An improved boosting algorithm and its implications on learning complexity

Yoav Freund
Computer and Information Sciences
University of California
Santa Cruz, CA 95064
yoav@cis.ucsc.edu

Abstract

In this work we present some improvements and extensions to previous work on boosting weak learners [Sch90, Fre90]. Our main result is an improvement of the boosting-by-majority algorithm. One implication of the performance of this algorithm is that if a concept class can be learned in the PAC model to within some fixed error smaller than 1/2, then it can be learned to within an arbitrarily small error $\epsilon > 0$ with time complexity $O((1/\epsilon)(\log 1/\epsilon)^2 \log\log 1/\epsilon)$ (fixing the sample space and concept class and the required reliability). We show that the majority rule is the optimal rule for combining general weak learners. We also extend the boosting algorithm to concept classes that give multi-valued labels and real-valued labels.

1 INTRODUCTION

In [Val84], Valiant introduced the polynomial-time, distribution-free and noise-free learning model commonly called the PAC (Probably Approximately Correct) learning model. The learning task in this model is to approximate a subset c (a "concept") of a domain X taken from a known class of subsets $\mathbf{C} \subset 2^X$ (the concept class). The learner is given two small real numbers ϵ and δ and is required to produce a hypothesis subset $h \subset X$ such that with probability larger than $1-\delta$ the accuracy of h is at least $1-\epsilon$, i.e. the probability of the symmetric difference between c and h is smaller than ϵ. In order to do that the learner receives examples randomly drawn from X, where each example is labeled '+' if it is in c and '-' otherwise. The algorithm should work equally well for any distribution on X. In order to define the complexity of learning algorithms, we consider concept classes of increasing complexity. We assume some standard description languages for examples and for concepts and denote by n the length of the description of an example and by s the length of the description of a concept. We define $\mathbf{C}(n,s)$ to be the class of concepts that can be described in length s whose domain is those examples in X that can be described in length n. For example, we can define $DNF(n,s)$ to be the boolean formulae over n binary variables that can be written in DNF using s characters (For complete details see [HKLW91]). The definition of a (strong) PAC learning algorithm requires that its running time is polynomial in $n, s, 1/\epsilon$ and $1/\delta$ and is independent of the distribution on X. We shall compare our results to some results concerning PAC learning *without computational constraints*. In this model only the *sample size* is required to be polynomial and the run time is unconstrained.

A seemingly less demanding learning model is the "weak"-PAC learning model [KV89], in which the requirements on ϵ and δ are much weaker.[1] The weak requirements are that the error ϵ be just slightly smaller than 1/2 and that the reliability $1-\delta$ be just slightly larger than zero. More precisely, the requirement of

[1]Kearns and Valiant weaken only the requirement on ϵ, for completeness, we also weaken the requirement on δ.

weak learning is that there exist some polynomials $p(n,s)$ and $q(n,s)$ such that the algorithm produces a hypotheses with error smaller than $\frac{1}{2}-\frac{1}{p(n,s)}$ with probability larger than $\frac{1}{q(n,s)}$. In [Sch90] Schapire proved the equivalence between weak and strong PAC-learning by showing that any weak learning algorithm can be transformed into a strong learning algorithm. His result is based on a so-called "boosting" algorithm. This algorithm combines several hypotheses generated by a weak-learner, which we call weak hypotheses, into a single more accurate hypothesis. The performance of the boosting algorithm provides general upper bounds on the space and time complexity of PAC learning, and has many other implications for data compression, on-line learning and the approximation of hard functions [Sch90]. The algorithm is also interesting from a practical point of view as it provides a general method for improving the performance of learning algorithms. In [Fre90] the author presented a different boosting algorithm, called "boosting-by-majority." This algorithm employs a single majority vote among the weak hypotheses instead of the circuit of three-way majority gates used in the original algorithm. The number of weak hypotheses combined by the final hypothesis was reduced, which improved results in data compression obtained through the application of boosting. Two variants of the boosting-by-majority algorithm were presented. Each gave a different trade-off of space and time of PAC learning, but neither of them provided an overall improvement over the algorithm in [Sch90]. In this paper we provide an improved version of the algorithm that is an overall improvement over the algorithms in [Sch90, Fre90] and is the best boosting algorithm known so far in terms of sample complexity and length of hypothesis.

Boosting algorithms control the hypotheses generated by the weak learning algorithm by changing the distribution on the sample space from which the weak learning algorithm draws its examples. This is done by discarding some of the examples in the original sample and accepting others in a process called "filtering." Intuitively, the role of filtering is to force the weak learning algorithm to concentrate its effort on those parts of the sample space on which previous hypotheses have performed poorly. The problem in the algorithm presented in [Fre90] is that the filtering rule might accept only $\tilde{\Omega}(\epsilon^2)$ (ignoring log factors) of the examples presented to it. This causes the sample and time complexity to be $\tilde{O}(1/\epsilon^2)$ which is inferior to the sample and time complexity of Schapire's algorithm. The key to reducing the sample size is to observe that in the cases in which less than $\tilde{\Omega}(\epsilon)$ of the examples are accepted, the sensitivity of the final error to the quality of the current weak hypothesis is so low that a random coin flip can be used instead of the weak hypothesis.

The algorithm presented here produces hypotheses that combine the weak hypotheses by circuits of depth 2 in which the input gates are threshold gates and the output gate is a randomized threshold gate where the threshold value is normally distributed. This output gate is very similar to the randomized sigmoidal-response gates used in Boltzmann machines [AHS85], and the resulting logic is very much like a two layer perceptron.

The most striking implication of Schapire's algorithm is that if a concept class is PAC learnable, then it is PAC learnable in time $(1/\epsilon)$ polylog$(1/\epsilon)$ using polylog$(1/\epsilon)$ space [Sch90][Theorem 4]. Specifically, fixing n, s and δ, the sample size and time complexity bounds are $O((1/\epsilon)(\log 1/\epsilon)^{5.17}(\log\log 1/\epsilon))$, and the bounds on the space complexity and the output hypothesis size are $O((\log 1/\epsilon)^{1.58})$ [Sch92]. Our new algorithm improves on these upper bounds. In particular we show that

If $\mathbf{C}$ *is a PAC learnable concept class, parameterized by* n *and* s *in the standard way [HKLW91], then there exists a PAC learning algorithm for* $\mathbf{C}$ *that learns with accuracy* ϵ *and reliability* δ *and:*

- *requires a sample of size* $(1/\epsilon)(\log 1/\epsilon)^2(\log\log 1/\epsilon + \log 1/\delta)p_1(n,s)$,
- *halts in time* $(1/\epsilon)(\log 1/\epsilon)^2(\log\log 1/\epsilon + \log 1/\delta)p_2(n,s)$,
- *uses space* $(\log 1/\epsilon)(\log\log 1/\epsilon + \log 1/\delta)p_3(n,s)$, *and*
- *outputs hypotheses of size* $(\log 1/\epsilon)p_4(n,s)$ *evaluatable in time* $(\log 1/\epsilon)p_5(n,s)$

for some polynomials p_1, p_2, p_3, p_4 *and* p_5.

Matching lower and upper bounds exist for the dependence of the sample size on the accuracy ϵ for PAC learning *without computational constraints*. A general lower bound of $\Omega(1/\epsilon)$ is given in [BEHW86] for learning any "non-trivial" concept class. This is matched by a general upper-bound, given in [HLW88][Theorem 5.1], which says that, ignoring dependence on δ, any concept class that can be learned using a sample of size polynomial in $1/\epsilon$ can be learned using a sample of size $O(1/\epsilon)$. Because of this, proving a higher lower bound for *polynomial time* PAC learning algorithms will imply that $RP \neq NP$. An open question that might be easier is to find an upper bound closer to $O(1/\epsilon)$ on the sample size for polynomial time PAC learning.

Similarly, we improve on Theorem 8 in [Sch90] which relates PAC learning to efficient on-line learning. The sample complexity of our boosting algorithm implies that a PAC learning algorithm can be translated into a polynomial-time on-line learning algorithm for which the probability of a mistake on the mth trial is $O(m^{-1}(\log m)^3)$.

Another improvement on Schapire's boosting algorithm is in the dependency of the sample size on the accuracy of the weak learning algorithm. The requirement on the weak learning algorithm is that its error be smaller than $1/2-\gamma$, where $\gamma = \frac{1}{p(n,s)}$ and p is a polynomial. The dependence of the sample complexity on γ quantifies the difficulty of transforming a very small advantage over random guessing into a significant advantage. The dependence of the sample complexity on γ in Schapire's algorithm is $\tilde{O}(1/\gamma^{13.25})$ [Sch92] while in our algorithm it is $\tilde{O}(1/\gamma^2)$. This improvement is the most significant improvement over Schapire's algorithm from the practical point of view. In Section (5) we give an argument that suggests that this dependency cannot be further improved.

Up to this point we have discussed the standard PAC model, in which the concepts are subsets of the domain X, or, using a different notation, indicator functions whose range is $\{0,1\}$. The algorithm can be extended to more general cases. In particular we show how to extend the results to functions whose range is $\{1,2,\ldots,k\}$ and to functions whose range is the real line. For k-valued concepts the best rule for combining the weak hypotheses is again the majority rule and the boosting algorithm is almost identical. When learning real-valued functions, the boosting algorithm can be used for translating a learning algorithm that makes a small average error (such as mean-squared-error, absolute-error etc.) for any distribution on the domain, to an algorithm that makes a small error on almost all the domain. The method is based on the observation that if the average error is smaller than d then the probability that the the error is smaller than $\frac{d}{1/2+\gamma}$ is larger than $1/2+\gamma$. Using boosting we can make this probability arbitrarily high. Here, instead of the majority rule that is used for combining binary hypotheses, the median of the real functions produced by the weak learner is used.

2 IMPROVED BOOSTING ALGORITHM

Our boosting algorithm receives as input a weak learner, denoted **weakLearner**, and the parameters γ, ϵ and δ defined above. It employs two variants of the "boosting-by-majority" algorithm [Fre90]. The original variant, from [Fre90] and a new variant, presented here. In the following discussion we ignore the dependence on the reliability parameter δ which can be easily analyzed and has a minor effect on the complexity.

The "boosting-by-majority" algorithm uses the same ideas as Schapire's algorithm [Sch90] but is serial rather than recursive. It can be separated into two parts. The first is the algorithm denoted **main** which computes the number k of weak hypotheses required for the given accuracy, $k = (2\gamma^2)^{-1}(\log 1/\epsilon)$. It then calls **weakLearner** k times and combines the weak hypotheses that are generated by a k-vote majority. We shall refer to each call of **weakLearner** as a *stage* and index stages by i, $0 \le i \le k-1$. The second part is the filtering algorithm, called **FiltEx**. This subroutine is called by **weakLearner** whenever it needs an example, it replaces the random example oracle **Ex** that **weakLearner** would regularly use as its source of examples. Whenever **FiltEx** is called to get a new example it makes several calls to **Ex** and selects one example to pass back to **weakLearner**. By doing this, **FiltEx** is presenting **weakLearner** with distributions different from the one generated by **Ex**.

The subroutine **FiltEx** decides which examples to accept according to the following stochastic rule. For each example it computes the labels according to all the previously generated weak hypotheses. It then compares these answers to the label given by **Ex** and counts the number r of weak hypotheses that are correct on this example. It then computes the desired probability α of accepting this example as a function of r, i, and k [2]

$$\alpha_r^i = \begin{cases} \text{if } r > \lfloor\frac{k}{2}\rfloor \text{ then } 0 \\ \text{if } i - \lceil\frac{k}{2}\rceil \le r \le \lfloor\frac{k}{2}\rfloor \text{ then} \\ \binom{k-i-1}{\lfloor\frac{k}{2}\rfloor - r}\left(\frac{1}{2}+\gamma\right)^{\lfloor\frac{k}{2}\rfloor - r}\left(\frac{1}{2}-\gamma\right)^{\lceil\frac{k}{2}\rceil - i - 1 + r} \\ \text{if } r < i - \lceil\frac{k}{2}\rceil \text{ then } 0 \end{cases} \qquad (1)$$

This desired probability of accepting the example is achieved by flipping a random coin that has probability α_r^i of "head" and accepting the example if the coin flip results in "head." We refer to α_r^i as the "filtering factor".

The problem with this scheme is that in some cases the probability that **FiltEx** accepts an example might be very low, which results in unacceptably high sample complexity. However, if such a situation occurs, then the hypotheses that have been created so far are, loosely speaking, much better than required. Using this, various methods for bounding the acceptance probability from below can be devised that guarantee low sample complexity. [3] Two solutions to the problem are combined in the new algorithm. We denote by t_i the probability that **FiltEx** accepts a random example generated by **Ex** during the i-th stage.

- The first solution was described in [Fre90] and is based on the observation that if $t_i < c_1\epsilon^2$, for some constant c_1, then the majority over the hypotheses that have been generated before the ith stage is already an ϵ-accurate hypothesis. Thus **FiltEx** aborts **weakLearner** and **main** if t_i is too low, causing **main** to return the partial majority as the final hypothesis. The advantage of this solution is that the dependence of its time and space complexity on γ is $\tilde{\Theta}(1/\gamma^2)$. However, the dependence of the sample size on ϵ is $\tilde{O}(1/\epsilon^2)$.
- The second solution, presented here, is based on the observation that if $t_i < c_2\epsilon$, then the accuracy

[2]This formula can be interpreted as the probability of $\lfloor\frac{k}{2}\rfloor - r$ successes in $k-i-1$ Bernoulli trials, each with probability $\frac{1}{2}+\gamma$ to succeed.

[3]Another solution to this problem is given in [Fre90] and achieves the same sample and time complexity of the new algorithm. However it does so at the cost of having to store all the examples in memory, which requires $\tilde{O}(1/\epsilon)$ space.

of the final hypothesis is insensitive to the accuracy of the i-th weak hypothesis. Based on this fact, `FiltEx` aborts `weakLearner` if t_i is too low. In this case `main` does continue to the next stage and uses, instead of the weak-hypothesis that was not generated, a trivial hypothesis that is just a flip of a fair coin. The advantage of this algorithm is that the dependence of the sample size on ϵ is $\tilde{\Theta}(1/\epsilon)$. However, the dependence of the time and space complexity on γ is $\tilde{O}(1/\gamma^3)$.

Note that the final hypothesis is a majority gate for which some of the inputs are random coin flips. An alternative way of viewing this gate is as a stochastic gate over the actual weak hypothesis whose output is a random variable. The probability that the output is 1 approaches 0 as the number of inputs that are one decreases and approaches 1 as this number increases. This type of gate is very similar to the sigmoidal response gates commonly used in neural network models such as Boltzmann machines [AHS85].

The boosting algorithm uses both methods in order to combine their advantages. The first method is used for reducing the error from $1/2 - \gamma$ to some fixed error, say 1/4 and the second solution is used to reduce the error from 1/4 to ϵ. This is done by treating the algorithm that uses the first method as a weak learner and boosting this weak learner by an algorithm that uses the second method. Thus the first stage does not depend on ϵ and the second does not depend on γ, resulting in a sample complexity of $\tilde{O}(1/\epsilon\gamma^2)$.

3 PERFORMANCE OF THE ALGORITHM

To further improve the performance of our boosting algorithm, we combine it with the algorithm presented in [HKLW91] for improving the *reliability* of a learning algorithm. By so doing we get the following upper bound on the resources needed for achieving arbitrary accuracy and reliability using an algorithm with some known accuracy and reliability.

Theorem 1 *Assume there exists a learning algorithm that can learn the concept class* $\mathbf{C}$ *with error* $0 < \epsilon_0 < 1/2$ *and reliability* $0 < \delta_0 < 1$. *Assume the weak algorithm requires time* t, m *examples, and* v *space, and that it outputs hypotheses of size* v' *that can be evaluated in time* t'. *Here* $\mathbf{C}, \epsilon_0, \delta_0, t, m, n, v, v', t'$ *are assumed to be functions of* n *and* s *defined in the standard way [HKLW91]. Define* $\gamma = 1/2 - \epsilon_0$. *Then* $\mathbf{C}$ *can be learned to within an arbitrary accuracy and reliability* $\epsilon, \delta > 0$ *by an algorithm that :*

- *requires a sample of size* $O(m\frac{(\log 1/\epsilon)^2(\log(1/\delta)+\log(1/\gamma)+\log\log(1/\epsilon))}{\epsilon\gamma^2\log(1/\delta_0)})$,
- *halts in time* $O(t\frac{(\log 1/\epsilon)^2(\log(1/\delta)+\log(1/\gamma)+\log\log(1/\epsilon))}{\epsilon\gamma^2\log(1/\delta_0)})$,
- *uses space* $O(v\frac{\log(1/\epsilon)(\log(1/\delta)+\log(1/\gamma)+\log\log(1/\epsilon))}{\gamma^2\log(1/\delta_0)})$, *and*
- *outputs hypotheses of size* $O(v'\frac{\log(1/\epsilon)}{\gamma^2})$ *that are evaluatable in time* $O(t'\frac{\log(1/\epsilon)}{\gamma^2})$

Note that the theorem implies PAC learning if the parameters defining the weak learner are all polynomial in n and s. However, it applies even if some of the resource requirements of the weak learning algorithm are not polynomial, in which case the strong learning algorithm, claimed by the theorem, will have corresponding non-polynomial requirements.

4 SKETCH OF THE PROOF

The proof of the correctness of the boosting algorithm is based on the following potential function, defined over the same parameters as those of the filtering factor α defined in (1)

$$\beta_r^i = \begin{cases} \text{if } r > \lfloor \frac{k}{2} \rfloor \text{ then } 0 \\ \text{if } i - \lceil \frac{k}{2} \rceil \leq r \leq \lfloor \frac{k}{2} \rfloor \text{ then} \\ \sum_{j=0}^{\lfloor \frac{k}{2} \rfloor - r} \binom{k-i}{j} (\frac{1}{2}+\gamma)^j (\frac{1}{2}-\gamma)^{k-i-j} \\ \text{if } r < i - \lceil \frac{k}{2} \rceil \text{ then } 1 \end{cases} \tag{2}$$

The potential function can be given the following interpretation. Suppose the errors of the weak learner are independent, i.e. the weak hypotheses are simply the correct concept with an addition of independent random noise. We call such a learner the "independent weak learner". Then β_r^i is the probability that an example will be labeled incorrectly by the final k-majority rule, given that r out of the first i hypotheses label it correctly. Thus, under these conditions, β_0^0 is the probability that *any* example is labeled incorrectly by a majority of k weak hypotheses created by the independent weak learner. It is easy to see that β_0^0 decreases very rapidly when k is increased, which means that a small majority suffices . It is also easy to realize that the expected value of β_r^i over the sample space and the runs of the algorithm is equal for all i.

Of course, if the mistakes of the weak learner were independent, then boosting would have been trivial; in general, there might be dependencies between the mistakes of the weak hypotheses. However, we proved in [Fre90] that for *any* weak learner the expected value of β_r^i (as a function of the random variable r) is non-increasing with i if the examples are filtered according to α_r^i as was described above. Thus the error of the majority over k weak hypotheses is the same as the error of a majority over k weak hypotheses created by the independent weak learner.

As said earlier, the new algorithm combines two approaches for solving the problem of an unacceptably low example acceptance rate. The first approach was described and proved in [Fre90]. The correctness of the second approach is proven below.

In the following discussion we fix a run of the boosting algorithm, i.e. we consider a particular sequence of weak hypotheses (or random coin flips). Thus all references to expected values in this section refer to expectations over the sample space only (essentially, over r) and not over different runs of the algorithm. We ignore the issue of reliability and assume that all weak hypotheses have less than $1/2-\gamma$ error and that the probability of accepting an example at each stage is known.

The main idea is that if the probability of accepting an example during some stage is low and instead of using a correct weak hypotheses `main` uses a random coin flip at this stage, then the expected value of the potential function β increases by only a little bit. This proof combines with the previous proof in [Fre90], that shows that if a correct weak hypotheses is used, the expected value of β does not increase at all. We first prove that if the probability of accepting an example at stage i is t_i then the increase in the expected value of β_r^i under these conditions is exactly γt_i. Then we show how combining this fact with the previous facts yields the claimed performance.

We define some notation needed for the proof. Let X be the sample space over which a probability distribution P is defined. Define $\mathcal{S}_i = \{S_0^i, S_1^i, \ldots, S_i^i\}$ to be a partition of X into $i+1$ parts where S_r^i consists of that part of the sample space that is labeled correctly by r out of the first i hypotheses. Define the following quantities related to the partitions $\mathcal{S}_i$:

$M_r^i = S_r^i \cap S_{r+1}^{i+1}$	the subset of S_r^i that is correctly labeled by the $i+1$st hypothesis
$q_r^i = P(S_r^i)$	
$x_r^i = \frac{P(M_r^i)}{P(S_r^i)}$	the probability of an example being correctly labeled given that it that it is in S_r^i

Note that $S_0^0 = X$ and thus $q_0^0 = 1$. Finally, denote by t_i the expected value of α_r^i w.r.t. the simulated distribution used in stage i, i.e., the probability of acceptance of a random example in stage i

$$t_i = \sum_{r=0}^{i} q_r^i \alpha_r^i$$

In the proof, we shall use the following relations that can easily be checked.

$$\alpha_r^i = \beta_r^{i+1} - \beta_{r+1}^{i+1} \qquad (3)$$

$$\beta_r^i = (\frac{1}{2}-\gamma)\beta_r^{i+1} + (\frac{1}{2}+\gamma)\beta_{r+1}^{i+1}. \qquad (4)$$

Using the new notation we now prove the following Lemma

Lemma 2 *If we use a random coin flip as the hypothesis at stage i of the boosting process we have the relationship*

$$\sum_{r=0}^{i} q_r^i \beta_r^i + \gamma t_i = \sum_{r=0}^{i+1} q_r^{i+1}\beta_r^{i+1}$$

Proof:

At each stage i we have the following formula for the transition to the next stage:

$$\begin{aligned} q_r^{i+1} &= q_{r-1}^i x_{r-1}^i + q_r^i(1-x_r^i) \quad \text{for} \quad 1 \le r \le i \quad (5)\\ q_0^{i+1} &= q_0^i(1-x_0^i) \quad \text{for} \quad r=0 \\ q_{i+1}^{i+1} &= q_i^i x_i^i \quad \text{for} \quad r = i+1. \end{aligned}$$

Using this, we can get a formula that relates the sum $\sum_{r=0}^{i} q_r^i \beta_r^i$ for consecutive stages

$$\sum_{r=0}^{i+1} q_r^{i+1}\beta_r^{i+1} =$$
$$q_0^i(1-x_0^i)\beta_0^{i+1} + q_0^i(1-x_0^i)\beta_0^{i+1} +$$
$$\sum_{r=1}^{i}[q_{r-1}^i x_{r-1}^i + q_r^i(1-x_r^i)]\beta_r^{i+1} +$$
$$q_i^i x_i^i \beta_{i+1}^{i+1}.$$

Rearranging the sum gives

$$\sum_{r=0}^{i+1} q_r^{i+1}\beta_r^{i+1} = \sum_{r=0}^{i} q_r^i[(1-x_r^i)\beta_r^{i+1} + x_r^i\beta_{r+1}^{i+1}]. \qquad (6)$$

In the stages where a random coin flip is used as a hypothesis $x_r^i = 1/2$ for all r. Substituting $1/2$ for x_r^i we get

$$\sum_{r=0}^{i+1} q_r^{i+1}\beta_r^{i+1} = \sum_{r=0}^{i} q_r^i[\frac{1}{2}\beta_r^{i+1} + \frac{1}{2}\beta_{r+1}^{i+1}]$$

which can also be written as

$$\sum_{r=0}^{i+1} q_r^{i+1}\beta_r^{i+1} =$$
$$\sum_{r=0}^{i} q_r^i[(\frac{1}{2}-\gamma)\beta_r^{i+1} + (\frac{1}{2}+\gamma)\beta_{r+1}^{i+1}]$$
$$+\gamma\sum_{r=0}^{i} q_r^i(\beta_r^{i+1} - \beta_{r+1}^{i+1}).$$

However, using Equation (4) for the first term and Equation (3) for the second term we get

$$\sum_{r=0}^{i+1} q_r^{i+1}\beta_r^{i+1} = \sum_{r=0}^{i} q_r^i\beta_r^i + \gamma\sum_{r=0}^{i} q_r^i\alpha_r^i$$

As the second sum is t_i we get the result. ▌

We use the lemma to bound the sample complexity in the following way. If, during stage i, the subroutine `FiltEx` detects, using a reliable decision rule, that

$t_i < \frac{(1-\epsilon)\epsilon}{k\gamma}$ it aborts `weakLearner` and causes `main` to use a random coin flip. In this case the average potential function increases by $\gamma t_i = \frac{(1-\epsilon)\epsilon}{k}$. Thus the total increase in the average potential over all stages is at most $(1-\epsilon)\epsilon$. On the other hand, by setting k twice as large as is required to insure $\beta_0^0 < \epsilon$ we insure that $\beta_0^0 < \epsilon^2$. Thus the average potential at the end of the process is at most $(1-\epsilon)\epsilon + \epsilon^2 = \epsilon$. From the definition of the potential function it immediately follows that the error of the final majority over all k hypotheses (both actual hypotheses and random coin flips) is smaller than ϵ.

Substituting k into the bound on t_i we get that the number of examples from `Ex` used at each stage is $O(\frac{1}{\gamma\epsilon}\log 1/\epsilon)$, thus the total number of examples used by the boosting algorithm is $O(\frac{1}{\gamma^3\epsilon}(\log 1/\epsilon)^2)$. The additional factor of $\log\log 1/\epsilon$ in the sample complexity bound given in Theorem (1) comes in from taking into account the required reliability δ.

5 THE ALGORITHM IS ALMOST OPTIMAL

Theorem (1) describes the resources required by our boosting algorithm. We claim that further improvement in the boosting algorithm is possible only by logarithmic factors. We have already discussed the lower bound of $\Omega(1/\epsilon)$ on the sample complexity. We further claim that the γ^{-2} factor in the resources required by the boosting algorithm is also unavoidable, and that the majority rule is the optimal way for combining weak hypotheses. In order to formalize these claims we have to define a separation between the boosting algorithm and the weak learning algorithm. Assume the examples are given as a sequence of pairs $\langle(x_1,y_1),(x_2,y_2),\ldots(x_n,y_n)\rangle$ $x_i \in X$, $y_i \in \{0,1\}$. The boosting algorithm is a PAC learning algorithm that has direct access only to the y_i part of each example. However, in addition to that, it has access to a learning oracle `weakLearner` and to a labeling oracle `label`. The input to `weakLearner` is a set of indices $1 \le i_1 \le i_2 \le \ldots \le i_k \le N$. `weakLearner` is a learning algorithm and uses as examples the subset $\langle(x_{i_1},y_{i_1}),(x_{i_2},y_{i_2}),\ldots,(x_{i_k},y_{i_k})\rangle$ of the sample. It is required to generate a hypothesis $h: X \to \{0,1\}$ that has error $\epsilon_0 = 1/2-\gamma$ and has reliability $1-\delta$ if the examples it uses as input are independently drawn from some distribution. The booster can use the hypotheses returned by `weakLearner` by calling the second oracle, `label`, to compute the label given by the hypothesis to any example in the sample, i.e. `label`$(h,i) = h(x_i)$.

Restricting a learning algorithm in such a way is natural in the context of hierarchical learning models such as weighted-majority [LW89] and layered neural networks. Some of the analysis of the weighted majority algorithm is concerned with efficiently searching for a good learning algorithm in a pool of algorithms. In this case the learning algorithm has access only to the outputs of the algorithms in the pool and not to the original input. Similarly, in a layered neural network model, units in the deeper hidden layers can receive input only from the layer below them and have no direct access to the input of the network. The process of filtering or sub-sampling can be interpreted, in this context, as a feedback mechanism by which a learning unit higher in the hierarchy directs lower level inputs to concentrate on those examples which will contribute most to the performance of the network as a whole.

Assuming these restrictions, it is easy to see that the $\Omega(\gamma^{-2})$ dependency is necessary. Assume `weakLearner` is the independent weak learner described earlier, i.e.

$$h_i(x) = \begin{cases} c(x) & \text{with probability } 1/2+\gamma \\ 1-c(x) & \text{with probability } 1/2-\gamma \end{cases}$$

Assume the errors of the different hypotheses are independent and that $Pr(c(x)=0) = Pr(c(x)=1) = 1/2$.[4] Because of the symmetry in the definitions it is easy to show that in this case the optimal way of combining the outputs of the hypotheses to get the most accurate prediction of $c(x)$ is to take the majority function over all the hypotheses. In this case the number of hypotheses required for achieving accuracy of ϵ is $\frac{1}{2}\gamma^{-2}\log 1/\epsilon$. The algorithm presented here will achieve this output hypothesis size to within a constant. If we assume that the weak learner requires some fixed amount of time and fixed number of examples to generate each hypothesis, then the factor of γ^{-2} must also exist in the time and sample complexity of the algorithm.

6 BOOSTING MULTI-VALUED FUNCTIONS

As was noted by Schapire [Sch91], the generalization of the equivalence between strong and weak learning to concepts with more than two labels does not enjoy the same tightness as the two label case. In the two label case an ability to predict the label with accuracy that is a polynomial fraction better than random guessing is equivalent to strong learning. In the j-label case the probability that a random guess is correct is equal to $1/j$, while the minimal requirement for weak learning to be equivalent to strong learning is to predict correctly with a probability slightly better than a *half*.[5] As any j-valued decision rule can be replaced by $j-1$ binary decision rules of the type: "is the label equal i", the binary boosting algorithm can be used $j-1$ times to generate the desired hypothesis. However, it is possible to perform the boosting process in one pass, generating a simple j-valued hypothesis and eliminating the dependence of the complexity on j. The combination

[4]If $Pr(c(x)=0) \ne Pr(c(x)=1)$ then some decrease in the output hypothesis size is possible. However, the $\Omega(\gamma^{-2})$ dependence is unavoidable.

[5]To realize this, consider a 3-label concept such that for any example there are only *two* possible labels (over the whole concept class). In this case, using a random coin flip to choose one of the two possible labels will give a correct answer half of the time, but the concept class might still be unlearnable [Sch91].

rule that is used is simply the j-valued majority, i.e. the strong hypothesis labels the input with the label given by the largest number of weak hypotheses. The algorithm and its analysis are almost exactly the same as in the binary case, the only difference is that the definition of the filtering factor is based on one more parameter, denoted by t, that is the number of incorrect hypothesis whose output is not equal to the incorrect label with the largest number of votes. For example, suppose the labels are the ten digits, assume the correct label for some example is "0" and the incorrect label that got the largest number of votes is "9" (irrespective of whether the number of votes "9" got is larger than the number of votes "0" got) then t is the number of votes that the digits "1" to "8" got. The change in Formula (1) is that k is replaced by $k - t$:

$$\alpha^i_{r,t} = \begin{cases} \text{if } r > \lfloor \frac{k-t}{2} \rfloor \text{ then } 0 \\ \text{if } i - \lceil \frac{k-t}{2} \rceil \leq r \leq \lfloor \frac{k-t}{2} \rfloor \text{ then} \\ \binom{k-t-i-1}{\lfloor \frac{k-t}{2} \rfloor - r} (\frac{1}{2} + \gamma)^{\lfloor \frac{k-t}{2} \rfloor - r} (\frac{1}{2} - \gamma)^{\lceil \frac{k-t}{2} \rceil - i - 1 + r} \\ \text{if } r < i - \lceil \frac{k-t}{2} \rceil \text{ then } 0 \end{cases} \tag{7}$$

It is interesting to note that the resources required are completely independent of j - the number of possible labels. This is even true if j is different for different n and s, or if j is infinite, even uncountable! However the requirement of weak learning for concepts with uncountable ranges is unreasonably hard. The hypothesis must generate the *exact* correct output for more than half the inputs (in probability). In this case the result described in the next section might be more relevant.

7 BOOSTING REAL-VALUED FUNCTIONS

The boosting algorithm can be used, with little modifications, to transform an algorithm with small average error to an algorithm that makes a small error with high probability.

Assume **C** is a set of functions $f : R \rightarrow R$ and A is a learning algorithm for **C** . Let p be any density function over R, and let $(x_1, f(x_1)), (x_2, f(x_2)), \ldots, (x_n, f(x_n))$ be a set of examples drawn independently at random according to p and labeled according to some $f \in$ **C**. Then A, upon observing this sample, generates a hypothesis function g such that with probability larger than $1 - \delta$

$$\int_{-\infty}^{+\infty} |f(x) - g(x)| dp < d \tag{8}$$

We shall sketch how the boosting algorithm can be used to generate a function h such that with high probability

$$P_p(|f(x) - h(x)| > \frac{d}{1/2 - \gamma}) < \epsilon$$

Where P_p is the probability according to the density p and $\gamma, \epsilon > 0$ are polynomial fractions.

Using the Markov inequality and setting $\Delta = \frac{d}{1/2 - \gamma}$ we get, from Equation (8)

$$P_p(|f(x) - g(x)| > \Delta) < \frac{1}{2} - \gamma$$

We extend the notion of *agreement* between a concept and a hypothesis on an example x to concepts defined on the reals by saying that f and g Δ-agree on x if $|f(x) - g(x)| < \Delta$. using the extended definition of agreement we can say that A is a weak-learner for the concept class **C** . If we replace all the places in the boosting algorithm in which it refers to "agree" or "correct" by corresponding references to "Δ-agree" or "Δ-agrees with the true function", we get a boosting algorithm for real valued functions.

Suppose, for simplicity, that we are using the single-majority combination rule described in [Fre90]. Then the result of running the booster for $k = \frac{1}{2}\gamma^{-2} \log 1/\epsilon$ stages are k functions $\{h_1, \ldots, h_k\}$ such that

$$P_p \left(\#\{h_i \ \Delta\text{-agrees with } f\} < \left\lceil \frac{k}{2} \right\rceil \right) < \epsilon$$

Observe that the fact that more than half of the functions Δ-agree with f implies that the median of the functions Δ-agrees with f. From this we get that the median has the desired property

$$P_p \left(\text{Median}(h_1, h_2, \ldots, h_k) \ \Delta\text{-agrees with } f \right) > 1 - \epsilon$$

8 SUMMARY AND OPEN PROBLEMS

The work on boosting algorithms has both theoretical and practical implications. On the theoretical aspect, improved boosting algorithms provide general upper bounds on the the complexity of PAC learning algorithms. In this respect the main remaining gap seems to be between the upper bound of $O(\frac{1}{\epsilon}(\log \frac{1}{\epsilon})^2)$ and $\Omega(\frac{1}{\epsilon})$ (ignoring dependence on the other parameters). It would be interesting to find a version of the boosting by majority algorithm that does not require prior knowledge of γ – the worst case accuracy of the weak learner. Another interesting question is to what degree can one relax the requirement that the weak learner be γ-correct with respect to *any* input distribution. The set of distributions that might be presented to the weak learner for any concept class and hypothesis class is only a subset of all possible distributions. It would be interesting to characterize families of distributions on which weak learning will suffice to provide strong learning on a given distribution.

The invention of efficient boosting algorithms promised to make the task of finding provably efficient learning algorithms and easier task. It would seem easier to prove that an algorithm is a weak learning algorithm rather than to prove its a strong learning algorithm. However, so far, there has been no applications of this type. Rather than showing that strong learning is easy, the equivalence of weak and strong learning seems to imply that weak PAC learning is hard!

Boosting might be used for increasing the accuracy of *practical* learning algorithms that have been developed in the realm of empirical work in machine learning. In this context it is not necessary to decrease the error from very close to 50% to very close to zero. A decrease from 10% to 5% will often be considered a worthwhile achievement. Thus whether or not an algorithm is a distribution independent PAC learning algorithm is somewhat beside the point, the question is to what degree does its accuracy degrade when the distribution is distorted. If the accuracy degrades slowly enough, then the performance of the weak learner on all the distributions generated during the boosting process might suffice to give a considerably better final hypothesis. This problem can ultimately be answered only empirically. However, a better understanding of the theoretical open problems described above can be of great value in finding boosting algorithms with higher practical significance.

Acknowledgments

I would like to thank Robert Schapire for several helpful discussions and suggestions. I would also like to thank Manfred Warmuth, David Haussler, Nicolo' Cesa-Bianchi, Phil Long, Madhukar Thakur, Ronitt Rubinfeld and Naftaly Tishby for their help in writing this paper.

This work was supported by the Office of Naval Research, contract number N0014-86-K-0454.

References

[AHS85] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive Science*, 9:147–169, 1985.

[BEHW86] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Classifying learnable geometric concepts with the Vapnik-Chervonenkis dimension. In *18th ACM Symposium on Theory of Computing*, pages 273–282, Berkeley, 1986.

[Fre90] Y. Freund. Boosting a weak learning algorithm by majority. In *Proceedings of the 1990 Workshop on Computational Learning Theory*, pages 202–216, San Mateo, CA, 1990. Morgan Kaufmann.

[HKLW91] David Haussler, Michael Kearns, Nick Littlestone, and Manfred K. Warmuth. Equivalence of models for polynomial learnability. *Information and Computation*, 95:129–161, 1991.

[HLW88] David Haussler, Nick Littlestone, and Manfred 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.

[KV89] 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.

[LW89] Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. In *30th Annual IEEE Symposium on Foundations of Computer Science*, pages 256–261, 1989.

[Sch90] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–226, 1990.

[Sch91] Robert E. Schapire. *The Design and Analysis of Efficient Learning Algorithms.* PhD thesis, M.I.T., 1991.

[Sch92] Robert E. Schapire. private correspondence, January 1992.

[Val84] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–42, 1984.