[17] D. Haussler. Generalizing the PAC model for neural net and other learning applications. *Information and Computation*, 1990. to appear.

[18] D. Haussler. Learning conjunctive concepts in structural domains. *Machine Learning*, 4:7–40, 1989.

[19] D. Haussler. Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence*, 36:177–221, 1988.

[20] D. Haussler, M. Kearns, N. Littlestone, and M. K. Warmuth. Equivalence of models for polynomial learnability. *Information and Computation*, 1990. to appear.

[21] D. Haussler, N. Littlestone, and M. Warmuth. Predicting 0,1-functions on randomly drawn points. In *Proceedings of the 29th Annual Symposium on the Foundations of Computer Science*, pages 100–109, IEEE, 1988.

[22] D. Haussler and L. Pitt, editors. *Proceedings of the 1988 Workshop on Computational Learning Theory.* Morgan Kaufmann, San Mateo, CA, 1988.

[23] M. A. John Shawe-Taylor and N. Biggs. *Bounding Sample Size with the Vapnik-Chervonenkis Dimension.* Technical Report CSD-TR-618, University of London, Surrey, England, 1989.

[24] M. Kearns and M. Li. Learning in the presence of malicious errors. In *20th ACM Symposium on Theory of Computing*, pages 267–279, Chicago, 1988.

[25] M. Kearns, M. Li, L. Pitt, and L. Valiant. On the learnability of boolean formulae. In *19th ACM Symposium on Theory of Computing*, pages 285–295, New York, 1987.

[26] M. Kearns and R. Schapire. Efficient distribution-free learning of probabilistic concepts. 1990. manuscript.

[27] M. Kearns and L. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. In *21st ACM Symposium on Theory of Computing*, pages 433–444, Seattle, WA, 1989.

[28] N. Littlestone. From on-line to batch learning. In *Proceedings of the 2nd Workshop on Computational Learning Theory*, pages 269–284, published by Morgan Kaufmann, 1989.

[29] N. Littlestone. Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[30] N. Littlestone. *Mistake Bounds and Logarithmic Linear-threshold Learning Algorithms.* PhD thesis, University of Calif., Santa Cruz, 1989.

[31] N. Littlestone and M. K. Warmuth. The weighted majority algorithm. In *30th Annual IEEE Symposium on Foundations of Computer Science*, pages 256–261, 1989.

[32] T. Mitchell. *The need for biases in learning generalizations.* Technical Report CBM-TR-117, Rutgers University, New Brunswick, NJ, 1980.

[33] B. K. Natarajan. On learning sets and functions. *Machine Learning*, 4(1), 1989.

[34] L. Pitt. *Inductive Inference, DFAs, and Computational Complexity.* Technical Report UIUCDCS-R-89-1530, U. Illinois at Urbana-Champaign, 1989.

[35] L. Pitt and L. Valiant. Computational limitations on learning from examples. *J. ACM*, 35(4):965–984, 1988.

[36] R. Rivest, D. Haussler, and M. Warmuth, editors. *Proceedings of the 1989 Workshop on Computational Learning Theory.* Morgan Kaufmann, San Mateo, CA, 1989.

[37] R. L. Rivest. Learning decision lists. *Machine Learning*, 2:229–246, 1987.

[38] D. Rumelhart. 1990. personal communication.

[39] W. Sarrett and M. Pazzani. *Average case analysis of empirical and explanation-based learning algorithms.* Technical Report 89-35, UC Irvine, 1989.

[40] G. Tesauro and D. Cohn. Experimental tests of statistical learning theories. In *Snowbird conference on Neural Networks for Computing,* 1990. unpublished manuscript.

[41] L. G. Valiant. Learning disjunctions of conjunctions. In *Proc. 9th IJCAI*, pages 560–6, Los Angeles, August 1985.

[42] L. G. Valiant. A theory of the learnable. *Comm. ACM*, 27(11):1134–42, 1984.

[43] V. N. Vapnik. *Estimation of Dependences Based on Empirical Data.* Springer-Verlag, New York, 1982.

proposes an instance in the instance space and then is told whether or not this instance is a member of the target concept. The ability to make membership queries can greatly enhance the ability of an algorithm to efficiently learn the target concept in both the mistake bound and PAC models. It has been shown that there are polynomial time algorithms that make polynomially many membership queries and have polynomial worst case mistake bounds for learning

1. monotone DNF concepts (Disjunctive Normal Form with no negated variables) [3],
2. μ-formulae (Boolean formulae in which each variable appears at most once) [5],
3. deterministic finite automata [2], and
4. Horn sentences (propositional PROLOG programs) [4].

In addition, there is a general method for converting an efficient learning algorithm that makes membership queries and has a polynomial worst case mistake bound into a PAC learning algorithm, as long as the PAC algorithm is also allowed to make membership queries. Hence, all of the concept classes listed above are PAC learnable when membership queries are allowed. This contrasts with the evidence from cryptographic assumptions that classes (2) and (3) above are not PAC learnable from random examples alone [27].

8 Conclusion

In this brief survey we were able to cover only a small fraction of the results that have been obtained recently in computational learning theory. For a glimpse at some of these further results we refer the reader to [22,36]. However, we hope that we have at least convinced the reader that the insights provided by this line of investigation, such as those about the difficulty of searching hypothesis spaces, the notion of bias and its effect on required training size, the effectiveness of majority voting methods, and the usefulness of actively making queries during learning, have made this effort worthwhile.

References

[1] J. Amsterdam. *The Valiant Learning Model: Extensions and Assessment.* Master's thesis, MIT Department of Electrical Engineering and Computer Science, Jan. 1988.

[2] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, Nov. 1987.

[3] D. Angluin. Queries and concept learning. *Machine Learning*, 2:319–342, 1988.

[4] D. Angluin, M. Frazier, and L. Pitt. Learning conjunctions of horn clauses. 1990. manuscript.

[5] D. Angluin, L. Hellerstein, and M. Karpinski. Learning read-once formulas with queries. *JACM*, 1990. to appear.

[6] D. Angluin and P. Laird. Learning from noisy examples. *Machine Learning*, 2(4):343–370, 1988.

[7] E. Baum. When are k-nearest neighbor and back propogation accurate for feasible sized sets of examples. In *Snowbird conference on Neural Networks for Computing*, 1990. unpublished manuscript.

[8] G. M. Benedek and A. Itai. Learnability by fixed distributions. In *Proc. 1988 Workshop on Comp. Learning Theory*, pages 80–90, Morgan Kaufmann, San Mateo, CA, 1988.

[9] F. Bergadano and L. Saitta. On the error probabilty of boolean concept descriptions. In *Proceedings of the 1989 European Working Session on Learning*, pages 25–35, 1989.

[10] A. Blum and R. L. Rivest. Training a three-neuron neural net is NP-Complete. In *Proceedings of the 1988 Workshop on Computational Learning Theory*, pages 9–18, published by Morgan Kaufmann, San Mateo, CA, 1988.

[11] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *JACM*, 36(4):929–965, 1989.

[12] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Occam's razor. *Information Processing Letters*, 24:377–380, 1987.

[13] W. Buntine. *A Theory of Learning Classification Rules.* PhD thesis, University of Technology, Sydney, 1990. Forthcoming.

[14] T. M. Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Trans. on Electronic Computers*, EC-14:326–334, 1965.

[15] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* Wiley, 1973.

[16] S. E. Hampson and D. J. Volper. Linear function neurons: structure and training. *Biol. Cybern.*, 53:203–217, 1986.

of this set-up using the discrete loss function, but with the added twist that learning performance is measured with respect to the worst case over all joint distributions in which the entire probability measure is concentrated on a set of examples that are consistent with a single target concept of a particular type. Hence, in the PAC case it is possible to get arbitrarily close to zero loss by finding closer and closer approximations to this underlying target concept. This is not possible in the general case, but one can still ask how close the hypothesis produced by the learning algorithm comes to the performance of the best possible hypothesis in the hypothesis space. For an unbiased hypothesis space, the latter is known as Bayes optimal classifier [15].

Some recent PAC research has used this more general framework. By using the quadratic loss function mentioned above in place of the discrete loss, Kearns and Shapire investigate the problem of efficiently learning a real-valued regression function that gives the probability of a "+" classification for each instance [26]. In [17] it is shown how the VC dimension and related tools, originally developed by Vapnik, Chervonenkis, and others for this type of analysis, can be applied to the study of learning in neural networks. Here no restrictions whatsoever are placed on the joint probability distribution governing the generation of examples, i.e. the notion of a target concept or target class is eliminated entirely.

7 Other Theoretical Learning Models

A number of other theoretical approaches to machine learning are flourishing in recent computational learning theory work. One of these is the *total mistake bound* model [29]. Here an arbitrary sequence of examples of an unknown target concept is fed to the learning algorithm, and after seeing each instance the algorithm must predict the label of that instance. This is an incremental learning model like the probability of mistake model described above, however here it is not assumed that the instances are drawn at random, and the measure of learning performance is the *total* number of mistakes in prediction in the worst case over all sequences of training examples (arbitrarily long) of all target concepts in the target class. We will call this latter quantity the *(worst case) mistake bound* of the learning algorithm. Of interest is the case when there exists a polynomial time learning algorithm for a concept class $\mathbf{C} = \{C_n\}_{n \geq 1}$ with a worst case mistake bound for target concepts in C_n that is polynomial in n. As in the PAC model, mistake bounds can also be allowed to depend on the syntactic complexity of the target concept.

The perceptron algorithm for learning linear threshold functions in the Boolean domain is a good example of a learning algorithm with a worst case mistake bound. This bound comes directly from the bound on the number of updates given in the perceptron convergence theorem (see e.g. [15]). The worst case mistake bound of the perceptron algorithm is polynomial (and at least linear) in the number n of Boolean attributes when the target concepts are conjunctions, disjunctions, or any concept expressible with 0-1 weights and an arbitrary threshold [16]. A variant of the perceptron learning algorithm with multiplicative instead of additive weight updates was developed that has a significantly improved mistake bound for target concepts with small syntactic complexity [29]. The performance of this algorithm has also been extensively analysed in the case when some of the examples may be mislabeled [30].

It can be shown that if there is a polynomial time learning algorithm for a target class $\mathbf{C}$ with a polynomial worst case mistake bound, then $\mathbf{C}$ is PAC learnable. General methods for converting a learning algorithm with a good worst case mistake bound into a PAC learning algorithm with a low sample complexity are given in [28]. Hence, the total mistake bound model is actually not unrelated to the PAC model.

Another fascinating transformation of learning algorithms is given by the *weighted majority method* [31]. This is a method of combining several incremental learning algorithms into a single incremental learning algorithm that is more powerful and more robust than any of the component algorithms. The idea is simple. All the component learning algorithms are run in parallel on the same sequence of training examples. For each example, each algorithm makes a prediction and these predictions are combined by a weighted voting scheme to determine the overall prediction of the "master" algorithm. After receiving feedback on its prediction, the master algorithm adjusts the voting weights for each of the component algorithms, increasing the weights of those that made the correct prediction, and decreasing the weights of those that guessed wrong, in each case by a multiplicative factor. It can be shown that this method of combining learning algorithms produces a master algorithm with a worst case mistake bound that approaches the best worst case mistake bound of any of the component learning algorithms, and that the resulting algorithm is very robust with regard to mislabeled examples [31]. The weighted majority method can also be used in conjunction with the conversion mentioned above to design better PAC learning algorithms.

Both the PAC and total mistake bound models can be extended significantly by allowing learning algorithms to perform experiments or make queries to a teacher during learning [3]. The simplest type of query is a *membership query*, in which the learning algorithm *mistake*. The results can be summarized as follows. Let $\mathbf{C} = \{C_n\}_{n\geq 1}$ be a concept class and $d_n = VCdim(C_n)$ for all $n \geq 1$.

First, for any concept class $\mathbf{C}$ and any consistent algorithm for $\mathbf{C}$ using hypothesis space $\mathbf{C}$, the worst case probability of mistake on example t is at most $O((d_n/t)\mathbf{ln}(t/d_n))$, where $t > d_n$. Furthermore, there are particular consistent algorithms and concept classes where the worst case probability of mistake on example t is at least $\Omega((d_n/t)\mathbf{ln}(t/d_n))$, hence this is the best that can be said in general of arbitrary consistent algorithms.

Second, for any concept class $\mathbf{C}$ there exists a learning algorithm for $\mathbf{C}$ (not necessarily consistent or computationally efficient) with worst case probability of mistake on example t at most $d_n/(t-1)$. (An extra factor of 2 appears in the bound in [21]. This can be removed.) In addition, any learning algorithm for $\mathbf{C}$ must have worst case probability of mistake on example t at least $\Omega(d_n/t)$. Furthermore, there are particular concept classes $\mathbf{C}$, particular prior probability distributions on the concepts in these classes, and particular distributions on the instance spaces of these classes, such that the average case probability of mistake on example t is at least $\Omega(d_n/t)$ for any learning algorithm.

These results show two interesting things. First, certain learning algorithms perform better than arbitrary consistent learning algorithms in the worst case and average case, therefore, even in this restricted setting there is definitely more to learning than just finding any consistent hypothesis in an appropriately biased hypothesis space. Second, the worst case is not always much worse than the average case. Some recent experiments in learning perceptrons and multilayer perceptrons have shown that in many cases d_n/t is a rather good predictor of actual (i.e. average case) learning curves for backpropagation on synthetic random data [7,40]. However, it is still often an overestimate on natural data [38], and in other domains such as learning conjunctive concepts on a uniform distribution [39]. Here the distribution (and algorithm) specific aspects of the learning situation must also be taken into account. Thus, in general we concur that extensions of the PAC model are required to explain learning curves that occur in practice. However, no amount of experimentation or distribution specific theory can replace the security provided by a distribution independent bound.

The second criticism of the PAC model is that the assumptions of well-defined target concepts and noise-free training data are unrealistic in practice. This is certainly true. However, it should be pointed out that the computational hardness results for learning described above, having been established for the simple noise-free case, must also hold for the more general case. The PAC model has the advantage of allowing us to state these negative results simply and in their strongest form. Nevertheless, the positive learnability results have to be strengthened before they can be applicable in practice, and some extensions of the PAC model are needed for this purpose. Many have been proposed (see e.g. [6,24]).

Since the definitions of target concepts, random examples and hypothesis error in the PAC model are just simplified versions of standard definitions from statistical pattern recognition and decision theory, one reasonable thing to do is to go back to these well-established fields and use the more general definitions that they have developed. First, instead of using the probability of misclassification as the only measure of error, a general *loss function* can be defined that for every pair consisting of a guessed value and an actual value of the classification, gives a non-negative real number indicating a "cost" charged for that particular guess given that particular actual value. Then the error of a hypothesis can be replaced by the average loss of the hypothesis on a random example. If the loss is 1 if the guess is wrong and 0 if it is right (*discrete loss*), we get the PAC notion of error as a special case. However, using a more general loss function we can also choose to make false positives more expensive than false negatives or vice-versa, which can be useful. The use of a loss function also allows us to handle cases where there are more than two possible values of the classification. This includes the problem of learning real-valued functions, where we might choose to use $|guess - actual|$ or $(guess - actual)^2$ as loss functions.

Second, instead of assuming that the examples are generated by selecting a target concept and then generating random instances with labels agreeing with this target concept, we might assume that for each random instance, there is also some randomness in its label. Thus, each instance will have a particular probability of being drawn and, given that instance, each possible classification value will have a particular probability of occurring. This whole random process can be described as making independent random draws from a single joint probability distribution on the set of all possible labeled instances. Target concepts with attribute noise, classification noise, or both kinds of noise can be modeled in this way. The target concept, the noise, and the distribution on the instance space are all bundled into one joint probability measure on labeled examples. The goal of learning is then to find a hypothesis that minimizes the average loss when the examples are drawn at random according to this joint distribution.

The PAC model, disregarding computational complexity considerations, can be viewed as a special case

This improves on earlier bounds given in [11], but may still be a considerable overestimate. In terms of the cardinality of H_n, denoted $|H_n|$, it can be shown [43,33,12] that the sample complexity is at most

$$\frac{1}{\epsilon}\left(\mathbf{ln}|H_n| + \mathbf{ln}\frac{1}{\delta}\right).$$

For most hypothesis spaces on Boolean domains, the second bound gives the better bound. However, linear threshold functions are a notable exception, since the VC dimension of this class is linear in n, while the logarithm of its cardinality is quadratic in n [11]. Most hypothesis spaces on real-valued attributes are infinite, so only the first bound is applicable.

6 Criticisms of the PAC Model

The two criticisms most often leveled at the PAC model by AI researchers interested in empirical machine learning are

1. the worst-case emphasis in the model makes it unusable in practice [13,39] and
2. the notions of target concepts and noise-free training data are too restrictive in practice [1,9].

We take these in turn.

There are two aspects of the worst case nature of the PAC model that are at issue. One is the use of the worst case model to measure the computational complexity of the learning algorithm, the other is the definition of the sample complexity as the worst case number of random examples needed over all target concepts in the target class and all distributions on the instance space. We address only the latter issue.

As pointed out above, the worst case definition of sample complexity means that even if we could calculate the sample complexity of a given algorithm exactly, we would still expect it to overestimate the typical error of the hypothesis produced as a function of the training set size on any particular target concept and particular distribution on the instance space. This is compounded by the fact that we usually cannot calculate the sample complexity of a given algorithm exactly even when it is a relatively simple consistent algorithm. Instead we are forced to fall back on the upper bounds on the sample complexity that hold for any consistent algorithm, given in the previous section, which themselves may contain overblown constants.

The upshot of this is that the basic PAC theory is not good for predicting learning curves. Some variants of the PAC model come closer, however. One simple variant is to make it distribution specific, i.e. define and analyze the sample complexity of a learning algorithm for a specific distribution on the instance space, e.g. the uniform distribution on a Boolean space [8,39]. There are two potential problems with this. The first is finding distributions that are both analyzable and indicative of the distributions that arise in practice. The second is that the bounds obtained may be very sensitive to the particular distribution analyzed, and not be very reliable if the actual distribution is slightly different.

A more refined, Bayesian extension of the PAC model is explored in [13]. Using the Bayesian approach involves assuming a prior distribution over possible target concepts as well as training instances. Given these distributions, the average error of the hypothesis as a function of training sample size, and even as a function of the particular training sample, can be defined. Also, $1-\delta$ confidence intervals like those in the PAC model can be defined as well. Experiments with this model on small learning problems are encouraging, but further work needs to be done on sensitivity analysis, and on simplifying the calculations so that larger problems can be analysed. This work, and the other distribution specific learning work, provides an increasingly important counterpart to PAC theory.

Another variant of the PAC model designed to address these issues is the "probability of mistake" model explored in [21]. This is a worst case model that was designed specifically to help understand some of the issues in incremental learning. Instead of looking at sample complexity as defined above, the measure of performance here is the probability that the learning algorithm incorrectly guesses the label of the tth training example in a sequence of t random examples. Of course, the algorithm is allowed to update its hypothesis after each new training example is processed, so as t grows, we expect the probability of a mistake on example t to decrease. For a fixed target concept and a fixed distribution on the instance space, it is easy to see that the probability of a mistake on example t is the same as the average error of the hypothesis produced by the algorithm from $t-1$ random training examples. Hence, the probability of mistake on example t is exactly what is plotted on empirical learning curves that plot error versus sample size and average several runs of the learning algorithm for each sample size.

In [21], some comparisons are made between the worst case probability of mistake on the tth example (over all possible target concepts and distributions on the training examples) and the probability of mistake on the tth example when the target concept is selected at random according to a prior distribution on the target class and the examples are drawn at random from a certain fixed distribution (a Bayesian approach). The former we will call the *worst case probability of mistake* and the latter we will call the *average case probability of* the complexity theoretic assumption that $\mathbf{RP} \neq \mathbf{NP}$ [35].

1. Conjunctive concepts are properly PAC learnable [42], but the class of concepts in the form of the disjunction of two conjunctions is not properly PAC learnable [35], and neither is the class of existential conjunctive concepts on structural instance spaces with two objects [18].

2. Linear threshold concepts (perceptrons) are properly PAC learnable on both Boolean and real-valued instance spaces [11], but the class of concepts in the form of the conjunction of two linear threshold concepts is not properly PAC learnable [10]. The same holds for disjunctions and linear thresholds of linear thresholds (i.e. multilayer perceptrons with two hidden units). In addition, if the weights are restricted to 1 and 0 (but the threshold is arbitrary), then linear threshold concepts on Boolean instances spaces are not properly PAC learnable [35].

3. The classes of k-DNF, k-CNF, and k-decision lists are properly PAC learnable for each fixed k [41,37], but it is unknown whether the classes of all DNF functions, all CNF functions, or all decision trees are properly PAC learnable.

Most of the difficulties in proper PAC learning are due to the computational difficulty of finding a hypothesis in the particular form specified by the target class. For example, while Boolean threshold functions with 0-1 weights are not properly PAC learnable on Boolean instance spaces (unless $\mathbf{RP} = \mathbf{NP}$), they are PAC learnable by general Boolean threshold functions. Here we have a concrete case where enlarging the hypothesis space makes the computational problem of finding a good hypothesis easier. The class of all Boolean threshold functions is simply an easier space to search than the class of Boolean threshold functions with 0-1 weights. Similar extended hypothesis spaces can be found for the two classes mentioned in (1.) above that are not properly PAC learnable. Hence, it turns out that these classes are PAC learnable [35,18]. However, it is not known if any of the classes of DNF functions, CNF functions, decision trees, or multilayer perceptrons with two hidden units are PAC learnable.

It is a much stronger result to show that a concept class is not PAC learnable than it is to show that it is not properly PAC learnable, since the former result implies that the class is not PAC learnable by any reasonable hypothesis space. Nevertheless, such non-learnability results have been obtained for several important concept classes, including the class of Boolean formulae (Boolean expressions using "and" "or" and "not"), the general class of multilayer perceptrons with a multiple (but fixed) number of hidden layers, and the class of deterministic finite automata [27]. These results assume certain widely used cryptographic postulates in place of the (weaker) postulate that $\mathbf{RP} \neq \mathbf{NP}$.

5 Methods for Proving PAC Learnability; Formalization of Bias

All of the positive learnability results above are obtained by

1. showing that there is an efficient algorithm that finds a hypothesis in a particular hypothesis space that is consistent with a given sample of any concept in the target class and

2. that the sample complexity of any such algorithm is polynomial.

By *consistent* we mean that the hypothesis agrees with every example in the training sample. An algorithm that always finds such a hypothesis (when one exists) is called a *consistent algorithm.*

As the size of the hypothesis space increases, it may become easier to find a consistent hypothesis, but it will require more random training examples to insure that this hypothesis is accurate with high probability. In the limit, when any subset of the instance space is allowed as a hypothesis, it becomes trivial to find a consistent hypothesis, but a sample size proportional to the size of the entire instance space will be required to insure that it is accurate. Hence, there is a fundamental tradeoff between the computational complexity and the sample complexity of learning.

Restriction to particular hypothesis spaces of limited size is one form of *bias* that has been explored to facilitate learning [32]. In addition to the cardinality of the hypothesis space, a parameter known as the Vapnik-Chervonenkis (VC) dimension of the hypothesis space has been shown to be useful in quantifying the bias inherent in a restricted hypothesis space [19]. The VC dimension of a hypothesis space H, denoted $VCdim(H)$, is defined to be the maximum number d of instances that can be labeled as positive and negative examples in all 2^d possible ways, such that each labeling is consistent with some hypothesis in H [14,43]. Let $\mathbf{H} = \{H_n\}_{n\geq 1}$ be a hypothesis space and $\mathbf{C} = \{C_n\}_{n\geq 1}$ be a target class, where $C_n \subseteq H_n$ for $n \geq 1$. Then it can be shown [23] that any consistent algorithm for learning $\mathbf{C}$ by $\mathbf{H}$ will have sample complexity at most

$$\frac{1}{\epsilon(1-\sqrt{\epsilon})}\left(2VCdim(H_n)\mathbf{ln}\frac{6}{\epsilon} + \mathbf{ln}\frac{2}{\delta}\right).$$

to D, and a label that is "+" if that instance is in the target concept c (*positive example*), otherwise "−" (*negative example*). Thus, training and testing use the same distribution, and there is no "noise" in either phase. A learning algorithm is then a computational procedure that takes a sample of the target concept c, consisting of a sequence of independent random examples of c, and returns a hypothesis.

For each $n \geq 1$ let C_n be a set of target concepts over the instance space $\{0,1\}^n$, and let $\mathbf{C} = \{C_n\}_{n \geq 1}$. Let H_n, for $n \geq 1$, and $\mathbf{H}$ be defined similarly. We can define PAC learnability as follows: The concept class $\mathbf{C}$ is PAC learnable by the hypothesis space $\mathbf{H}$ if there exists a polynomial time learning algorithm A and a polynomial $p(\cdot,\cdot,\cdot)$ such that for all $n \geq 1$, all target concepts $c \in C_n$, all probability distributions D on the instance space $\{0,1\}^n$, and all ϵ and δ, where $0 < \epsilon, \delta < 1$, if the algorithm A is given at least $p(n, 1/\epsilon, 1/\delta)$ independent random examples of c drawn according to D, then with probability at least $1-\delta$, A returns a hypothesis $h \in H_n$ with $error(h) \leq \epsilon$. The smallest such polynomial p is called the *sample complexity* of the learning algorithm A.

The intent of this definition is that the learning algorithm must process the examples in polynomial time, i.e. be computationally efficient, and must be able to produce a good approximation to the target concept with high probability using only a reasonable number of random training examples. The model is worst case in that it requires that the number of training examples needed be bounded by a single fixed polynomial for all target concepts in $\mathbf{C}$ and all distributions D in the instance space. It follows that if we fix the number of variables n in the instance space and the confidence parameter δ, and then invert the sample complexity function to plot the error ϵ as a function of training sample size, we do not get what is usually thought of as a learning curve for A (for this fixed confidence), but rather the upper envelope of all learning curves for A (for this fixed confidence), obtained by varying the target concept and distribution on the instance space. Needless to say, this is not a curve that can be observed experimentally. What is usually plotted experimentally is the error versus the training sample size for particular target concepts on instances chosen randomly according to a single fixed distribution on the instance space. Such a curve will lie below the curve obtained by inverting the sample complexity. We will return to this point later.

Another thing to notice about this definition is that target concepts in a concept class $\mathbf{C}$ may be learned by hypotheses in a different class $\mathbf{H}$. This gives us some flexibility. Two cases are of interest. The first is that $\mathbf{C} = \mathbf{H}$, i.e. the target class and hypothesis space are the same. In this case we say that $\mathbf{C}$ is *properly* PAC learnable. Imposing the requirement that the hypothesis be from the class $\mathbf{C}$ may be necessary, e.g. if it is to be included in a specific knowledge base with a specific inference engine. However, as we will see, it can also make learning more difficult. The other case is when we don't care at all about the hypothesis space $\mathbf{H}$, so long as the hypotheses in $\mathbf{H}$ can be evaluated efficiently. This occurs when our only goal is accurate and computationally efficient prediction of future examples. Being able to freely choose the hypothesis space may make learning easier. If $\mathbf{C}$ is a concept class and there exists some hypothesis space $\mathbf{H}$ such that hypotheses in $\mathbf{H}$ can be evaluated on given instances in polynomial time and such that $\mathbf{C}$ is PAC learnable by $\mathbf{H}$, then we will say simply that $\mathbf{C}$ is *PAC learnable*.

There are many variants of the basic definition of PAC learnability. One important variant defines a notion of syntactic complexity of target concepts and, for each $n \geq 1$, further classifies each concept in C_n by its syntactic complexity. Usually the syntactic complexity of a concept c is taken to be the length of (number of symbols in) the shortest description of c in a fixed concept description language. In this variant of PAC learnability, the number of training examples is also allowed to grow polynomially in the syntactic complexity of the target concept. This variant is used whenever the concept class is specified by a concept description language that can represent any boolean function, for example, when discussing the learnability of DNF (Disjunctive Normal Form) formulae or decision trees. Other variants of the model let the algorithm request examples, use separate distributions for drawing positive and negative examples, or use randomized (i.e. coin flipping) algorithms [25]. It can be shown that these latter variants are equivalent to the model described here, in that, modulo some minor technicalities, the concept classes that are PAC learnable in one model are also PAC learnable in the other [20]. Finally, the model can easily be extended to non-Boolean attribute-based instance spaces [19] and instance spaces for structural domains such as the blocks world [18]. Instances can also be defined as strings over a finite alphabet so that the learnability of finite automata, context-free grammars, etc. can be investigated [34].

4 Outline of Results for the Basic PAC Model

A number of fairly sharp results have been found for the notion of proper PAC learnability. The following summarizes some of these results. For precise definitions of the concept classes involved, the reader is referred to the literature cited. The negative results are based on

Probably Approximately Correct Learning

David Haussler*
haussler@saturn.ucsc.edu
Baskin Center for Computer Engineering and Information Sciences
University of California, Santa Cruz, CA 95064

1 Abstract

This paper surveys some recent theoretical results on the efficiency of machine learning algorithms. The main tool described is the notion of Probably Approximately Correct (PAC) learning, introduced by Valiant. We define this learning model and then look at some of the results obtained in it. We then consider some criticisms of the PAC model and the extensions proposed to address these criticisms. Finally, we look briefly at other models recently proposed in computational learning theory.

2 Introduction

It's a dangerous thing to try to formalize an enterprise as complex and varied as machine learning so that it can be subjected to rigorous mathematical analysis. To be tractable, a formal model must be simple. Thus, inevitably, most people will feel that important aspects of the activity have been left out of the theory. Of course, they will be right. Therefore, it is not advisable to present a theory of machine learning as having reduced the entire field to its bare essentials. All that can be hoped for is that some aspects of the phenomenon are brought more clearly into focus using the tools of mathematical analysis, and that perhaps a few new insights are gained. It is in this light that we wish to discuss the results obtained in the last few years in what is now called PAC (Probably Approximately Correct) learning theory [3].

Valiant introduced this theory in 1984 [42] to get computer scientists who study the computational efficiency of algorithms to look at learning algorithms. By taking some simplified notions from statistical pattern recognition and decision theory, and combining them with approaches from computational complexity theory, he came up with a notion of learning problems that are feasible, in the sense that there is a polynomial time algorithm that "solves" them, in analogy with the class **P** of feasible problems in standard complexity theory. Valiant was successful in his efforts. Since 1984 many theoretical computer scientists and AI researchers have either obtained results in this theory, or complained about it and proposed modified theories, or both.

The field of research that includes the PAC theory and its many relatives has been called computational learning theory. It is far from being a monolithic mathematical edifice that sits at the base of machine learning; it's unclear whether such a theory is even possible or desirable. We argue, however, that insights have been gained from the varied work in computational learning theory. The purpose of this short monograph is to survey some of this work and reveal those insights.

3 Definition of PAC Learning

The intent of the PAC model is that successful learning of an unknown target concept should entail obtaining, with high probability, a hypothesis that is a good approximation of it. Hence the name Probably Approximately Correct. In the basic model, the instance space is assumed to be $\{0,1\}^n$, the set of all possible assignments to n Boolean variables (or *attributes*) and concepts and hypotheses are subsets of $\{0,1\}^n$. The notion of approximation is defined by assuming that there is some probability distribution D defined on the instance space $\{0,1\}^n$, giving the probability of each instance. We then let the *error* of a hypothesis h w.r.t. a fixed target concept c, denoted $error(h)$ when c is clear from the context, be defined by

$$error(h) = \sum_{x \in h \Delta c} D(x),$$

where Δ denotes the symmetric difference. Thus, $error(h)$ is the probability that h and c will disagree on an instance drawn randomly according to D. The hypothesis h is a good approximation of the target concept c if $error(h)$ is small.

How does one obtain a good hypothesis? In the simplest case one does this by looking at independent random examples of the target concept c, each example consisting of an instance selected randomly according

*Supported by ONR grant N00014-86-K-0454