6.858/18.428 Machine Learning	Lecture 13: October 26, 1994
Lecturer: Ron Rivest	*Scribe: Rob Miller*

Outline

- Finish the proof that weak learnability implies strong learnability.

13.1 Weak Learnability Implies Strong Learnability

Today we finish the proof that weak learnability implies strong learnability, a theorem due originally to Shapire [3]. This is a rather surprising result; essentially it says that if we can find an efficient learning algorithm that guarantees only a fixed accuracy ϵ_0 ($\epsilon_0 \leq \frac{1}{2} - \frac{1}{p(n)}$) with only a fixed confidence $1 - \delta_0$, then we can turn it into a general PAC algorithm for arbitrary ϵ and δ.

Last time, we saw a procedure for boosting the confidence of an algorithm, from a fixed δ_0 to arbitrary δ. We also saw how an algorithm's accuracy can be boosted by running the algorithm three times, on three different distributions: the original distribution $\mathcal{D}_1$; a distribution $\mathcal{D}_2$ which is drawn half from examples where h_1 is correct, and half from where h_1 is incorrect; and a distribution $\mathcal{D}_3$, which consists only of examples where h_1 and h_2 disagree. Taking the majority of h_1, h_2, and h_3 yields a hypothesis with error rate less than $g(\epsilon) = 3\epsilon^2 - 2\epsilon^3$. This one-shot accuracy-boosting procedure has been applied at Bell Labs in teaching a neural net to recognize characters, with some success in reducing its error rate. So hypothesis boosting is not only interesting in theory, but useful in practice. [1]

Today we will show how to recursively apply the accuracy-boosting procedure to achieve arbitrary accuracy, in time polynomial in $\frac{1}{\epsilon}$ and n.

13.2 Recursive Hypothesis-Boosting Algorithm

Suppose L is a learning algorithm which produces hypotheses with guaranteed error rate less than $\frac{1}{2} - \frac{1}{p(n)}$ (for some polynomial $p(n)$) with probability at least $1 - \delta$. We will define a recursive algorithm $SL(\alpha, EX(c, \mathcal{D}'))$, which uses the example oracle $EX(c, \mathcal{D}')$ to find a hypothesis with error rate less than α, for any $\alpha > 0$.

Algorithm $SL(\alpha, EX(c, \mathcal{D}'))$

1. If $\alpha > \frac{1}{2} - \frac{1}{p(n)}$, then return $L(EX(c, \mathcal{D}'))$.
2. $\beta \leftarrow g^{-1}(\alpha)$.
3. $h_1 \leftarrow SL(\beta, EX(c, \mathcal{D}'))$.
4. Create $EX(c, \mathcal{D}'_2)$ by filtering $EX(c, \mathcal{D}')$ so that half of its examples are drawn subject to the constraint $h_1 = c$, and half are subject to $h_1 \neq c$.
5. $h_2 \leftarrow SL(\beta, EX(c, \mathcal{D}'_2))$.
6. Create $EX(c, \mathcal{D}'_3)$ by filtering $EX(c, \mathcal{D}')$ subject to $h_1 \neq h_2$.
7. $h_3 \leftarrow SL(\beta, EX(c, \mathcal{D}'_3))$.
8. Return $h = majority(h_1, h_2, h_3)$.

To achieve an arbitrary input accuracy of ϵ on the original distribution $\mathcal{D}$, we run $SL(\epsilon, EX(c, \mathcal{D}))$. The execution of SL performs a recursive computation like that shown in Figure 13.1. The error rate required of the root node is ϵ, but thanks to the hypothesis-boosting at each node, its children have a weaker error-rate requirement, $g^{-1}(\epsilon) > \epsilon$ (see Figure 13.2). Intuitively, SL finds a sufficiently accurate hypothesis for the root by searching depth-first down the tree until the required error rate is greater than $\frac{1}{2} - \frac{1}{p(n)}$, at which point L itself can produce acceptable hypotheses. The hypotheses from the leaves are then combined by majority vote at each node to produce the final hypothesis.

We care about the depth B of this tree, since it will play a major role in the bounds of the running time and sample size of SL.

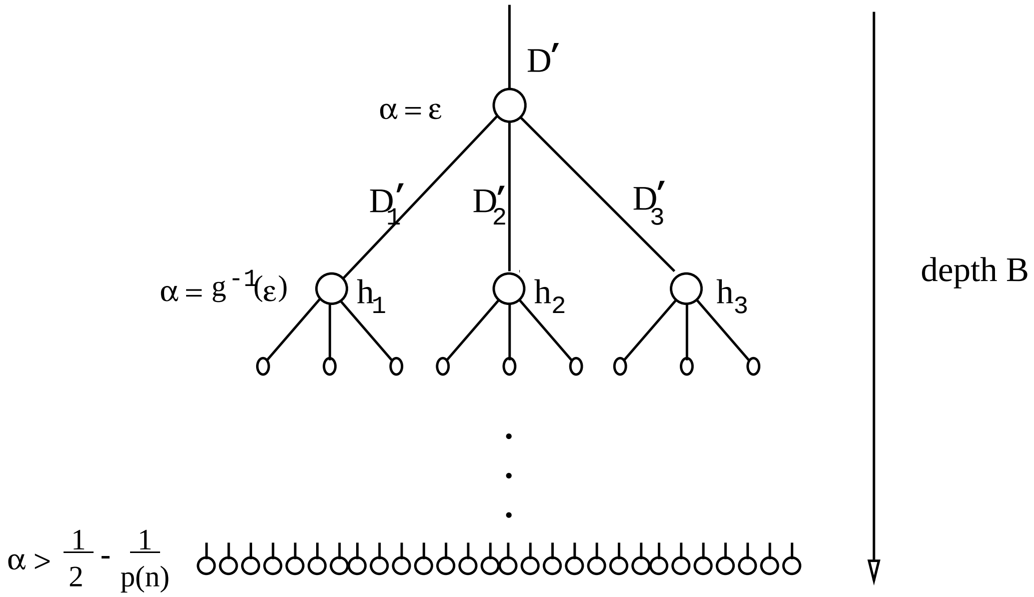

Figure 13.1: The tree of recursive calls made by SL. The recursion bottoms out when the required error rate α is greater than $\frac{1}{2} - \frac{1}{p(n)}$, for which the weak learning algorithm L is able to generate a satisfactory hypothesis.

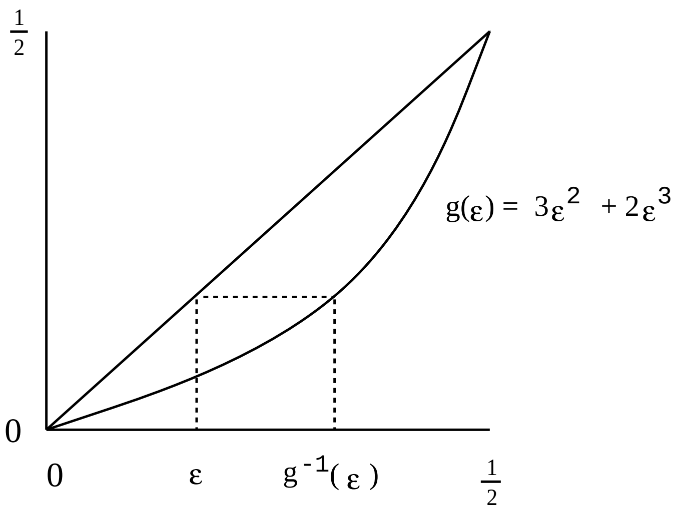

Figure 13.2: The error rate requirement becomes less stringent for nodes lower in the tree. Although the root is required to output a hypothesis with error rate less than ϵ, its children need to achieve an error rate of $g^{-1}(\epsilon)$, which the graph above shows is greater than ϵ.

Lemma 1 $B = O(\log p(n) + \log\log \frac{1}{\epsilon})$.

Proof: Let β_i be the required error rate at depth i in the tree, where the root has depth 0. So $\beta_0 = \epsilon$, and β of a node is related to the β of its parent by $\beta_{i+1} = g^{-1}(\beta_i)$.

We will break the analysis into two parts, each giving one term of the result $O(\log p(n) + \log\log \frac{1}{\epsilon})$. First, consider how many levels of the tree are sufficient to get β up from ϵ to $\frac{1}{4}$. We will do the analysis in the reverse direction, considering how β decreases as we go up the tree from $\frac{1}{4}$. In one step up the tree, β decreases to $g(\beta) = 3\beta^2 - 2\beta^3 \leq 3\beta^2$. In k steps, it is less than $\frac{1}{3}(3\beta)^{2^k}$. If initially $\beta \leq \frac{1}{4}$, then after k steps it falls below $\frac{1}{3}(\frac{3}{4})^{2^k}$. Setting this less than ϵ and solving for k gives $k \geq c_0 \log\log \frac{1}{\epsilon}$, for some constant $c_0 > 0$. So $O(\log\log \frac{1}{\epsilon})$ levels are sufficient to get β up from ϵ to $\frac{1}{4}$.

For the second part, we consider how quickly β approaches $\frac{1}{2}$, as we go down the tree from $\beta \geq \frac{1}{4}$ to $\beta \geq \frac{1}{2} - \frac{1}{p(n)}$. We want to compare the deviation $\frac{1}{2} - \beta$ with the deviation for its parent, $\frac{1}{2} - g(\beta)$.

$$\frac{1}{2} - g(\beta) = \frac{1}{2} - \left(3\beta^2 - 2\beta^3\right) = \left(\frac{1}{2} - \beta\right)\left(1 + 2\beta - 2\beta^2\right)$$

When $\frac{1}{4} \leq \beta \leq \frac{1}{2}$, then $(1+2\beta-2\beta^2) \geq \frac{11}{8}$. (This is just its minimum on the specified interval, found by basic calculus.) So the distance from β to $\frac{1}{2}$ decreases by a factor of at least $\frac{11}{8}$ for every level down the tree, so

$$k \geq \log_{\frac{11}{8}}\left(\frac{1/4}{1/p(n)}\right) = O(\log p(n))$$

■

The size of the tree is $3^B = O(p(n) \log \frac{1}{\epsilon})$.

13.3 Filtering Efficiency

An issue that has been glossed over in the discussion to this point is the efficiency of the filtering process that creates $\mathcal{D}'_2$ and $\mathcal{D}'_3$. After all, the running time of SL is proportional not only to the size of the computation tree, but also to the number of examples that must be drawn from the original distribution $\mathcal{D}$ in order to satisfy draws on the filtered distributions. In fact, in the worst case, the algorithm SL as given will fail to terminate! Suppose h_1 at some node is a perfect hypothesis, with zero error rate. The filter for $\mathcal{D}'_2$ will run forever, looking for examples where $h_1 \neq c$. A similar problem occurs if $h_1 = h_2$, since the filter for $\mathcal{D}'_3$ will never find an example where the two hypotheses disagree. We need to patch the SL algorithm so that it avoids these pitfalls.

The basic idea is to test h_1 and h_2 before using them to filter examples. If either of the hypotheses has a small estimated error rate, we return it immediately; otherwise we can show that the filtering process will terminate in polynomial time with high probability. We will consider h_1 and h_2 in turn.

For h_1, the dangerous situation occurs when $error_{\mathcal{D}'}(h_1)$ is close to zero (where $error_{\mathcal{D}'}$ is measured with respect to $\mathcal{D}'$). So we calculate an estimate $\hat{e}_1$ of $error_{\mathcal{D}'}(h_1)$ by drawing enough examples from $EX(c, \mathcal{D})$ to make the estimate accurate within $\pm\frac{\alpha}{3}$ with high probability.[1] If $\hat{e}_1 \leq \frac{2\alpha}{3}$, then with high probability, the true error rate of h_1 is less than α, so we can just return it. Otherwise, we can be confident that the true error rate $error_{\mathcal{D}'}(h_1) \geq \frac{\alpha}{3}$, so generating an example for $\mathcal{D}'_2$ will take at most $\frac{3}{\alpha}$ examples from $\mathcal{D}'$ (on average).

To test h_2, calculate an estimate $\hat{e}_2$ of $error_{\mathcal{D}'}(h_2)$ to within $\pm\tau$, where $\tau = \frac{\alpha}{8}(1 - 2g^{-1}(\alpha))$. If $\hat{e}_2 \leq \alpha - \tau$, then h_2 is a sufficiently good hypothesis, so we can return it. Otherwise $error_{\mathcal{D}'}(h_2) \geq \alpha - 2\tau$, so we can apply the following lemma.

Lemma 2 *If $error_{\mathcal{D}'}(h_1) \geq \frac{\alpha}{3}$ and $error_{\mathcal{D}'}(h_2) \geq \alpha - 2\tau$, then $Pr_{x \in \mathcal{D}'}(h_1(x) \neq h_2(x)) \geq \frac{\alpha}{24}$. (For a proof of this result, see Lemma 4.3 in [2].)*

Lemma 2 implies that generating a $\mathcal{D}'_3$ example requires at most $\frac{24}{\alpha}$ draws from $\mathcal{D}'$ on the average. We are interested in finding an upper bound on the filtering efficiency, which is the expected number of draws on the root oracle $\mathcal{D}$ for each draw by a node lower in the tree. The filtering overhead introduced by a single node is at most $\frac{24}{\alpha}$, so the filtering efficiency at level i is at most

$$\frac{24}{\epsilon}\,\frac{24}{g^{-1}(\epsilon)}\,\frac{24}{g^{-2}(\epsilon)}\cdots\frac{24}{g^{-i}(\epsilon)}$$

.

Lemma 3 $g^{-i}(\epsilon) \geq \frac{1}{3}\epsilon^{2^{-i}}$.

[1] By the additive Chernoff bound, the number of examples required will be polynomial in $\frac{3}{\alpha}$ and $log\frac{1}{\delta'}$, where $1 - \delta'$ is the desired confidence for the estimate. A sufficient value for δ' will be derived in the next section.

Proof: Proof by induction. For the base case $i = 1$, we have $g(x) \leq 3x^2$, so $x \geq \sqrt{\frac{g(x)}{3}}$. Substituting $x = g^{-1}(\epsilon)$ gives $g^{-1}(\epsilon) \geq \sqrt{\frac{\epsilon}{3}} > \frac{1}{3}\sqrt{\epsilon}$ $(**)$.

For the induction step, suppose the claim is true for $i - 1$. Then

$$\begin{aligned}
g^{-i}(\epsilon) &= g^{-1}\left(g^{-(i-1)}(\epsilon)\right) \\
&\geq g^{-1}\left(\frac{1}{3}\epsilon^{2^{-(i-1)}}\right) \\
&\geq \frac{1}{\sqrt{3}}\left(\frac{1}{3}\epsilon^{2^{-(i-1)}}\right)^{\frac{1}{2}} \qquad \text{(using } **\text{)} \\
&\geq \frac{1}{3}\epsilon^{2^{-i}}
\end{aligned}$$

■

Applying Lemma 3 to the upper bound on filtering efficiency gives

$$\begin{aligned}
\frac{24}{\epsilon}\frac{24}{g^{-1}(\epsilon)}\cdots\frac{24}{g^{-i}(\epsilon)} &\leq \frac{72}{\epsilon}\frac{72}{\epsilon^{1/2}}\frac{72}{\epsilon^{1/4}}\cdots\frac{72}{\epsilon^{2^{-(i-1)}}} \\
&= \frac{72^i}{\epsilon^{1+2^{-1}+2^{-2}+\cdots+2^{-(i-1)}}} \\
&= \frac{72^i\epsilon^{2^{-(i-1)}}}{\epsilon^2} \\
&\leq \frac{72^i(3g^{-(i-1)}(\epsilon))}{\epsilon^2} \\
&\leq \frac{72^i \cdot 3g\left(g^{-i}(\epsilon)\right)}{\epsilon^2} \\
&\leq \frac{72^i \cdot 3\left(3\left(g^{-i}\left(\epsilon\right)\right)^2\right)}{\epsilon^2} \\
&\leq \frac{9 \cdot 72^i\left(g^{-i}(\epsilon)\right)^2}{\epsilon^2}
\end{aligned}$$

Since i is bounded by $B = O(\log p(n) + \log\log\frac{1}{\epsilon})$, and $g^{-i}(\epsilon) < 1$, the filtering efficiency is polynomial in $\frac{1}{\epsilon}$ and $p(n)$.

The revised algorithm is shown below.

Revised Algorithm $SL(\alpha, EX(c, \mathcal{D}'))$

1. If $\alpha > \frac{1}{2} - \frac{1}{p(n)}$, then return $L(EX(c, \mathcal{D}'))$.
2. $\beta \leftarrow g^{-1}(\alpha)$.
3. $h_1 \leftarrow SL(\beta, EX(c, \mathcal{D}'))$.
4. $\hat{e}_1 \leftarrow$ an estimation of $error_{\mathcal{D}'}(h_1)$ to within $\pm\frac{\alpha}{3}$.
   If $\hat{e}_1 \leq \frac{2\alpha}{3}$, then return h_1.
5. Create $EX(c, \mathcal{D}'_2)$ by filtering $EX(c, \mathcal{D}')$ so that half of its examples are drawn subject to the constraint $h_1 = c$, and half are subject to $h_1 \neq c$.
6. $h_2 \leftarrow SL(\beta, EX(c, \mathcal{D}'_2))$.
7. $\hat{e}_2 \leftarrow$ an estimation of $error_{\mathcal{D}'}(h_2)$ to within $\pm\tau$.
   If $\hat{e}_2 \leq \alpha - \tau$, then return h_2.
8. Create $EX(c, \mathcal{D}'_3)$ by filtering $EX(c, \mathcal{D}')$ subject to $h_1 \neq h_2$.
9. $h_3 \leftarrow SL(\beta, EX(c, \mathcal{D}'_3))$.
10. Return $h = majority(h_1, h_2, h_3)$.

13.4 Confidence Analysis

Repeatedly throughout the construction of SL we have said "with high probability," without specifying the actual probability required. We turn now to this question. In the end, SL must output an accurate hypothesis with a chance of failure no greater than δ. Each node in the recursive computation of SL can make an mistake in any of five ways: in estimating the hypotheses h_1, h_2, or h_3, or in estimating the error rates $\hat{e}_1$ and $\hat{e}_2$. The simplest solution divides δ among all of the possible points of failure, requiring each to have chance of failure less than $\delta' = \frac{\delta}{5 \cdot 3^B}$. Then applying the union bound will limit the probability that the entire algorithm fails to at most δ.

Note that we require L to accept an arbitrary confidence parameter δ, so if we start with a weak learning algorithm L with fixed accuracy $\epsilon_0 < \frac{1}{2}$ and fixed confidence δ_0, we need to boost the confidence of its hypotheses. Unfortunately, the confidence-boosting procedure we saw last lecture delivers less-accurate hypotheses L (error rate at most $2\epsilon_0$), which works only if $\epsilon_0 < \frac{1}{4}$. It is possible to tweak the confidence-boosting procedure so that it outputs hypotheses with error rates at most $\epsilon_0 + \gamma$, where γ can be arbitrary small. (For a proof, see section 4.2 in [2].) This makes the confidence-boosted L a candidate for the accuracy-boosting procedure described here.

13.5 Overall Performance of SL

To finish up, we present some results on the running time, sample size, and hypothesis-evaluation time of SL, as functions of the input parameters and the corresponding quantities for L. Let $t(\delta, n)$ be the running time of L, $m(\delta, n)$ be the sample size required by L, and $u(\delta, n)$ be the time to evaluate a hypothesis returned by L.

Lemma 4 *The time to evaluate a hypothesis returned by SL is*

$$U = O\left(3^B u(\delta', n)\right)$$

Proof: U is comprised of the time to evaluate the hypotheses at each of the $O(3^B)$ leaves of the tree, plus a majority operation at each of the $O(3^B)$ internal nodes, so $U = O(3^B u(\delta', n) + 3^B) = O(3^B u(\delta', n))$. ■

Lemma 5 *The expected sample size drawn by SL from its root oracle $EX(c, \mathcal{D})$ is*

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

Proof: Each leaf of the tree runs L, which draws $m(\delta', n)$ examples. Multiplying by the filtering ratio $\frac{9 \cdot 72^B (g^{-i}(\epsilon))^2}{\epsilon^2} \leq \frac{9 \cdot 72^B}{\epsilon^2}$ and the number of leaves 3^B gives the first term of the upper bound on M.

Each internal node must estimate the error rates of h_1 and h_2, with a confidence of $1 - \delta$. Actually, the estimation of $\hat{e}_2$ must be more accurate than $\hat{e}_1$ (within $\pm\tau = \frac{\alpha}{8}(1 - 2g^{-1}(\alpha))$, which is tighter than the $\pm\frac{\alpha}{3}$ allowed for $\hat{e}_2$), so the sample size required by $\hat{e}_2$ dominates. Using the additive Chernoff bound, this sample size is $O(\frac{1}{\tau^2}\log\frac{1}{\delta'})$. Since

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

we have $\tau \geq \frac{\alpha}{4p(n)}$. For a node at level i, $\alpha = g^{-i}(\epsilon)$, so the sample size drawn by the node is $O(\frac{p^2(n)}{g^{-i}} \log \frac{1}{\delta'})$. Multiplying by the filtering ratio and the number of internal nodes gives the second term of M. ■

Lemma 6 *The running time of SL is*

$$T = O(BUM + 3^B t(\delta', n))$$

Proof: The second term in T is simply the running time of L in each leaf, multiplied by the number of leaves. The first term is an upper bound on the cost of filtering examples. As each of the M examples drawn from the root distribution travels down the tree, it must be tested by h_1 and/or h_2 at each node on the path to determine if it should be passed or rejected. Since the path has length at most B, and each hypothesis takes at most U time to evaluate, the resulting cost is $O(BU)$ per example, for a total cost of BUM. ■

It is easy to see that if t, u, and m are polynomial in n, $\frac{1}{\epsilon}$, and $\log \frac{1}{\delta}$, then so are T, U, and M. So we have found an efficient strong PAC algorithm for $\mathcal{C}$.

References

[1] H. Drucker, R. Shapire, and P. Simard. Improving performance in neural networks using a boosting algorithm. In S.J. Hanson, J.D. Cowan, and C.L. Giles, editors, *Advances in Neural Information Processing Systems*, pages 42-49. Morgan Kaufmann, San Mateo, CA, 1992.

[2] M. Kearns and U. Vazirani. *An Introduction to Computational Learning Theory.* MIT Press, 1994.

[3] R.E. Shapire. The strength of weak learnability. *Machine Learning*, 5(2):197-227, 1990.