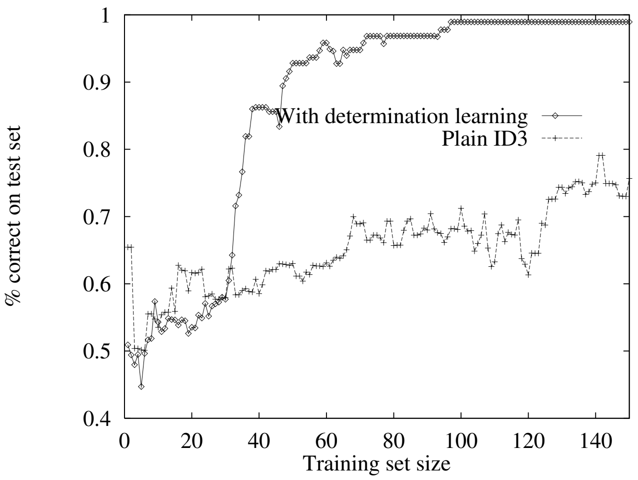

Figure 3: Learning curves for ID3 with feature set selection versus raw ID3 (sixteen features, five relevant).

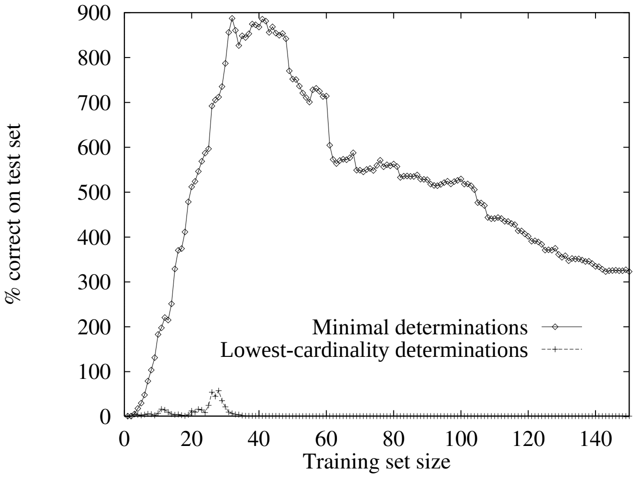

Figure 4: Number of "minimal" vs. number of lowest-cardinality sets

L maintained by the algorithm above contains a subset of these "minimal" sets, i.e., the ones of the lowest possible cardinality. Empirically, this list L is often much smaller than the list of all "minimal" feature sets, and is therefore much quicker to update when a new example is presented to the system. Figure 4 shows the average number of "minimal" sets that existed during the generation of the learning curve in Figure 3, and contrasts this with the much smaller number of sets that were actually of the lowest possible cardinality. Regardless of the method by which the minimal sets were used to form a feature set for ID3, we found that the inductive performance of the two approaches was more or less the same.

It is interesting to note that the minimum-cardinality approach settled on a single determination in all three trials after about 30 examples—the same point at which the learning curve shows a dramatic improvement. This suggests strongly that learning the determination, in addition to the decision tree itself, is an important part of inductive performance.

Conclusion

FOCUS-type algorithms are fairly simple to implement, and can significantly improve the performance of machine learning programs faced with certain types of problems. In general, however, their approach to searching for relevant features appears to be intractable. Furthermore, they must be significantly changed if they are to work properly for noisy or floating-point functions. John *et al.* [4] suggest that it may be better to select a feature set that yields the best inductive performance by M' (as measured by cross-validation), rather than simply identifying an exactly relevant feature set. This approach seems promising, and more likely to tolerate noise.

The problem may tractable if we allowed to make certain assumptions about the domains in which the machine learning system is working. If the learning system is allowed to make queries about the outputs corresponding to arbitrary input vectors, then the determination of smallest possible feature sets for discrete data becomes completely tractable [3]. If it is known that the function being learned is of a certain class (e.g., K-CNF or K-DNF), then the problem once again becomes tractable [2].

References

[1] Almuallim, H. and Dietterich, T. (1991) "Learning with Many Irrelevant Features." In *Proc. AAAI-91.*

[2] Blum, A. 1990. "Learning Boolean Functions in an Infinite Attribute Space," *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing,* pp. 64-72. Baltimore, MD.

[3] Blum, A.; Hellerstein, L.; and Littlestone, N. 1991. "Learning in the Presence of Finitely or Infinitely Many Irrelevant Attributes," *Proceedings of the Fourth Annual Workshop on Computational Learning Theory.* Santa Cruz, CA: Morgan Kaufman.

[4] John, G. H., Kohavi, R., and Pfleger, K. 1994. "Irrelevant features and the subset selection problem." In *Proceedings of ML-94.* New Brunswick, NJ: Morgan Kaufmann.

[5] Quinlan, R. 1986. "Induction of Decision Trees," *Machine Learning,* Kluwer Academic, 1:1.

[6] Russell, S. J. (1989) *The Use of Knowledge in Analogy and Induction.* London: Pitman Press.

[7] Schlimmer, J. (1993) "Efficiently inducing determinations: A complete and systematic search algorithm that uses optimal pruning." In *Proc. ML-93.*

[8] Wong, S. K. M., and Ziarko, W. (1985) "On optimal decision rules in decision tables." *Bull. Polish Acad. Sci. (Math.),* **33**(11-12), 693-696.