4

Weak and Strong Learning

4.1 A Relaxed Definition of Learning?

There are two parameters that quantify and control the performance of a PAC learning algorithm — the error parameter ϵ and the confidence parameter δ. The smaller the values of these parameters, the stronger the guarantee on the quality of the hypothesis output by the learning algorithm. In our definition of PAC learning, we demanded that a learning algorithm be able to achieve arbitrarily small values for ϵ and δ, and that the running time be polynomially bounded in $1/\epsilon$ and $1/\delta$ (as well as n and $size(c)$).

Suppose that instead of a PAC learning algorithm, we had in our possession a weaker but perhaps still useful algorithm L that could achieve the PAC criteria not for any ϵ and δ but only for some fixed, constant values ϵ_0 and δ_0. Thus, for any target concept $c \in C$ and any distribution $\mathcal{D}$, L manages to find a hypothesis h that with probability at least $1 - \delta_0$ satisfied $error(h) \leq \epsilon_0$, and now L runs in time polynomial in just n and $size(c)$. Is there any way we could use L as a subroutine to obtain an improved algorithm L' that achieved the PAC criteria any values for ϵ and δ?

In this chapter, we show that the answer to this question is positive in the strongest possible sense: even an efficient algorithm whose hypotheses are only slightly better than "random guessing" can be used to obtain an efficient algorithm meeting the definition of PAC learning. By slightly better than random guessing, we mean hypotheses that correctly classify an instance with probability just exceeding $1/2$. Note that if all we desired was the ability to correctly classify instances with probability *exactly* $1/2$, we could always accomplish this by skipping the learning process altogether, and simply flipping a fair coin to classify each new instance! Thus, a hypothesis of error strictly less than $1/2$ is the least nontrivial criterion we could ask a learning algorithm to meet.

More precisely, let $\mathcal{C}$ be a concept class, and let L be an algorithm that is given access to $EX(c, \mathcal{D})$ for target concept $c \in \mathcal{C}_n$ and distribution $\mathcal{D}$. We say that L is a **weak PAC learning algorithm for $\mathcal{C}$ using $\mathcal{H}$** if there are fixed polynomials $p(\cdot,\cdot)$ and $q(\cdot,\cdot)$ such that L outputs a hypothesis $h \in \mathcal{H}$ that with probability at least $1/q(n, size(c))$ satisfies $error(h) \leq 1/2 - 1/p(n, size(c))$. Thus, with only inverse polynomial confidence, L outputs a hypothesis whose predictive ability has only an inverse polynomial advantage over $1/2$.

With this definition, we can now more formally verify that weak PAC learning really is the weakest demand we could place on an algorithm in the PAC setting without trivializing the learning problem. For instance, over the boolean domain $\{0,1\}^n$ we can always obtain error bounded by $1/2 - 1/e(n)$ for some *exponentially* growing function $e(n)$ just by taking a small random sample S of the target concept, and letting our hypothesis be the randomized mapping that classifies an instance according to S if the instance appears in S, and otherwise flips a fair coin to classify the instance. Note that in polynomial time, we could not even detect that this hypothesis gave a slight predictive advantage over random guessing.

Thus, our definition demands that the hypothesis of a weak PAC learning algorithm achieve the least nontrivial generalization from the sample — that is, the least ability to predict the label of instances outside the observed sample. Furthermore, in keeping with our notion that n

and $size(c)$ are natural measures of the complexity of the target concept, we even allow the confidence and the advantage of the hypothesis over random guessing to diminish to 0 as the complexity of the target concept increases.

We will sometimes refer to our original definition of PAC learning as **strong PAC learning** to distinguish it from this new notion. The somewhat surprising main result of this chapter is that if C is efficiently weakly PAC learnable, then C is efficiently strongly PAC learnable.

We prove the equivalence of weak and strong learning by providing an explicit and efficient transformation of a weak PAC learning algorithm into a strong PAC learning algorithm. If ϵ and δ are the desired error and confidence parameters for the strong learning algorithm, the overhead in running time of this transformation is a surprisingly slowly growing function of $1/\epsilon$ and $1/\delta$. The transformation for achieving greater confidence (that is, reducing δ) is entirely straightforward, as we shall see momentarily. The transformation for reducing the error is much more involved, and forms the bulk of this chapter.

An important consequence of the construction used to prove the equivalence is that it shows that any class that is efficiently PAC learnable is in fact efficiently PAC learnable with specific upper bounds on the required resources. For example, using the construction we can prove that if a concept class is efficiently PAC learnable, then it is efficiently PAC learnable by an algorithm whose required memory is (of course) bounded by a polynomial in n and $size(c)$, but by an only *polylogarthmic* function of $1/\epsilon$. (By this we mean polynomial in $\log(1/\epsilon)$.) When contrasted with the lower bound of $\Omega(1/\epsilon)$ on the number of examples required for PAC learning given by Theorem 3.5 (ignoring for now the dependence on all quantities other than ϵ), this shows that there are no concept classes for which efficient PAC learnability requires that the entire sample be contained in memory at one time — there is always another algorithm that "forgets" most of the sample.

Another consequence of the construction is that if C is efficiently PAC learnable, then there is an efficient algorithm taking a sample of m labeled examples of any $c \in \mathcal{C}$, and finding a consistent hypothesis whose size is polynomial in $size(c)$ but only polylogarithmic in m. This gives a strong converse to the results on Occam's Razor presented in Chapter 2.

These and several other interesting consequences of the construction are explored in the exercises at the end of the chapter.

4.2 Boosting the Confidence

We begin our proof of the equivalence of weak and strong learning with the easy part: namely, showing that we can efficiently boost the confidence of a learning algorithm from an inverse polynomial to a value arbitrarily close to 1. Without loss of generality, and for simplicity in the following argument, let us fix a value ϵ for the error parameter, and suppose we have an algorithm L such that for any target concept $c \in \mathcal{C}$ and any distribution $\mathcal{D}$, L outputs h such that $error(h) \leq \epsilon$ with probability only at least $\delta_0 = 1/q(n, size(c))$ for some fixed polynomial $q(\cdot,\cdot)$.

We now show that if we are willing to tolerate the slightly higher hypothesis error $\epsilon + \gamma$ (for $\gamma > 0$ arbitrarily small), then we can achieve arbitrarily high confidence $1 - \delta$ (that is, arbitrarily small confidence parameter δ).

Our new algorithm L' will simulate algorithm L a total of k times (where k will be determined shortly), using an independent sample from $EX(c, \mathcal{D})$ for each simulation. Let $h_1, \ldots, h_k$ denote the hypotheses output by L on these k runs. Then because the simulations are independent, the probability that all of $h_1, \ldots, h_k$ have error larger than ϵ is at most $(1 - \delta_0)^k$. Solving $(1 - \delta_0)^k \leq \delta/2$ yields $k \geq (1/\delta_0)\ln(2/\delta)$ for our choice of k.

The remaining task of L' can now be expressed as follows: given the hypotheses $h_1, \ldots, h_k$, and assuming that at least one has error bounded by ϵ, output one which has error at most $\epsilon + \gamma$ with probability at least $1-\delta/2$. This is easily accomplished by drawing a sufficiently large sample of labeled examples S from $EX(c,\mathcal{D})$, and choosing the h_i that makes the fewest mistakes on S. We will choose S large enough so that with confidence at least $1 - \delta/2$, the empirical error of each h_j on S is within $\gamma/2$ of its true value *error*(h_j) with respect to c and $\mathcal{D}$. This will ensure that with confidence $1 - \delta/2$ the hypothesis that is output has error at most $\epsilon + \gamma$.

For any fixed hypothesis h_j, whether h_j makes a mistake on a randomly drawn example from $EX(c,\mathcal{D})$ can be regarded as a biased coin flip with probability of heads equal to *error*(h_i). By the Chernoff bounds discussed in the Appendix in Chapter 9 the empirical error of h_j on S is an estimate for *error*(h_j) that is accurate to within an additive factor of $\gamma/2$ with confidence at least $1 - \delta/2k$ provided that the number of examples m in S satisfies $m \geq (c_0/\gamma^2)\log(2k/\delta)$ for some appropriate constant $c_0 > 0$.

Now by the union bound, the probability that any of the k hypotheses has empirical error on S deviating from its true error by more than $\gamma/2$ is at most $k(\delta/2k) = \delta/2$. Note that we have arranged things so that the total failure probability — the probability that we fail to get at least one hypothesis of error less than ϵ out of the k runs of L, plus the probability that we fail to choose a hypothesis of error less than $\epsilon + \gamma$ — is at most $\delta/2 + \delta/2 = \delta$.

In summary, the algorithm to boost the confidence from $1/q(n, \mathit{size}(c))$ to $1 - \delta$ (at the expense of an additive factor of γ in the error) is:

- Run L a total of $k = \lceil (1/q(n, \mathit{size}(c)))\ln(2/\delta)\rceil$ times to obtain hypotheses $h_1, \ldots, h_k$.
- Choose a sample S of size $(c_0/\gamma^2)\log(2k/\delta)$ from $EX(c,\mathcal{D})$, and output the h_i that makes fewest mistakes on S.

It is easily verified that the running time is polynomial in n, *size*(c),

$\log(1/\delta)$, $1/\epsilon$ and $1/\gamma$. To eliminate the parameter γ from this argument, if ϵ is our desired error bound, we can make each run of L using the smaller value $\epsilon' = \epsilon/2$ as the error parameter given to L, and then set $\gamma = \epsilon/2$.

4.3 Boosting the Accuracy

We now turn to the harder problem of decreasing the error. Let us assume for now that we are given an algorithm L for C that with high probability outputs a hypothesis with error at most $\beta < 1/2$, where β is fixed for the moment. We would like to transform L into an algorithm that with high probability outputs a hypothesis with error at most ϵ, where $0 \leq \epsilon < \beta$ is any given error parameter. We will eventually substitute the value $1/2 - 1/p(n, size(c))$ for β in accordance with the definition of weak learning.

It should be readily apparent that the problem of boosting the accuracy is much harder than the problem of confidence boosting. In the confidence boosting problem, the available learning algorithm did manage to output an acceptable hypothesis (that is, one with the desired error bound ϵ) with probability $1/q(n, size(c))$, so a small pool of independent hypotheses contained an acceptable hypothesis with high probability, and thus confidence boosting involved simply identifying a good hypothesis from the small pool. In contrast, we are now faced with the situation where the available learning algorithm might *always* output a hypothesis with an unacceptably high error (that is, error larger than ϵ).

At first glance, this might make our stated goal seem impossible to achieve. The key to obtaining the result lies in the fact that even though the available learning algorithm L may output hypotheses with unacceptably large error β, it is guaranteed to do so for any distribution on the input space. The idea will be to run L many times, not only on the target distribution $\mathcal{D}$, but also on other related probability distributions which somehow "focus the attention" of L on regions in which the hypotheses of previous runs perform poorly. For example, after running L once to obtain a hypothesis h of error at most β with respect to $\mathcal{D}$, we could then run L again, but only giving L those inputs from $\mathcal{D}$ on which h errs — the intuition being that the second run of L is forced to learn something "new". While this idea does not quite work, a variation of it will, and the many hypotheses output by L on multiple runs will be judiciously combined to yield a new hypothesis with error at most ϵ on the original distribution $\mathcal{D}$.

We will present the accuracy boosting construction in two parts. First, we define and analyze a simple and modest accuracy boosting procedure. Given a learning algorithm L that outputs hypotheses of error at most β, this procedure uses L as a subroutine and outputs hypotheses of error at most $g(\beta) < \beta$, for some function $g(\beta)$ that we will specify shortly. To do this, the procedure defines a sequence of three probability distributions on the input space. It then invokes L three times on these three distributions to obtain three hypotheses h_1, h_2 and h_3. These hypotheses are combined into the single function h that takes the majority vote of h_1, h_2 and h_3 and which forms the output of the procedure. The hypothesis h is guaranteed to have error at most $g(\beta)$, which still may be much larger than the desired value ϵ.

In the second and more complex part of the construction, we use this modest boosting procedure repeatedly in a recursive fashion in order to drive the error down to ϵ.

4.3.1 A Modest Accuracy Boosting Procedure

Throughout this section, we shall assume $c \in C$ is the target concept, $\mathcal{D}$ is the target distribution, and L is an algorithm that with high probability outputs hypotheses with error at most $\beta < 1/2$ when given access to $EX(c, \mathcal{D})$. When there is ambiguity about the distribution, we use $\mathit{error}_{\mathcal{D}}(h)$ to explicitly indicate the error of h with respect to c and $\mathcal{D}$.

Our modest accuracy boosting procedure operates as follows. To begin, algorithm L is invoked on the oracle $EX(c,\mathcal{D})$. We let h_1 denote the hypothesis output by L.

Now we run L again, but this time on a new distribution. Intuitively, this second distribution is designed to extract new information about the target concept — information that was absent in h_1. More precisely, the new distribution $\mathcal{D}_2$ is created by **filtering** $\mathcal{D}$ with respect to the first hypothesis h_1.

Distribution $\mathcal{D}_2$ is defined as follows: to sample from $EX(c,\mathcal{D}_2)$, we first flip a fair coin. If the outcome is heads, we draw labeled examples from $EX(c,\mathcal{D})$ until an example $\langle x, c(x)\rangle$ is drawn such that $h_1(x) = c(x)$ and output $\langle x, c(x)\rangle$. If the outcome is tails, then we draw labeled examples from $EX(c,\mathcal{D})$ until an example $\langle x, c(x)\rangle$ is drawn such that $h_1(x) \neq c(x)$ and output $\langle x, c(x)\rangle$.

Thus, $\mathcal{D}_2$ is essentially $\mathcal{D}$ normalized to give weight exactly 1/2 to those instances on which h_1 errs; the relative weight of two such instances, however, is unaltered, as is the relative weight of two instances on which h_1 is correct. $\mathcal{D}_2$ is constructed so that h_1 has no advantage over random guessing: that is, $error_{\mathcal{D}_2}(h_1) = 1/2 > \beta$. Thus, invoking L on $EX(c,\mathcal{D}_2)$ yields a hypothesis h_2 which gives us "new information" about c, that is, information not contained in h_1. In particular, we must have $h_1 \neq h_2$.

It is important to note that we can sample from $EX(c,\mathcal{D}_2)$ given access to $EX(c,\mathcal{D})$, and that the expected number of calls we need to $EX(c,\mathcal{D})$ to simulate a single call to $EX(c,\mathcal{D}_2)$ becomes large only if $error_{\mathcal{D}}(h_1) \approx 0$ or $error_{\mathcal{D}}(h_1) \approx 1$. Roughly speaking, neither of these is a major problem since if $error_{\mathcal{D}}(h_1) \approx 0$ then h_1 already has error significantly smaller than β, and $error_{\mathcal{D}}(h_1) \approx 1 > \beta$ cannot happen if L meets its guarantee when run on $EX(c,\mathcal{D})$. However, we shall rigorously handle the issue of the efficiency of filtering later, and for now we simply assume we can sample $EX(c,\mathcal{D}_2)$ in unit time.

For the third simulation of L, we create a third distribution $\mathcal{D}_3$ by again filtering $\mathcal{D}$, this time with respect to both h_1 and h_2. In order to sample from $EX(c, \mathcal{D}_3)$, we draw labeled examples from $EX(c, \mathcal{D})$ until an example $\langle x, c(x) \rangle$ is drawn such that $h_1(x) \neq h_2(x)$, and then output $\langle x, c(x) \rangle$. Invoking L on $EX(c, \mathcal{D}_3)$ yields a hypothesis h_3 which once again gives gives "new information" about c, this time on those inputs such that h_1 and h_2 disagree. We again defer analysis of how efficiently we can sample $EX(c, \mathcal{D}_3)$, and for now we assume we can sample $EX(c, \mathcal{D}_3)$ in unit time.

Finally, the output of the modest accuracy boosting procedure is $h = \mathit{majority}(h_1, h_2, h_3)$; by this we mean that $h(x) = 1$ if and only if at least 2 out of 3 of $h_1(x), h_2(x)$ and $h_3(x)$ are 1. Let us introduce the notation $\beta_1 = \mathit{error}_{\mathcal{D}}(h_1)$, $\beta_2 = \mathit{error}_{\mathcal{D}_2}(h_2)$ and $\beta_3 = \mathit{error}_{\mathcal{D}_3}(h_3)$. Our goal now is to argue that even though β_1,β_2 and β_3 may all be as large as β, $\mathit{error}_{\mathcal{D}}(h)$ is significantly smaller than β.

4.3.2 Error Analysis for the Modest Procedure

Before embarking on the specifics of this argument, it will be helpful to introduce a technical fact that we shall use repeatedly. Throughout the chapter, we will need to map the probability of an event with respect to $\mathcal{D}_2$ back to its probability with respect to $\mathcal{D}$. More precisely, consider any instance x. By the definition of $\mathcal{D}_2$, given the value of $\mathcal{D}_2[x]$ (the weight of x under $\mathcal{D}_2$), the weight $\mathcal{D}[x]$ is completely determined by whether $h_1(x) = c(x)$ or $h_1(x) \neq c(x)$. Thus, if $h_1(x) = c(x)$ then $\mathcal{D}[x] = 2(1 - \beta_1)\mathcal{D}_2[x]$. To see this, note that the transformation of $\mathcal{D}$ to $\mathcal{D}_2$ changes the total weight of the instances where $h_1(x) = c(x)$ from $1 - \beta_1$ to $1/2$, or viewed in reverse, an instance x such that $h_1(x) = c(x)$ and $\mathcal{D}_2[x] = p$ must have had weight $2(1 - \beta_1)p$ under $\mathcal{D}$. Similarly, if $h_1(x) \neq c(x)$ then $\mathcal{D}[x] = 2\beta_1\mathcal{D}_2[x]$. More generally, for any set $S \subseteq X$ we can write

$$\begin{aligned}\mathbf{Pr}_{x \in \mathcal{D}}[x \in S] \;=\;& 2(1 - \beta_1)\mathbf{Pr}_{x \in \mathcal{D}_2}[h_1(x) = c(x) \wedge x \in S] \\ & + 2\beta_1 \mathbf{Pr}_{x \in \mathcal{D}_2}[h_1(x) \neq c(x) \wedge x \in S]. \qquad (4.1)\end{aligned}$$

In what follows, we will repeatedly obtain expressions for probabilities over $\mathcal{D}$ by first decomposing those probabilities over $\mathcal{D}_2$ into their $h_1 = c$ and $h_1 \neq c$ components, and then obtaining the probability with respect to $\mathcal{D}$ via Equation 4.1.

We now embark on the analysis of the error of the hypothesis found by the modest accuracy boosting procedure. The following lemma gives an upper bound of $g(\beta) = 3\beta^2 - 2\beta^3$ on this error with respect to the original distribution $\mathcal{D}$. The function $g(\beta)$ is important to all of the ensuing analysis, so we pause to note two important facts. First, for any $0 \leq \beta < 1/2$ we have $g(\beta) < \beta$, and $g(1/2) = 1/2$. Second, $g(\beta)$ has a well-defined inverse $g^{-1}(\alpha)$ obeying $g^{-1}(\alpha) > \alpha$ for $0 \leq \alpha < 1/2$.

Lemma 4.1 *Let $g(\beta) = 3\beta^2 - 2\beta^3$. Let the distributions $\mathcal{D}$, $\mathcal{D}_2$ and $\mathcal{D}_3$ be as defined above, and let h_1, h_2 and h_3 satisfy $error_{\mathcal{D}}(h_1) \leq \beta$, $error_{\mathcal{D}_2}(h_2) \leq \beta$, and $error_{\mathcal{D}_3}(h_3) \leq \beta$. Then if $h = majority(h_1, h_2, h_3)$, $error_{\mathcal{D}}(h) \leq g(\beta)$.*

Proof: We will show in stages that the error of h is maximized when $\beta_i = \beta$ for all i, that is, when each h_i has the maximum allowed error with respect to its corresponding distribution. The bound of $g(\beta)$ on the error of h with respect to $\mathcal{D}$ will then follow quite easily.

For the purposes of analysis, we will decompose the errors of h into two mutually exclusive types. The first type of error is on those inputs where both h_1 and h_2 already make an error. Note that the output of h_3 is irrelevant in this case, since the majority is already determined by h_1 and h_2, and this majority gives the incorrect label. The second type of error is on those inputs where h_1 and h_2 disagree. In this case, the hypothesis h_3 gets to cast the deciding vote for the majority, and therefore h makes an error if and only if h_3 does. This yields:

$$\begin{aligned} error_{\mathcal{D}}(h) \; = \; & \mathbf{Pr}_{x\in\mathcal{D}}[h_1(x) \neq c(x) \wedge h_2(x) \neq c(x)] \\ & + \mathbf{Pr}_{x\in\mathcal{D}}[h_3(x) \neq c(x) | h_1(x) \neq h_2(x)] \mathbf{Pr}_{x\in\mathcal{D}}[h_1(x) \neq h_2(x)] \end{aligned}$$

$$= \mathbf{Pr}_{x\in\mathcal{D}}[h_1(x) \neq c(x) \wedge h_2(x) \neq c(x)] + \beta_3 \mathbf{Pr}_{x\in\mathcal{D}}[h_1(x) \neq h_2(x)] \tag{4.2}$$

The last equality comes from the fact that choosing randomly from $\mathcal{D}$ conditioned on $h_1(x) \neq h_2(x)$ is the same as choosing from $\mathcal{D}_3$, since this is exactly the definition of $\mathcal{D}_3$.

Our goal now is to use Equation (4.2) to obtain an upper bound on $error_{\mathcal{D}}(h)$ that is first an algebraic expression over the β_i, and eventually over only β. From this expression it will be a simple matter to bound $error_{\mathcal{D}}(h)$ by $g(\beta)$.

To begin with, it is clear from Equation (4.2) that $error_{\mathcal{D}}(h)$ is maximized when $\beta_3 = \beta$, because no other term in the equation depends on $\mathcal{D}_3$ and h_3. This gives

$$error_{\mathcal{D}}(h) \leq \mathbf{Pr}_{x\in\mathcal{D}}[h_1(x) \neq c(x) \wedge h_2(x) \neq c(x)] + \beta \mathbf{Pr}_{x\in\mathcal{D}}[h_1(x) \neq h_2(x)] \tag{4.3}$$

It remains to analyze the two probabilities involving h_1 and h_2 in Inequality 4.3. We can represent the distributions $\mathcal{D}$ and $\mathcal{D}_2$ as shown in Figure 4.1. Distribution $\mathcal{D}_2$ can be partitioned into two equal parts as shown: one corresponding to those inputs x where $h_1(x) = c(x)$ and the other corresponding to $h_1(x) \neq c(x)$. As shown, let γ_1 and γ_2 respectively be the error of h_2 on each of these parts, that is,

$$\gamma_1 = \mathbf{Pr}_{x\in\mathcal{D}_2}[h_1(x) = c(x) \wedge h_2(x) \neq c(x)]$$

and

$$\gamma_2 = \mathbf{Pr}_{x\in\mathcal{D}_2}[h_1(x) \neq c(x) \wedge h_2(x) \neq c(x)].$$

Clearly we have $\gamma_1 + \gamma_2 = \beta_2$.

Using the expression for γ_1 and using Equation 4.1 to go back to $\mathcal{D}$, we may write

$$\mathbf{Pr}_{x\in\mathcal{D}}[h_1(x) = c(x) \wedge h_2(x) \neq c(x)] = 2(1-\beta_1)\gamma_1. \tag{4.4}$$

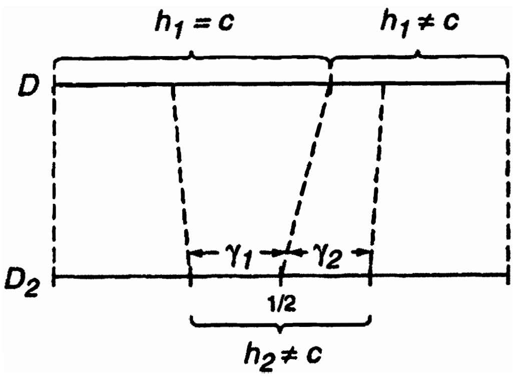

Figure 4.1: *The mapping of distribution $\mathcal{D}$ to distribution $\mathcal{D}_2$. The region $h_1 = c$ "shrinks" from weight $1 - \beta_1 > 1/2$ under $\mathcal{D}$ to weight exactly $1/2$ under $\mathcal{D}_2$, while the region $h_1 \neq c$ "expands" from weight $\beta_1 < 1/2$ under $\mathcal{D}$ to weight exactly $1/2$ under $\mathcal{D}_2$.*

Also note (see Figure 4.1) that

$$\mathbf{Pr}_{x\in\mathcal{D}_2}[h_1(x) \neq c(x) \wedge h_2(x) = c(x)] = 1/2 - \gamma_2$$

and thus by Equation 4.1

$$\mathbf{Pr}_{x\in\mathcal{D}}[h_1(x) \neq c(x) \wedge h_2(x) = c(x)] = 2\beta_1(1/2 - \gamma_2). \qquad (4.5)$$

Using Equations (4.4) and (4.5) and the fact that

$$\begin{aligned}&\mathbf{Pr}_{x\in\mathcal{D}}[h_1(x) \neq h_2(x)] = \\ &\qquad \mathbf{Pr}_{x\in\mathcal{D}}[h_1(x) = c(x) \wedge h_2(x) \neq c(x)] \\ &\qquad + \mathbf{Pr}_{x\in\mathcal{D}}[h_1(x) \neq c(x) \wedge h_2(x) = c(x)]\end{aligned}$$

we get

$$\mathbf{Pr}_{x\in\mathcal{D}}[h_1(x) \neq h_2(x)] = 2(1 - \beta_1)\gamma_1 + 2\beta_1(1/2 - \gamma_2). \qquad (4.6)$$

Now using the expression for γ_2 and again using Equation 4.1 to go back to $\mathcal{D}$, we may write

$$\mathbf{Pr}_{x\in\mathcal{D}}[h_1(x) \neq c(x) \wedge h_2(x) \neq c(x)] = 2\beta_1\gamma_2. \qquad (4.7)$$

Substituting Equations (4.6) and (4.7) into Inequality (4.3), we get:

$$\begin{aligned} error_{\mathcal{D}}(h) &\leq 2\beta_1\gamma_2 + \beta(2(1-\beta_1)\gamma_1 + 2\beta_1(1/2-\gamma_2)) \\ &= \beta_1\beta(1-2\gamma_1) + 2\beta_1\gamma_2(1-\beta) + 2\gamma_1\beta. \end{aligned}$$

The last equality can be easily verified algebraically. The coefficient of β_1 in this expression is $\beta(1-2\gamma_1) + 2\gamma_2(1-\beta)$, which is non-negative because $\beta, \gamma_2 \geq 0$ and $\beta, \gamma_1 < 1/2$. Thus the error of h is maximized if $\beta_1 = \beta$, the maximum allowed value for β_1. This, along with some more algebra, allows the expression for the error of h to be further simplified to

$$error_{\mathcal{D}}(h) \leq \beta^2 + 2\beta(1-\beta)(\gamma_1+\gamma_2).$$

This is maximized if $\gamma_1 + \gamma_2 = \beta_2$ is set to its maximum value β. This yields

$$error_{\mathcal{D}}(h) \leq 3\beta^2 - 2\beta^3$$

as desired. □(Lemma 4.1)

4.3.3 A Recursive Accuracy Boosting Algorithm

So far, we have only given a procedure that drives the error down from β to $g(\beta)$. We now consider the larger problem, where we are given a learning algorithm L with an error bound of only $1/2 - 1/p(n, size(c))$ for some polynomial $p(\cdot,\cdot)$ and we wish to construct a new learning algorithm whose error bound can be made as small as any input ϵ. The basic idea for tackling this problem is quite simple: we will just invoke the modest accuracy boosting mechanism recursively, until the desired error bound is obtained.

Let us be a little more precise. Suppose we are given a desired error bound ϵ. If we had in our possession a learning algorithm L whose hypotheses were guaranteed to have error bounded by $g^{-1}(\epsilon)$ (which is larger than ϵ), then we have already shown how we could make three calls to L using filtered distributions to obtain a hypothesis with error $g(g^{-1}(\epsilon)) = \epsilon$, and we would be done. However, we may not have such an L available. But if we only had an L whose error bound was $g^{-1}(g^{-1}(\epsilon))$, then we could apply our boosting procedure once to improve the error bound to $g(g^{-1}(g^{-1}(\epsilon))) = g^{-1}(\epsilon)$, then again a second time to get the error down to ϵ. Since we regard ϵ as an input parameter, and we must run in time polynomial in $1/\epsilon$, a primary concern with such an approach is how deeply we must nest such recursive calls as a function of ϵ.

The tentative description of the recursive algorithm follows; we will shortly make some small but important modifications. The algorithm takes two arguments as input: a desired error bound α (we discuss the confidence parameter δ momentarily), and an oracle $EX(c, \mathcal{D}')$ for examples. As before, L denotes the assumed weak learning algorithm, and we will use $L(EX(c, \mathcal{D}'))$ to denote the hypothesis returned by an invocation of L using the oracle $EX(c, \mathcal{D}')$.

Algorithm Strong-Learn(α,$EX(c, \mathcal{D}')$):

- If $\alpha \geq 1/2 - 1/p(n, size(c))$ then return $L(EX(c, \mathcal{D}'))$. In this case, the error parameter α for this call to **Strong-Learn** can already be achieved by the weak PAC learning algorithm L.
- $\beta \leftarrow g^{-1}(\alpha)$. Here β is the error we require from the level of recursion below us if we are to achieve error α.
- Let $\mathcal{D}'_2$ and $\mathcal{D}'_3$ be obtained by filtering $\mathcal{D}'$ as described in the modest boosting procedure.
- $h_1 \leftarrow$ **Strong-Learn**($\beta, EX(c, \mathcal{D}')$).
- $h_2 \leftarrow$ **Strong-Learn**($\beta, EX(c, \mathcal{D}'_2)$).

- $h_3 \leftarrow$ **Strong-Learn**$(\beta, EX(c, \mathcal{D}_3'))$.
- $h \leftarrow majority(h_1, h_2, h_3)$.
- Return h.

Throughout the coming analysis, it will be helpful to think about the execution of this recursive algorithm as a ternary tree. Each node of the tree is labeled by two quantities: a (possibly filtered) oracle for random examples of the target concept, and a desired error bound for this node. The root of the tree is labeled by the oracle $EX(c, \mathcal{D})$ (where c is the target concept and $\mathcal{D}$ is the target distribution) and by the final desired error bound ϵ. Now any node labeled by $EX(c, \mathcal{D}')$ and α has either three children or is a leaf, as we now describe.

If the label $\alpha < 1/2 - 1/p(n, size(c))$ then the desired error bound for this node is still too small to be obtained by invocation of the weak learning algorithm L. In this case, the three children will be labeled by the oracles $EX(c, \mathcal{D}')$, $EX(c, \mathcal{D}_2')$ and $EX(c, \mathcal{D}_3')$ as specified by our modest accuracy boosting procedure, and all three children will be labeled by the larger error bound of $\beta = g^{-1}(\alpha)$. This can be interpreted as a request from the parent node to its children for hypotheses of error at most β on the filtered distributions, for if these are supplied by the children then the parent can take the majority and fulfill its desired error bound of α. Thus, these three children correspond to a recursive call by our algorithm.

If, on the other hand, $\alpha \geq 1/2 - 1/p(n, size(c))$ then the desired error bound for this node can be immediately satisfied by a call to the weak learning algorithm L using the oracle $EX(c, \mathcal{D}')$. Then this node will be a leaf, and corresponds to a base case of the recursion.

Note that in this ternary tree, the oracle labeling any node is actually implemented by making calls to the oracle of its parent, which in turn is implemented by making calls to the oracle of its parent, and so on to the root node with oracle $EX(c, \mathcal{D})$.

To simplify the analysis of our recursive algorithm, we will defer until later the issue of the dependence on the confidence parameter. More precisely, in addition to an examples oracle $EX(c, \mathcal{D}')$ and an error parameter α, our algorithm really should also be taking a confidence parameter δ, which is our allowed probability of failing to output a hypothesis of error bounded by α with respect to c and $\mathcal{D}'$. Now in our algorithm, there will many steps at which the algorithm could potentially fail locally, thereby causing a global failure to output a good hypothesis. For instance, if any call to the weak algorithm L fails, or any recursive call of our algorithm fails, we may fail to find a good hypothesis. But for now, we will simply assume that all such steps succeed and analyze the complexity and correctness of the algorithm under this assumption. In other words, for now we will simply ignore the confidence parameter δ. Towards the end of the analysis, it will be easy to reintroduce δ and the possibility of failed steps by giving a bound N on the total number of possible places the algorithm could fail in any execution, and allocating probability at most δ/N to each of these.

We begin the analysis by bounding the maximum depth of the ternary recursion tree induced by the execution of algorithm **Strong-Learn**.

4.3.4 Bounding the Depth of the Recursion

Let $B(\epsilon, p(n, size(c)))$ denote the *depth* (that is, the longest path from the root to a leaf) of the execution tree whose root is labeled by the oracle $EX(c, \mathcal{D})$ and error parameter ϵ when the given weak learning algorithm has a $1/2 - 1/p(n, size(c))$ error bound. Thus, $B(\epsilon, p(n, size(c)))$ is the maximum nesting depth of the recursive calls to the procedure **Strong-Learn**. It is considerably less than the total number of invocations of **Strong-Learn** (which is the total number of nodes in the execution tree), but its analysis will lead to a bound on this total as well.

Lemma 4.2

$$B(\epsilon, p(n, size(c))) = O(\log p(n, size(c)) + \log\log(1/\epsilon)).$$

Proof: To prove this lemma, we first argue that the number of recursive calls to **Strong-Learn** needed to drive the error from its largest value of $1/2 - 1/p(n, size(c))$ down to $1/4$ is $O(\log p(n, size(c)))$. In other words, the depth of any subtree of the execution tree whose error parameter label is $1/4$ or larger is $O(\log p(n, size(c)))$.

To see this, consider any node with a desired error bound of $\beta > 1/4$. The distance of this value to $1/2$ is just $1/2 - \beta$, and the distance to $1/2$ of the desired error bound of the node's parent is the larger value $1/2 - g(\beta)$. We wish to argue that this distance is actually moving away from $1/2$ by a constant multiplicative factor with each invocation of the modest boosting procedure. This is because

$$1/2 - g(\beta) = 1/2 - 3\beta^2 + 2\beta^3 = (1/2 - \beta)(1 + 2\beta - 2\beta^2).$$

It is easy to show using basic calculus that for $\beta \geq 1/4$, the second factor is at least $11/8$. Thus a single invocation of the modest boosting procedure increases the distance of the error bound from $1/2$ by a multiplicative factor of $11/8$. Thus, $\log_{11/8}(p(n, size(c))/4)$ levels of recursion suffice to drive the error bound down to $1/4$.

For $\beta \leq 1/4$, things are even better: the error bound decreases at a doubly exponential rate! To see this, now we simply look at the rate at which the error itself decreases, and we see that in one step β is replaced by $g(\beta) = 3\beta^2 - 2\beta^3 \leq 3\beta^2$. Therefore in k steps the error is at most $(1/3)(3\beta)^{2^k}$. Since $\beta \leq 1/4$, this is at most $(3/4)^{2^k}$. Solving for $(3/4)^{2^k} \leq \epsilon$, we find that $k \geq c_0 \log\log(1/\epsilon)$ suffices for some constant $c_0 > 0$. □(Lemma 4.2)

4.3.5 Analysis of Filtering Efficiency

We are now ready to tackle one of the main issues we have been avoiding so far: how do we bound the time required to obtain examples from the filtered distributions at each node of the recursion tree given that in reality we have direct access only to the root oracle $EX(c, \mathcal{D})$? It turns out that in order to obtain such a bound, we will need to modify algorithm **Strong-Learn** slightly.

Recall that there were two types of filtering performed at any node in the execution tree labeled by oracle $EX(c, \mathcal{D}')$ and error parameter α: the filtered oracle $EX(c, \mathcal{D}'_2)$ was implemented by filtering $\mathcal{D}'$ with respect to the hypothesis h_1 returned by the child labeled by $EX(c, \mathcal{D}')$ and $g^{-1}(\alpha)$. With probability $1/2$, this involved waiting for an example $\langle x, c(x)\rangle$ such that $h_1(x) \neq c(x)$. The expected waiting time is $1/error_{\mathcal{D}'}(h_1)$, which is unacceptably large if $error_{\mathcal{D}'}(h_1)$ is small.

To handle this situation, we simply add a test to make sure that $error_{\mathcal{D}'}(h_1)$ is not "too small" before we attempt to make a recursive call with the filtered oracle $EX(c, \mathcal{D}'_2)$. More precisely, after recursively calling **Strong-Learn** on $\beta = g^{-1}(\alpha)$ and $EX(c, \mathcal{D}')$ to obtain h_1, we draw a sufficiently large number m of random examples from $EX(c, \mathcal{D}')$ and use this sample to obtain an empirical estimate $\hat{e}_1$ for $error_{\mathcal{D}'}(h_1)$. The sample size m will be sufficiently large to ensure that

$$error_{\mathcal{D}'}(h_1) - \alpha/3 \leq \hat{e}_1 \leq error_{\mathcal{D}'}(h_1) + \alpha/3.$$

Thus our estimate is accurate within an additive error of $\alpha/3$. Although we shall not compute the required sample size precisely, it is bounded by an inverse polynomial in α, and it is a straightforward exercise to derive it using the Chernoff bounds described in the Appendix in Chapter 9.

Now if $\hat{e}_1 \leq 2\alpha/3$, then we know $error_{\mathcal{D}'}(h_1) \leq \alpha$, and we can already return h_1 without performing the recursive calls. Otherwise, if $\hat{e}_1 > 2\alpha/3$, then we know $error_{\mathcal{D}'}(h_1) \geq \alpha/3$, and therefore the expected number of calls to the oracle $EX(c, \mathcal{D}')$ needed to simulate one call to the filtered oracle $EX(c, \mathcal{D}'_2)$ is at most $3/\alpha$.

Bounding the expected number of calls to $EX(c, \mathcal{D}')$ to implement one call to the filtered oracle $EX(c, \mathcal{D}'_3)$ is more involved, and again involves a modification to algorithm **Strong-Learn**. Let h_2 denote the hypothesis returned by the recursive call using oracle $EX(c, \mathcal{D}'_2)$ and error parameter $\beta = g^{-1}(\alpha)$. Then before making the recursive call to

Strong-Learn with oracle $EX(c, \mathcal{D}'_3)$ and error parameter β, we sample from $EX(c, \mathcal{D}')$ to obtain an estimate $\hat{e}_2$ for $error_{\mathcal{D}'}(h_2)$. We take enough examples so that $\hat{e}_2$ is accurate to within additive error τ, where we define $\tau = ((1-2\beta)/8) \cdot \alpha$. Again the required sample size can be easily derived using Chernoff bounds, and is bounded by an inverse polynomial in τ.

Now if $\hat{e}_2 \le \alpha - \tau$ then we know we have already reached our goal of $error_{\mathcal{D}'}(h_2) \le \alpha$, and we can simply ignore h_1, not make the third recursive call, and return h_2. On the other hand, if $\hat{e}_2 > \alpha - \tau$ then we know $error_{\mathcal{D}'}(h_2) \ge \alpha - 2\tau$, and we will prove below that this (along with the fact that $error_{\mathcal{D}'}(h_1) \ge \alpha/3$) implies $\mathbf{Pr}_{x \in \mathcal{D}'}[h_1(x) \neq h_2(x)] \ge \alpha/24$. Thus the expected number of calls to $EX(c, \mathcal{D}')$ needed to simulate one call to $EX(c, \mathcal{D}'_3)$ is at most $24/\alpha$, and we can safely proceed with the recursive call.

Before verifying this claim, we present the modified algorithm just outlined.

Algorithm Strong-Learn(α,$EX(c, \mathcal{D}')$):

- If $\alpha \ge 1/2 - 1/p(n, size(c))$ then return $L(EX(c, \mathcal{D}'))$.
- $\beta \leftarrow g^{-1}(\alpha)$.
- Let $\mathcal{D}'_2$ and $\mathcal{D}'_3$ be obtained by filtering $\mathcal{D}'$ as described in the modest boosting procedure.
- $h_1 \leftarrow$ **Strong-Learn**($\beta, EX(c, \mathcal{D}')$).
- Compute an estimate $\hat{e}_1$ for $error_{\mathcal{D}'}(h_1)$ that is accurate within additive error $\alpha/3$.
- If $\hat{e}_1 \le 2\alpha/3$ then return h_1.
- $h_2 \leftarrow$ **Strong-Learn**($\beta, EX(c, \mathcal{D}'_2)$).

- Compute an estimate $\hat{e}_2$ for $error_{\mathcal{D}'}(h_2)$ that is accurate within additive error $\tau = ((1 - 2\beta)/8) \cdot \alpha$.
- If $\hat{e}_2 \leq \alpha - \tau$ then return h_2.
- $h_3 \leftarrow$ **Strong-Learn**$(\beta, EX(c, \mathcal{D}'_3))$.
- $h \leftarrow majority(h_1, h_2, h_3)$.
- Return h.

Lemma 4.3 *Let a node of the execution tree be labeled by oracle $EX(c, \mathcal{D}')$ and error bound α, and let $\beta = g^{-1}(\alpha)$. Let h_1 be the hypothesis returned by the child labeled with $EX(c, \mathcal{D}')$ and β, and let h_2 be the hypothesis returned by the child labeled with $EX(c, \mathcal{D}'_2)$ and β. Let $\tau = ((1 - 2\beta)/8) \cdot \alpha$. Then if $error_{\mathcal{D}'}(h_1) \geq \alpha/3$ and $error_{\mathcal{D}'}(h_2) \geq \alpha - 2\tau$,*

$$\mathbf{Pr}_{x \in \mathcal{D}'}[h_1(x) \neq h_2(x)] \geq \alpha/24.$$

Proof: Let us define

$$\gamma_2 = \mathbf{Pr}_{x \in \mathcal{D}'_2}[h_1(x) \neq c(x) \wedge h_2(x) \neq c(x)] \tag{4.8}$$

as was done in the proof of Lemma 4.1. Note that if we have an upper bound on γ_2 that is strictly less than 1/2, then we have a lower bound on

$$\mathbf{Pr}_{x \in \mathcal{D}'_2}[h_1(x) \neq c(x) \wedge h_2(x) = c(x)] = 1/2 - \gamma_2$$

that is strictly greater than 0. Furthermore, we can translate this this latter probability back to $\mathcal{D}'$ to obtain a lower bound on

$$\mathbf{Pr}_{x \in \mathcal{D}'}[h_1(x) \neq c(x) \wedge h_2(x) = c(x)]$$

and this finally is a lower bound on $\mathbf{Pr}_{x \in \mathcal{D}'}[h_1(x) \neq h_2(x)]$.

We will now prove that subject to the condition $error_{\mathcal{D}'}(h_2) \geq \alpha - 2\tau$, γ_2 is at most $7/16 < 1/2$. To do this, we maximize γ_2 subject to the condition $error_{\mathcal{D}'}(h_2) \geq \alpha - 2\tau$.

For the purposes of this maximization, we can assume that $error_{\mathcal{D}'}(h_1)$ and $error_{\mathcal{D}'_2}(h_2)$ are as large as possible (which is β, the error parameter given to the recursive calls that compute h_1 and h_2), as shown in Figure 4.2. This is because for any fixed values of $error_{\mathcal{D}'}(h_1)$ and $error_{\mathcal{D}'_2}(h_2)$, if we "added" a region of error to h_1 or h_2 it could only increase the region of their common errors, and thus the value of γ_2.

Thus we set $error_{\mathcal{D}'}(h_1) = error_{\mathcal{D}'_2}(h_2) = \beta$ and hence

$$\mathbf{Pr}_{x \in \mathcal{D}'_2}[h_1(x) = c(x) \wedge h_2(x) \neq c(x)] = \gamma_1 = \beta - \gamma_2. \tag{4.9}$$

Now by decomposing $error_{\mathcal{D}'}(h_2)$ into its $h_1 = c$ and $h_2 \neq c$ components, and using Equation (4.1) to translate Equations (4.8) and (4.9) back to $\mathcal{D}'$, we write

$$\begin{aligned} error_{\mathcal{D}'}(h_2) &= 2\beta\gamma_2 + 2(1-\beta)(\beta-\gamma_2) \\ &= 2\gamma_2(2\beta-1) + 2\beta - 2\beta^2. \end{aligned} \tag{4.10}$$

Also, under the constraint that $error_{\mathcal{D}'}(h_2) \geq \alpha - 2\tau$ we have

$$\begin{aligned} error_{\mathcal{D}'}(h_2) &\geq \alpha - 2\tau \\ &= \alpha(1 - (1-2\beta)/4) \\ &= g(\beta)(1 - (1-2\beta)/4) \\ &= (3\beta^2 - 2\beta^3)(3+2\beta)/4. \end{aligned} \tag{4.11}$$

Putting Equations (4.10) and (4.11) together and multiplying both by 4, we obtain

$$8\gamma_2(2\beta-1) + 8\beta - 8\beta^2 \geq (9\beta^2 - 4\beta^4).$$

Thus

$$8\gamma_2(2\beta-1) \geq (-8\beta + 17\beta^2 - 4\beta^4) = \beta(2\beta-1)(8-\beta-2\beta^2).$$

Therefore, since $(2\beta - 1) < 0$,

$$\gamma_2 \leq \beta/8(8-\beta-2\beta^2).$$

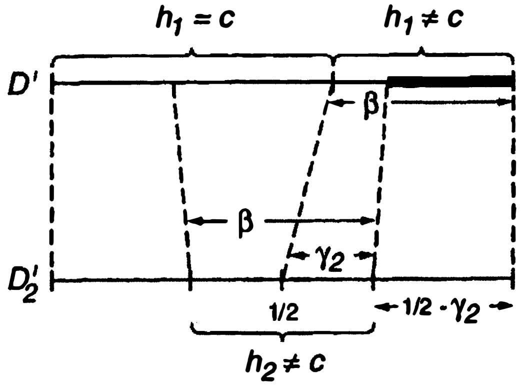

Figure 4.2: *The mapping of distribution $\mathcal{D}'$ to distribution $\mathcal{D}_2'$, with maximum error β for h_1 and h_2. The current goal is to lower bound the shaded region, which is the weight under $\mathcal{D}'$ of the compound event $h_1 \neq c$ and $h_2 = c$ (which implies the desired event $h_1 \neq h_2$).*

Some basic calculus shows that the maximum value of this expression in the range $\beta \in [0, 1/2]$ occurs at $\beta = 1/2$, and is 7/16. This completes the proof of the claimed bound on γ_2 subject to the constraint $error_{\mathcal{D}'}(h_2) \geq \alpha - 2\tau$.

Now since $\gamma_2 \leq 7/16$, we have that

$$\mathbf{Pr}_{x \in \mathcal{D}_2'}[h_1(x) \neq c(x) \wedge h_2(x) = c(x)] = 1/2 - \gamma_2 \geq 1/16.$$

We now translate this probability back to $\mathcal{D}'$, but must be a little careful in doing so, since the assumption that $\beta_1 = error_{\mathcal{D}'}(h_1)$ was the maximum value β was valid only for the purposes of maximizing γ_2. Thus by Equation 4.1, we write

$$\mathbf{Pr}_{x \in \mathcal{D}'}[h_1(x) \neq c(x) \wedge h_2(x) = c(x)] \geq (2\beta_1)(1/16) = \beta_1/8.$$

Under our assumption $\beta_1 \geq \alpha/3$, we finally obtain

$$\mathbf{Pr}_{x \in \mathcal{D}'}[h_1(x) \neq c(x) \wedge h_2(x) = c(x)] \geq \alpha/24$$

as promised. □(Lemma 4.3)

The following lemma summarizes our bounds on the number of expected calls to $EX(c, \mathcal{D}')$ needed to simulate the two filtered distributions.

Lemma 4.4 *Let a node of the execution tree be labeled by oracle $EX(c, \mathcal{D}')$ and error bound α. Then the expected number of examples that must be drawn from $EX(c, \mathcal{D}')$ to simulate a single draw from $EX(c, \mathcal{D}'_2)$ is at most $3/\alpha$, and the expected number of examples that must be drawn from $EX(c, \mathcal{D}')$ to simulate a single draw from $EX(c, \mathcal{D}'_3)$ is at most $24/\alpha$.*

We now invoke the bounds of Lemma 4.4 repeatedly in order to obtain a bound on how many calls to the root oracle $EX(c, \mathcal{D})$ are required to simulate a single draw from an oracle labeling a node at depth i in the execution tree. To simulate one call to an oracle at the first level of recursion (the children of the root, where the error bound is $g^{-1}(\epsilon)$), we need at most $24/\epsilon$ calls to $EX(c, \mathcal{D})$; to simulate one call to an oracle at the second level of recursion (where the error bound is $g^{-1}(g^{-1}(\epsilon)) = g^{-2}(\epsilon)$), we need at most $24/g^{-1}(\epsilon)$ calls to the first level of recursion, each of which in turn require 24ϵ calls to $EX(c, \mathcal{D})$. By similar reasoning, a call to an oracle at the ith level of the execution tree requires at most

$$\left(\frac{24}{\epsilon}\right)\left(\frac{24}{g^{-1}(\epsilon)}\right)\cdots\left(\frac{24}{g^{-(i-1)}(\epsilon)}\right)$$

calls to $EX(c, \mathcal{D})$ in expectation.

To bound this expression, recall that $g(\beta) = 3\beta^2 - 2\beta^3 \leq 3\beta^2$. We first prove by induction on i that $g^{-i}(\epsilon) \geq (\epsilon^{1/2^i})/3$. For the base case, since $g(x) \leq 3x^2$ we have $x \geq \sqrt{g(x)/3}$, or setting $x = g^{-1}(\epsilon)$, $g^{-1}(\epsilon) \geq \sqrt{\epsilon/3} \geq \sqrt{\epsilon}/3$ as desired. Inductively, we write $g^{-i}(\epsilon) = g^{-1}(g^{-(i-1)}(\epsilon)) \geq g^{-1}(\epsilon^{1/2^{i-1}}/3) \geq ((\epsilon^{1/2^{i-1}}/3)/3)^{1/2} = (\epsilon^{1/2^i})/3$. Therefore

$$\left(\frac{24}{\epsilon}\right)\left(\frac{24}{g^{-1}(\epsilon)}\right)\cdots\left(\frac{24}{g^{-(i-1)}(\epsilon)}\right) \leq \left(\frac{72}{\epsilon}\right)\left(\frac{72}{\epsilon^{1/2}}\right)\left(\frac{72}{\epsilon^{1/4}}\right)\cdots\left(\frac{72}{\epsilon^{1/2^{i-1}}}\right)$$

$$
\begin{aligned}
&= \frac{72^i}{\epsilon^{1+1/2+1/4+\cdots+1/2^{i-1}}} \\
&= \frac{72^i \epsilon^{1/2^{i-1}}}{\epsilon^2} \\
&\leq \frac{72^i(3g^{-(i-1)}(\epsilon))}{\epsilon^2} \\
&\leq \frac{72^i \cdot 3(3g^{-i}(\epsilon)^2)}{\epsilon^2} \\
&= \frac{9 \cdot 72^i(g^{-i}(\epsilon))^2}{\epsilon^2}
\end{aligned}
$$

We have shown:

Lemma 4.5 *Let the root of the execution tree be labeled by the oracle $EX(c, \mathcal{D})$ and error bound ϵ. Then the expected number of examples that must be drawn from $EX(c, \mathcal{D})$ to simulate a single draw from an oracle labeling a node of the execution tree at depth i is bounded by*

$$\frac{9 \cdot 72^i(g^{-i}(\epsilon))^2}{\epsilon^2}.$$

4.3.6 Finishing Up

We are now almost ready to bound the sample complexity and running time of **Strong-Learn**$(\epsilon, EX(c, \mathcal{D}))$, but before doing so must address one final piece of unfinished business, which is the handling of the confidence parameter δ. Recall that to simplify the analysis so far, we have assumed that many steps in the algorithm that may in fact have some probability of failure (such as estimating the error of an intermediate hypothesis, or a call to the weak learning algorithm, or a recursive call with some error bound) are always successful. Now we must remove this assumption, and it is straightforward to do this by "dividing" up our desired global confidence parameter δ into many small pieces, one for each possible failure. Recall that we have already given a bound on the recursion depth $B = B(\epsilon, p(n, size(c))$, thus giving a bound of at most 3^B nodes in the execution tree. (Remember that B bounds the recursion depth under the assumption that there are no failures, and therefore does not depend on δ). Now for any node in the execution tree, there are five local steps that may result in a failure we wish to guard against: the estimation of the errors of $error_{\mathcal{D}'}(h_1)$ and $error_{\mathcal{D}'_2}(h_2)$ to sufficient accuracy, and the three recursive calls. Thus, each of these steps will be carried out using confidence parameter $\delta' = \delta/(5 \cdot 3^B)$, resulting in overall probability of failure at most δ.

Throughout the remainder of this section, we will use the shorthand notation $B = B(\epsilon, p(n, size(c)))$ for the execution tree depth which we have already bounded, and let $\delta' = \delta/(5 \cdot 3^B)$. Note that by Lemma 4.2, any quantity that is bounded by an exponential function of B is bounded by $O(p(n, size(c)) \log(1/\epsilon))$.

Let $T(\epsilon, \delta, n, size(c))$ be the expected running time of the invocation of algorithm **Strong-Learn**$(\epsilon, \delta, EX(c, \mathcal{D}))$, let $M(\epsilon, \delta, n, size(c))$ be the expected total number of examples drawn from the oracle $EX(c, \mathcal{D})$ by this invocation, and let $U(\epsilon, \delta, n, size(c))$ be the time required to evaluate the final hypothesis returned by this invocation. Let $t(\delta, n, size(c))$, $m(\delta, n, size(c))$ and $u(\delta, n, size(c))$ be the analogous quantities for the invocation of the weak learning algorithm $L(\delta, EX(c, \mathcal{D}))$.

We start by bounding the time required to evaluate a hypothesis returned by **Strong-Learn**$(\epsilon, \delta, EX(c, \mathcal{D}))$.

Lemma 4.6

$$U(\epsilon, \delta, n, size(c)) = O(3^B \cdot u(\delta', n, size(c)))$$

which is polynomial in $1/\epsilon$, $1/\delta$, n, *and* $size(c)$.

Proof: The hypothesis returned by **Strong-Learn** exactly mirrors the structure of the execution tree: it is a ternary tree of height B, whose internal nodes are majority functions, and whose leaves are hypotheses returned by calls to the weak learning algorithm L. The total time taken to evaluate all the hypotheses at the leaves is $O(3^B \cdot u(\delta', n, size(c)))$. This is because there are at most 3^B leaves, and each leaf of the execution tree is an invocation of L with confidence parameter δ'. The time taken to evaluate the majorities at the internal nodes is dominated by this bound. □(Lemma 4.6)

We now bound the expected number of examples $M(\epsilon, \delta, n, size(c))$.

Lemma 4.7

$$M(\epsilon, \delta, n, size(c)) = O\left(\frac{216^B}{\epsilon^2}\left(m(\delta', n, size(c)) + p^2(n, size(c)) \log \frac{1}{\delta'}\right)\right)$$

which is polynomial in $1/\epsilon$, $1/\delta$, n, *and* $size(c)$.

Proof: **Strong-Learn**$(\epsilon, \delta, EX(c, \mathcal{D}))$ invokes the weak learning algorithm L at most 3^B times. Each such invocation requires $m(\delta', n, size(c))$ filtered examples, since each time L is called it is with confidence parameter δ'. We have already shown in Lemma 4.5 that to obtain one filtered example at depth B in the execution tree, **Strong-Learn** is expected to draw at most $9 \cdot 72^B(g^{-B}(\epsilon))^2/\epsilon^2 \leq 9 \cdot 72^B/\epsilon^2$ examples from the oracle $EX(c, \mathcal{D})$. Therefore $3^B(9 \cdot 72^B/\epsilon^2)m(\delta', n, size(c)) = O((216^B/\epsilon^2)m(\delta', n, size(c)))$ examples suffice to implement the filtered oracles at the execution tree leaves.

In addition, **Strong-Learn** draws samples at each node of the execution tree in order to estimate the quality of the hypotheses h_1 and h_2. Recall that at the ith level of the execution tree, the desired error bound is $\alpha = g^{-i}(\epsilon)$, and the desired error bound for the $i+1$st level is $\beta = g^{-1}(\alpha)$. The estimate for the error of h_1 at level i must have additive error bounded by $\alpha/3$, and the estimate for the error of h_2 at level i must have additive error bounded by $\tau = ((1-2\beta)/8) \cdot \alpha$. Since $\tau < \alpha/3$, the number of examples required to accurately estimate the error of h_2

dominates the sample size required to estimate the error of h_1, so we will limit our attention to this dominating term.

Note that $\tau \geq (1/(4p(n, size(c))))\alpha$ because

$$1 - 2\beta \geq \left(1 - 2\left(\frac{1}{2} - \frac{1}{p(n, size(c))}\right)\right) = \frac{2}{p(n, size(c))}$$

due to the base case of the recursion. Using Lemma 4.5 and Chernoff bounds, the number of filtered examples required for the tests at level i is thus

$$O\left(\left(\frac{p^2(n, size(c))}{(g^{-i}(\epsilon))^2} \log \frac{1}{\delta'}\right) \frac{72^i (g^{-i}(\epsilon))^2}{\epsilon^2}\right)$$

which is

$$O\left(\left(p^2(n, size(c)) \log \frac{1}{\delta'}\right) \frac{72^B}{\epsilon^2}\right)$$

since $i \leq B$. Since the number of internal nodes is bounded by 3^B, this gives an overall bound for the internal node tests of

$$O\left(\left(p^2(n, size(c)) \log \frac{1}{\delta'}\right) \frac{216^B}{\epsilon^2}\right).$$

Combining the bounds for the leaves and internal nodes gives the stated overall bound. □(Lemma 4.7)

We now bound $T(\epsilon, \delta, n, size(c))$, the expected running time.

Lemma 4.8

$$T(\epsilon, \delta) = O\left(\frac{648^B}{\epsilon^2}\left(m(\delta', n, size(c)) + \left(p^2(n, size(c)) \log \frac{1}{\delta'}\right) B \cdot u(\delta', n, size(c))\right) + 3^B \cdot t(\delta', n, size(c))\right)$$

which is polynomial in $1/\epsilon$*,* $1/\delta$*,* n*, and* $size(c)$*.*

Proof: The running time of **Strong-Learn** may be decomposed into two parts. First, there is time spent at the leaves of the recursion, in the invocations to the weak learning algorithm L. The time for each call to L is at most $t(\delta', n, size(c))$, and the number of such calls is at most 3^B, giving a total of $3^B \cdot t(\delta', n, size(c))$.

The remainder of the running time of can be ascribed to the examples drawn from $EX(c, \mathcal{D})$ and their subsequent passage from the root down to a node in the execution tree. In its passage down the execution tree, an instance may be given as input to at most B hypotheses (one per node passed), where each such hypothesis is being used to implement a filtered distribution.

From the fact that evaluating a hypothesis takes at most time $O(3^B \cdot u(\delta', n, size(c)))$ (the most expensive hypothesis to evaluate is the final root hypothesis), and the fact that the expected total number of examples drawn is

$$O\left(\frac{216^B}{\epsilon^2}\left(m(\delta', n, size(c)) + p^2(n, size(c)) \log \frac{1}{\delta'}\right)\right)$$

we obtain the stated total time. □(Lemma 4.8)

Now it is a simple exercise to show that the polynomial bounds on the expected values of the sample size and running time can instead be expressed as polynomial bounds that hold with high probability, by allotting a fraction of the confidence parameter δ to the small probability that the sample size or running time are excessively large. Combining Lemmas 4.6, 4.7 and 4.8, we obtain our main result:

Theorem 4.9 *Let C be any concept class and $\mathcal{H}$ any hypothesis class. Then if C is efficiently weakly PAC learnable using $\mathcal{H}$, C is efficiently strongly PAC learnable using a hypothesis class of ternary majority trees with leaves from $\mathcal{H}$.*

4.4 Exercises

4.1. Use our transformation of a weak PAC learning algorithm to a strong PAC learning algorithm to show that for any concept class $\mathcal{C}$, if $\mathcal{C}$ is efficiently PAC learnable, then it is efficiently PAC learnable by an algorithm that:

- requires a sample of size at most $1/\epsilon^2 \cdot p_1(n, size(c), \log(1/\epsilon), \log(1/\delta))$;
- runs in time at most $1/\epsilon^2 \cdot p_2(n, size(c), \log(1/\epsilon), \log(1/\delta))$;
- uses memory at most $p_3(n, size(c), \log(1/\epsilon), \log(1/\delta))$;
- outputs hypotheses of size at most $p_4(n, size(c), \log(1/\epsilon))$ that can be evaluated in time at most $p_5(n, size(c), \log(1/\epsilon))$

for some fixed polynomials p_1, p_2, p_3, p_4, p_5.

4.2. Use our transformation of a weak PAC learning algorithm to a strong PAC learning algorithm to show that for any concept class $\mathcal{C}$, if $\mathcal{C}$ is efficiently PAC learnable, then there is an efficient algorithm that, given $0 < \delta \leq 1$ and a sample S of m examples of $c \in \mathcal{C}_n$, outputs with probability at least $1 - \delta$ a hypothesis h that is consistent with S such that $size(h) \leq p(n, size(c), \log m)$ for some polynomial p.

4.3. We say that an algorithm L in the PAC setting is a **group PAC learning algorithm** for $\mathcal{C}$ if there exists a polynomial $p(\cdot,\cdot)$ such for any target concept $c \in \mathcal{C}_n$ and any distribution, when L is given access to $EX(c, \mathcal{D})$ it outputs a hypothesis $h : X^{p(n,size(c))} \rightarrow \{0, 1\}$ that with probability $1 - \delta$ satisfies

$$\mathbf{Pr}_{S \in \mathcal{D}^{p(n,size(c))}}[h(S) = 0 | (\forall x \in S) c(x) = 1] \leq \epsilon$$

and

$$\mathbf{Pr}_{S \in \mathcal{D}^{p(n,size(c))}}[h(S) = 1 | (\forall x \in S) c(x) = 0] \leq \epsilon$$

Thus, the hypothesis output by L must be given a large (but still only polynomial size) collection of random examples that are either all positive or all negative, in which case it accurately determines which is the case.

Prove that for any concept class $\mathcal{C}$, $\mathcal{C}$ is efficiently group PAC learnable if and only if it is efficiently weakly PAC learnable.

4.5 Bibliographic Notes

The equivalence of weak and strong learning was proved by Schapire [84, 85], and the proof given in this chapter is due to him. Exercises 4.1 and 4.2 are also due to Schapire, and his paper explores many other fascinating consequences of the construction. Alternative boosting methods have been given by Freund [35, 36]. In Freund's construction, the strong learning algorithm's hypothesis is simply a majority of many hypotheses obtained from filtered runs of the weak learning algorithm.

Experimental results on neural network learning based on boosting ideas are reported by Drucker, Schapire and Simard [30]. Goldman, Kearns and Schapire [42] examine the sample size required for weak learning, showing that it can be considerably less than for strong learning in some cases. Helmbold and Warumth [52] study various properties of the weak learning model and its relationship to sample compression and Occam learning. Boneh and Lipton [24] examine conditions under which boosting can be performed with respect to special distributions, and Decatur and Aslam [12] show that weak learning is still equivalent to strong learning in a restricted version of the PAC model known as statistical query learning, which will be the focus of our study in Chapter 5. Exercise 4.3 is from a paper by Kearns and Valiant [60], which also first introduced the notion of weak learning.