Relevant Examples and Relevant Features: Thoughts from Computational Learning Theory

Avrim L. Blum*
School of Computer Science
Carnegie Mellon University
Pittsburgh PA 15213-3891

Introduction

It can be said that nearly all results in machine learning, whether experimental or theoretical, deal with problems of separating relevant from irrelevant information in some way. In this paper I will attempt to survey some of the results and intuitions developed in the area of computational learning theory. My focus will be on two issues in particular: that some examples may be more relevant than others, and that within an example, some features may be more relevant than others. This survey is by no means even close to comprehensive, and strongly reflects my own personal biases as well as issues brought up by results presented at this workshop.

Issues of relevance are fundamental in the theoretical study of machine learning. In particular, questions regarding the meaning of a "relevant" or "informative" example are key motivations for the most popular and most basic theoretical models. Let me begin in the traditional manner of defining the basic models discussed, but do so from the point of view of the motivations from "relevance."

Models and preliminaries

A common paradigm for concept learning is to have a "training phase" in which the learning procedure studies some set of examples and produces a hypothesis, and then a separate "testing phase" in which that hypothesis is evaluated. The Probably-Approximately-Correct or "PAC" learning model of Valiant [Valiant, 1984], [Kearns *et al.*, 1987] is a model of concept learning of this form. It is currently the most popular theoretical model studied.

The motivation behind the PAC model can be thought of as follows. In order for a learning algorithm to be able to generalize from its training data and produce a hypothesis of good quality for the testing phase, the training examples need to be "relevant" to the final test in some way. For instance, if the goal of a learning program was to distinguish between pencils and pens, and all pencils in the training phase were wood but all in the testing phase were mechanical, one would not expect to see good performance (and the test would seem a bit unfair). One of the key ideas of the PAC model is to postulate a fixed (but unknown) distribution on examples, and assume that the algorithm will be trained *and* tested on this same distribution. Because there is always the possibility that examples selected for training may not be representative of the distribution (and therefore not so relevant), this model requires two "fudge" parameters: an error ϵ and a confidence δ. We say that an algorithm succeeds in learning a "class" or set of concepts C in the PAC model if for any distribution $\mathcal{D}$ on examples and any target concept $c \in C$, the algorithm with probability at least $1 - \delta$ produces a hypothesis with error at most ϵ with respect to $\mathcal{D}$. We generally will require an algorithm to run in time polynomial in various parameters such as $1/\epsilon$, $1/\delta$, and the description length of the target concept under some agreed-upon encoding.

Perhaps one of the most important general results in learnability theory is that in the PAC model, one can effectively say that "learning is data compression". In particular, if one can take a large sample of data and extract the relevant information into a rule significantly smaller than the data set, then this is guaranteed to achieve good generalization [Blumer *et al.*, 1987] [Blumer *et al.*, 1989]. Conversely, any good learning algorithm can either be viewed as behaving in this manner or transformed into one that does [Board and Pitt, 1992] [Schapire, 1990].

The mistake-bound model [Littlestone, 1988] is a model for on-line prediction motivated in part by this same basic issue. For an on-line prediction scenario (ad infinitum you are given an unlabeled example, asked to predict its classification, and then told the correct answer) we would like to say that a good algorithm will "converge quickly." However if the same example is repeated over and over, we cannot expect this to happen. So, in the mistake-bound model, we just count the number of mistakes made and use that quantity,

*This material is based upon work supported under NSF National Young Investigator grant CCR-9357793 and a Sloan Foundation Research Fellowship. Email: avrim@cs.cmu.edu

rather than the total number of examples seen, as our measure of time. In particular, in this model, we say that an algorithm succeeds to learn a concept class if the number of mistakes it makes given that examples are classified by some concept in the class is guaranteed to be at most some polynomial in certain relevant parameters (the number of features and the description length of the target concept).

In this paper I will use the terms "variable", "feature", and "attribute" interchangeably. The features will be named $x_1, x_2, \ldots, x_n$ and an example is an assignment of some value (usually 0 or 1) to each of them.

Filtering irrelevant examples

Boosting

The PAC model guarantees that the distribution used for training will be relevant to the success criteria used to test hypotheses. However, as learning progresses, the learner's knowledge about certain parts of the input space increases, and examples in the "well-understood" portion of the space become less useful. Thus, if one has an algorithm for which *training* is expensive but *prediction* is cheap, then as its hypotheses improve, it makes sense to try to filter the training data to weed out those less useful examples. As discussed in Freund's paper in this workshop, the boosting technique of Schapire [Schapire, 1990] and Freund [Freund, 1990; Freund, 1992] provides a mechanism for doing this.

To get some intuition about the degree of filtering one might hope to achieve, it is helpful to consider the simpler case of *conservative* algorithms. A conservative algorithm only pays attention to an example if its current hypothesis predicts incorrectly on it. Since such an algorithm ignores all the examples on which its hypothesis is correct, it is clear one can filter those out without causing harm. Thus, when the algorithm is predicting with 20% error one can filter out 80% of the examples, and when it achieves 10% error one can filter out 90% of the examples and so forth. Learning algorithms in the PAC model can be assumed to need to roughly double the number of examples seen in order to halve their error rate [Schapire, 1990], [Freund, 1992], [Blumer *et al.*, 1989]. Since the number of example passing through the filter is proportional to the error rate, however, the number of new examples actually *used* by the algorithm each time it wishes to halve its error rate remains (roughly) constant. Thus the number of examples actually used is really just logarithmic in $1/\epsilon$ rather than linear. The boosting techniques enable one to achieve this kind of filtering in much more general conditions.

It is interesting to relate this use of boosting to its original motivations. Boosting was developed as an answer to the following puzzling question: Why are there no classes known that are "somewhat learnable" in the PAC model in the sense that some algorithm achieves, say, 40% error under arbitrary distributions but no algorithm can achieve an arbitrarily small error rate? Schapire [1990] proved that no such class could exist (i.e. weak-learnability implies strong-learnability) by providing a general technique to transform any algorithm of the former type into an algorithm of the latter type. The intuitive idea is that since the former algorithm is supposed to be (weakly) successful over *every* distribution, you can improve any mediocre hypothesis by applying the weak algorithm to a modified (filtered) distribution on which your current hypothesis gives you very little information. In other words, filtering to focus on the current "hard cases" is not just a method for focusing on the relevant examples, but is also a way of forcing one's algorithm to do something useful with them.

Perhaps the biggest contribution of the boosting technique to computational learning theory is to point out that in the standard PAC model, the quantity ϵ is "irrelevant" in the same sense that the quantity δ can be seen to be "irrelevant" by more elementary methods. Namely, one may as well assume that $\epsilon = \delta = 1/3$ and attack that case. Boosting will take care of the rest.

Query by Committee

As described in Freund's paper, the Query by Committee (QBC) algorithm allows one to filter examples in a stronger sense than boosting: examples can be filtered out without even knowing their labels. This is useful if the labeling process is expensive. QBC assumes a Bayesian prior distribution on the target concepts. This assumption means there is a natural notion of the information in an unlabeled example: how well does it split the concepts currently under consideration? The basic idea of QBC is to allow through only the more informative examples, and the assumptions imply that if such examples are rare then random consistent hypotheses are likely to be quite good. What QBC requires, however, is having a much better handle on the space of hypotheses than boosting requires. To use QBC one needs to be able to sample consistent concepts according to the prior, which can be quite difficult, although it is also a major topic of algorithmic research, e.g., [Sinclair and Jerrum, 1989], [Dyer *et al.*, 1989], [Lovász and Simonovits, 1992].

Comments

It is useful to keep in mind a distinction between examples that are relevant from the point of view of *information*, and examples that are relevant or helpful from the point of view of one's algorithm. In this regard, let us consider the model in which in addition to seeing examples from a distribution, one is allowed to select one's own unlabeled examples and ask for their labelings. These requests are often called *membership queries*. This ability to make active queries seems to greatly enlarge the types of concept classes for which one can guarantee polynomial-time learning [Angluin, 1987], [Angluin *et al.*, 1989], [Hancock, 1991], [Bshouty, 1993], [Jackson, 1994].... One very common technique used in these algorithms is to take some example that one wants to understand better and slightly tweak it to see how this would affect its labeling. For instance one might take two examples with different labelings and slowly "walk" one example to the other to see when the classification changes. From the perspective of "relevance" the point here is that even though these slight variations may not be the most relevant from an information-theoretic viewpoint, the type of information that one gets is in an especially usable form. This phenomenon has been noticed for quite some time [Winston, 1975].

Relevant Features

Much of the work in machine learning theory regarding "relevancy" concerns the issue of relevant features. Given the setup of the basic models, one can define a feature x_i to be relevant to a given target concept f if there exists a pair of examples A and B in the input space such that A and B differ only in their assignment to x_i and $f(A) \neq f(B)$. Notice that in this definition, "relevancy" depends on the target concept, and not the distribution or the set of examples seen by the learner. For instance, it might be that in all the examples seen by the learner so far, the values of features x_1 and x_2 are exactly the same, but x_1 could be deemed relevant and x_2 not. A discussion of a useful way to view this definition is given in the comments section below.

In [Blum and Kannan, 1993] (also presented at this workshop), a generalization of this notion of relevance to the idea of relevant linear combinations of features is addressed.

The question most commonly addressed in theoretical work with respect to the issue of relevant features can be stated roughly as follows:

> For a given concept class, is there a polynomial-time algorithm whose performance is especially good when the number of relevant features is small? That is, can one devise an algorithm whose sample size or mistake bound has a low dependence on the number of irrelevant features?

These sorts of algorithms are termed "attribute efficient" in [Blum *et al.*, 1991]. A similar question, addressed by the work of [Dhagat and Hellerstein, 1994] (presented at this workshop), is whether one can produce a hypothesis that also itself depends only on a small subset of features.

In the framework of computational learning theory, two main techniques have come to light for achieving performance guarantees of the form described above. In the PAC model, the technique is "greedy set cover," first used in this context by Haussler [Haussler, 1986]. In the mistake-bound model, the technique is that of multiplicative weight updating, introduced by Littlestone [Littlestone, 1988].

The greedy set cover algorithm, applied to the case of learning a disjunction of boolean variables, is as follows. Given a training set, ignore any variable with the property that some negative example sets it to 1. For each variable remaining, count the number of positive examples that do set it to 1. Pick the variable x_i whose count is highest and place it into the hypothesis disjunction. Now throw away those positive examples with $x_i = 1$ and repeat. The guarantee [Johnson, 1974] is that the number of variables included will be at most a logarithmic factor (in n) larger than the smallest possible.* This approach can be applied directly to other situations where the learning algorithm can produce a list of functions with the property that the target concept can be viewed as a disjunction (or conjunction) of some small set of them. Situations of this form include learning intersections of halfspaces in constant-dimensional spaces [Blumer *et al.*, 1989], learning DNF formulas in $n^{O(\log n)}$ time under the uniform distribution [Verbeurgt, 1990], and many others.

The greedy set-cover approach is the basis for the results of Dhagat and Hellerstein presented in this workshop. They extend this technique in a recursive fashion to apply to other classes of target functions.

In the setting of on-line learning, Littlestone [Littlestone, 1988] was the first to demonstrate that in interesting cases one could achieve a mistake bound with provably low dependence on the number of irrelevant features. His Winnow algorithms apply to learning a variety of concept classes such as conjunctions, k-DNF, and linear threshold functions with good separation between positive and negative examples. For these, he gives a very elegant proof that the number of mistakes made has only a logarithmic dependence on the number of irrelevant variables. For instance, the number of mistakes made in learning a disjunction of r out of n variables is $O(r \log n)$. His general technique involves attaching weights to each element in a given set of "predictors" (for instance to learn disjunctions one would assign a weight to each variable). These weights represent the perceived importance of each one. The weights are then updated in a multiplicative manner based on the examples seen. This general approach has then been used in other algorithms such as the weighted majority algorithm of Littlestone and Warmuth [1994], and the algorithms of [Vovk, 1990; Littlestone *et al.*, 1991; Cesa-Bianchi *et al.*, 1993]. Kivinen and Warmuth in this workshop describe relations between these approaches and additive updating methods such as the Widrow-Hoff algorithm [Widrow and Hoff, 1960].

*Finding the smallest consistent disjunction is NP-hard [Garey and Johnson, 1979]; in fact, a polynomial-time algorithm to find disjunctions only $c \log n$ times larger than the smallest for $c < 1/4$ would place NP into quasi-polynomial time [Lund and Yannakakis, 1993].

One general fact regarding attribute-efficient algorithms is that if membership queries are allowed, then any mistake-bound algorithm can be converted into one that has only a logarithmic dependence (in terms of number of mistakes plus queries made) on the number of irrelevant features [Blum *et al.*, 1991].

Comments

It is worth pointing out that in the PAC and mistake-bound models (without membership queries) the notion of relevance is really being used as a *measure of complexity*. In other words, the algorithms do not really care what features are relevant and (generally) they are not really searching for a set of relevant ones. Rather, relevance is used to say how "complicated" a target function is, and the goal of the algorithms is to perform well when this quantity is low.

Suppose, for instance, we saw the following examples:

```
111111111100000000000000000000 +
000000000011111111110000000000 +
000000000000000000001111111111 +
000000000000000000000000000000 -
```

According to our definition of relevance, one cannot tell which features are relevant from the above data since that depends on the unknown target concept. In a practical sense, if somebody asks *which* features are relevant, perhaps all you can say is that they are all somewhat relevant (see the definition in [Kohavi, 1994]). On the other hand, another useful thing one can say is "3". Namely, 3 is the size of the smallest relevant set, assuming we are learning a class like "disjunctions" where it is possible to be consistent with the data.

One other point: algorithms such as the Winnow and Weighted-Majority algorithm have the following nice property related to their ability to focus on the relevant features. By dropping predictors (or "experts") whose weights become too low, the algorithm can be made to actually speed up as it learns more. (Dropping bad predictors can be justified under distributional assumptions). This behavior is in contrast to a large number of algorithms in use that slow down as they learn more because their internal representations get increasingly complicated.

Other thoughts

It is my belief that some of the most central open problems in computational learning theory are, at their core, questions about finding relevant variables. In this last section, I will describe one of my personal favorite open questions. I believe this is the essence of the difficulty in learning DNF formulas and decision trees (without membership queries) in the PAC model. The problem is the following:

> Does there exists a polynomial time algorithm for learning the class of boolean functions over $\{0,1\}^n$ that have $\log_2(n)$ relevant variables, when the example distribution $\mathcal{D}$ is the uniform distribution?

A few points regarding this question: First, notice that the class of functions having $\log_2 n$ relevant variables is a special case of polynomial-size DNF formulas and decision trees. That is because any function which has only $\log_2 n$ relevant variables can, by definition, be written as a truth table having only n entries. Second, this class is easy to learn when membership queries are allowed (exercise for reader). Indeed, the algorithm of Bshouty [Bshouty, 1993] learns the larger class of decision trees with membership queries in the exact leaning model, and a recent algorithm of Jackson [Jackson, 1994] learns the even larger class of general DNF formulas using membership queries, with respect to the uniform distribution. Third, *random* functions in this class, namely functions with random truth tables, turn out to be easy to learn. The hard cases seem to be close to, but not exactly, parity functions (see below). Finally, the base "2" in the logarithm is just for concreteness. Any constant base greater than 1 will do.

As a practical matter, one difficulty with the above open problem is that since no distribution on the *target functions* is stated, it is unclear how to experimentally test a proposed algorithm. To handle this issue, the following is a *specific* distribution on the target functions that seems quite hard (for convenience, the number of relevant variables will be $2\log_2 n$).

> Select at random two disjoint sets $A, B \subset \{1, \ldots, n\}$ each of size $\log_2 n$. On input x, compute the exclusive-or of the bits indexed by A and the majority function of the bits indexed by B, and output the exclusive-or of the two results.

For instance, if $A = \{1,2,3\}$ and $B = \{4,5,6\}$ then the classification of the example 011101001010 would be positive since the first 3 bits have XOR equal to 0, and the majority function applied to the next 3 bits is 1, and the XOR of those two quantities is 1.

Formally, the associated open question is this. Does there exist a polynomial-time algorithm A, a polynomial p, and a constant $\epsilon > 0$ such that with probability at least $1/2 + \epsilon$, A passes the following test: given $p(n)$ uniformly-selected random examples classified by a target function drawn from the above distribution, it correctly classifies a new unlabeled random example.

One interesting feature about this open question is that in [Blum *et al.*, 1994] it is shown how to create a pseudorandom bit generator with the property that if the answer to this question is "no", then the generator is secure (against polynomial-time adversaries).

Conclusions

Issues of relevant examples and relevant features are central to the theoretical study of machine learning. This paper has attempted to survey just a few of the many results related to this topic.

References

D. Angluin, L. Hellerstein, and M. Karpinski. Learning read-once formulas with queries. Technical Report UCB/CSD 89/528, University of California Berkeley Computer Science Division, 1989.

Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, November 1987.

A. Blum and R. Kannan. Learning an intersection of k halfspaces over a uniform distribution. In *Proceedings of the* 34^{th} *Annual IEEE Symposium on Foundations of Computer Science*, pages 312–320, November 1993.

A. Blum, L. Hellerstein, and N. Littlestone. Learning in the presence of finitely or infinitely many irrelevant attributes. In *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, pages 157–166, Santa Cruz, California, August 1991. Morgan Kaufmann.

A. Blum, M. Furst, M. Kearns, and D. Lipton. Cryptographic primitives based on hard learning problems. In D. Stinson, editor, *Proceedings CRYPTO 93*. Springer, 1994. Lecture Notes in Computer Science No. 773.

Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Occam's razor. *Information Processing Letters*, 24:377–380, April 1987.

Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, 36(4):929–965, 1989.

Raymond Board and Leonard Pitt. On the necessity of Occam algorithms. *Theoretical Computer Science*, 100:157–184, 1992.

N. H. Bshouty. Exact learning via the monotone theory. In *Proceedings of the IEEE Symp. on Foundation of Computer Science*, pages 302–311, Palo Alto, CA., 1993.

N. Cesa-Bianchi, Y. Freund, D.P. Helmbold, D. Haussler, R.E. Schapire, and M.K. Warmuth. How to use expert advice. In *Proceedings of the Annual ACM Symp. on the Theory of Computing*, pages 382–391, 1993.

A. Dhagat and L. Hellerstein. PAC learning with irrelevant attributes. In *Proceedings of the IEEE Symp. on Foundation of Computer Science*, 1994. To appear.

M. Dyer, A. Frieze, and R. Kannan. A random polynomial time algorithm for approximating the volume of convex bodies. In *Proceedings of the Annual ACM Symp. on the Theory of Computing*, pages 375–381, 1989.

Yoav Freund. Boosting a weak learning algorithm by majority. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 202–216. Morgan Kaufmann, 1990.

Yoav Freund. An improved boosting algorithm and its implications on learning complexity. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, pages 391–398. ACM Press, 1992.

M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco, 1979.

Thomas Hancock. Learning 2μ DNF formulas and $k\mu$ decision trees. In *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, pages 199–209, Santa Cruz, California, August 1991. Morgan Kaufmann.

David Haussler. Quantifying the inductive bias in concept learning. In *Proceedings AAAI-86*, pages 485–489. American Association for Artificial Intelligence, August 1986.

J. Jackson. An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. In *Proceedings of the IEEE Symp. on Foundation of Computer Science*, 1994. To Appear.

D.S. Johnson. Approximation algorithms for combinatorial problems. *J. of Computer and System Sciences*, 9:256–278, 1974.

Michael Kearns, Ming Li, Leonard Pitt, and Leslie Valiant. On the learnability of boolean formulae. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, pages 285–295, New York, New York, May 1987.

Ron Kohavi. Feature subset selection as search with probability estimates. This workshop, 1994.

N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994.

Nick Littlestone, Philip M. Long, and Manfred K. Warmuth. On-line learning of linear functions. In *Proceedings of the Twenty-third Annual ACM Symposium on Theory of Computing*, pages 465–475, New Orleans, Louisiana, May 1991. ACM.

Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

L. Lovász and M. Simonovits. On the randomized complexity of volume and diameter. In *Proceedings of the IEEE Symp. on Foundation of Computer Science*, pages 482–491, 1992.

C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. In *Proceedings of the Annual ACM Symp. on the Theory of Computing*, pages 286–293, 1993.

Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

Alistair Sinclair and Mark Jerrum. Approximate counting, uniform generation and rapidly mixing markov chains. *Information and Computation*, 82(1):93–133, July 1989.

Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.

K. Verbeurgt. Learning DNF under the uniform distribution in polynomial time. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 314–325. Morgan Kaufmann, 1990.

V. Vovk. Aggregating strategies. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 371–383. Morgan Kaufmann, 1990.

Bernard Widrow and Marcian E. Hoff. Adaptive switching circuits. *1960 IRE WESCON Convention Record*, pages 96–104, 1960. Reprinted in *Neurocomputing* (MIT Press, 1988).

Patrick H. Winston. Learning structural descriptions from examples. In Patrick H. Winston, editor, *The Psychology of Computer Vision*. McGraw-Hill, New York, 1975.