10 Boosting

Boosting is an algorithmic paradigm that grew out of a theoretical question and became a very practical machine learning tool. The boosting approach uses a generalization of linear predictors to address two major issues that have been raised earlier in the book. The first is the bias-complexity tradeoff. We have seen (in Chapter 5) that the error of an ERM learner can be decomposed into a sum of *approximation error* and *estimation error*. The more expressive the hypothesis class the learner is searching over, the smaller the approximation error is, but the larger the estimation error becomes. A learner is thus faced with the problem of picking a good tradeoff between these two considerations. The boosting paradigm allows the learner to have smooth control over this tradeoff. The learning starts with a basic class (that might have a large approximation error), and as it progresses the class that the predictor may belong to grows richer.

The second issue that boosting addresses is the computational complexity of learning. As seen in Chapter 8, for many interesting concept classes the task of finding an ERM hypothesis may be computationally infeasible. A boosting algorithm amplifies the accuracy of *weak learners*. Intuitively, one can think of a weak learner as an algorithm that uses a simple "rule of thumb" to output a hypothesis that comes from an easy-to-learn hypothesis class and performs just slightly better than a random guess. When a weak learner can be implemented efficiently, boosting provides a tool for aggregating such weak hypotheses to approximate gradually good predictors for larger, and harder to learn, classes.

In this chapter we will describe and analyze a practically useful boosting algorithm, AdaBoost (a shorthand for Adaptive Boosting). The AdaBoost algorithm outputs a hypothesis that is a linear combination of simple hypotheses. In other words, AdaBoost relies on the family of hypothesis classes obtained by composing a linear predictor on top of simple classes. We will show that AdaBoost enables us to control the tradeoff between the approximation and estimation errors by varying a single parameter.

AdaBoost demonstrates a general theme, that will recur later in the book, of expanding the expressiveness of linear predictors by composing them on top of other functions. This will be elaborated in Section 10.3.

AdaBoost stemmed from the theoretical question of whether an efficient weak learner can be "boosted" into an efficient strong learner. This question was raised

by Kearns and Valiant in 1988 and solved in 1990 by Robert Schapire, then a graduate student at MIT. However, the proposed mechanism was not very practical. In 1995, Robert Schapire and Yoav Freund proposed the AdaBoost algorithm, which was the first truly practical implementation of boosting. This simple and elegant algorithm became hugely popular, and Freund and Schapire's work has been recognized by numerous awards.

Furthermore, boosting is a great example for the practical impact of learning theory. While boosting originated as a purely theoretical problem, it has led to popular and widely used algorithms. Indeed, as we shall demonstrate later in this chapter, AdaBoost has been successfully used for learning to detect faces in images.

10.1 Weak Learnability

Recall the definition of PAC learning given in Chapter 3: A hypothesis class, $\mathcal{H}$, is PAC learnable if there exist $m_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$ and a learning algorithm with the following property: For every $\epsilon, \delta \in (0,1)$, for every distribution $\mathcal{D}$ over $\mathcal{X}$, and for every labeling function $f : \mathcal{X} \to \{\pm 1\}$, if the realizable assumption holds with respect to $\mathcal{H}, \mathcal{D}, f$, then when running the learning algorithm on $m \geq m_{\mathcal{H}}(\epsilon, \delta)$ i.i.d. examples generated by $\mathcal{D}$ and labeled by f, the algorithm returns a hypothesis h such that, with probability of at least $1-\delta$, $L_{(\mathcal{D},f)}(h) \leq \epsilon$.

Furthermore, the fundamental theorem of learning theory (Theorem 6.8 in Chapter 6) characterizes the family of learnable classes and states that every PAC learnable class can be learned using any ERM algorithm. However, the definition of PAC learning and the fundamental theorem of learning theory ignores the computational aspect of learning. Indeed, as we have shown in Chapter 8, there are cases in which implementing the ERM rule is computationally hard (even in the realizable case).

However, perhaps we can trade computational hardness with the requirement for accuracy. Given a distribution $\mathcal{D}$ and a target labeling function f, maybe there exists an efficiently computable learning algorithm whose error is just slightly better than a random guess? This motivates the following definition.

DEFINITION 10.1 (γ-Weak-Learnability)

- A learning algorithm, A, is a γ-weak-learner for a class $\mathcal{H}$ if there exists a function $m_{\mathcal{H}} : (0,1) \to \mathbb{N}$ such that for every $\delta \in (0,1)$, for every distribution $\mathcal{D}$ over $\mathcal{X}$, and for every labeling function $f : \mathcal{X} \to \{\pm 1\}$, if the realizable assumption holds with respect to $\mathcal{H}, \mathcal{D}, f$, then when running the learning algorithm on $m \geq m_{\mathcal{H}}(\delta)$ i.i.d. examples generated by $\mathcal{D}$ and labeled by f, the algorithm returns a hypothesis h such that, with probability of at least $1-\delta$, $L_{(\mathcal{D},f)}(h) \leq 1/2 - \gamma$.
- A hypothesis class $\mathcal{H}$ is γ-weak-learnable if there exists a γ-weak-learner for that class.

This definition is almost identical to the definition of PAC learning, which here we will call *strong learning*, with one crucial difference: Strong learnability implies the ability to find an arbitrarily good classifier (with error rate at most ϵ for an arbitrarily small $\epsilon > 0$). In weak learnability, however, we only need to output a hypothesis whose error rate is at most $1/2 - \gamma$, namely, whose error rate is slightly better than what a random labeling would give us. The hope is that it may be easier to come up with efficient weak learners than with efficient (full) PAC learners.

The fundamental theorem of learning (Theorem 6.8) states that if a hypothesis class $\mathcal{H}$ has a VC dimension d, then the sample complexity of PAC learning $\mathcal{H}$ satisfies $m_{\mathcal{H}}(\epsilon, \delta) \geq C_1 \frac{d+\log(1/\delta)}{\epsilon}$, where C_1 is a constant. Applying this with $\epsilon = 1/2 - \gamma$ we immediately obtain that if $d = \infty$ then $\mathcal{H}$ is not γ-weak-learnable. This implies that from the statistical perspective (i.e., if we ignore computational complexity), weak learnability is also characterized by the VC dimension of $\mathcal{H}$ and therefore is just as hard as PAC (strong) learning. However, when we do consider computational complexity, the potential advantage of weak learning is that maybe there is an algorithm that satisfies the requirements of weak learning and can be implemented efficiently.

One possible approach is to take a "simple" hypothesis class, denoted B, and to apply ERM with respect to B as the weak learning algorithm. For this to work, we need that B will satisfy two requirements:

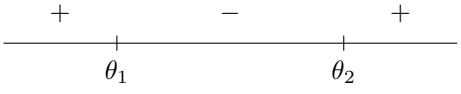
- ERM_B is efficiently implementable.
- For every sample that is labeled by some hypothesis from $\mathcal{H}$, any ERM_B hypothesis will have an error of at most $1/2 - \gamma$.

Then, the immediate question is whether we can boost an *efficient* weak learner into an *efficient* strong learner. In the next section we will show that this is indeed possible, but before that, let us show an example in which efficient weak learnability of a class $\mathcal{H}$ is possible using a base hypothesis class B.

Example 10.1 (Weak Learning of 3-Piece Classifiers Using Decision Stumps) Let $\mathcal{X} = \mathbb{R}$ and let $\mathcal{H}$ be the class of 3-piece classifiers, namely, $\mathcal{H} = \{h_{\theta_1,\theta_2,b} : \theta_1, \theta_2 \in \mathbb{R}, \theta_1 < \theta_2, b \in \{\pm 1\}\}$, where for every x,

$$h_{\theta_1,\theta_2,b}(x) = \begin{cases} +b & \text{if } x < \theta_1 \text{ or } x > \theta_2 \\ -b & \text{if } \theta_1 \leq x \leq \theta_2 \end{cases}$$

An example hypothesis (for $b = 1$) is illustrated as follows:



Let B be the class of Decision Stumps, that is, $B = \{x \mapsto \text{sign}(x - \theta) \cdot b : \ \theta \in \mathbb{R}, b \in \{\pm 1\}\}$. In the following we show that ERM_B is a γ-weak learner for $\mathcal{H}$, for $\gamma = 1/12$.

To see that, we first show that for every distribution that is consistent with $\mathcal{H}$, there exists a decision stump with $L_{\mathcal{D}}(h) \leq 1/3$. Indeed, just note that every classifier in $\mathcal{H}$ consists of three regions (two unbounded rays and a center interval) with alternate labels. For any pair of such regions, there exists a decision stump that agrees with the labeling of these two components. Note that for every distribution $\mathcal{D}$ over $\mathbb{R}$ and every partitioning of the line into three such regions, one of these regions must have $\mathcal{D}$-weight of at most $1/3$. Let $h \in \mathcal{H}$ be a zero error hypothesis. A decision stump that disagrees with h only on such a region has an error of at most $1/3$.

Finally, since the VC-dimension of decision stumps is 2, if the sample size is greater than $\Omega(\log(1/\delta)/\epsilon^2)$, then with probability of at least $1-\delta$, the ERM_B rule returns a hypothesis with an error of at most $1/3+\epsilon$. Setting $\epsilon = 1/12$ we obtain that the error of ERM_B is at most $1/3 + 1/12 = 1/2 - 1/12$.

We see that ERM_B is a γ-weak learner for $\mathcal{H}$. We next show how to implement the ERM rule efficiently for decision stumps.

10.1.1 Efficient Implementation of ERM for Decision Stumps

Let $\mathcal{X} = \mathbb{R}^d$ and consider the base hypothesis class of decision stumps over $\mathbb{R}^d$, namely,

$$\mathcal{H}_{\text{DS}} = \{\mathbf{x} \mapsto \text{sign}(\theta - x_i) \cdot b : \ \theta \in \mathbb{R}, i \in [d], b \in \{\pm 1\}\}.$$

For simplicity, assume that $b = 1$; that is, we focus on all the hypotheses in $\mathcal{H}_{\text{DS}}$ of the form $\text{sign}(\theta - x_i)$. Let $S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m))$ be a training set. We will show how to implement an ERM rule, namely, how to find a decision stump that minimizes $L_S(h)$. Furthermore, since in the next section we will show that AdaBoost requires finding a hypothesis with a small risk relative to some distribution over S, we will show here how to minimize such risk functions. Concretely, let $\mathbf{D}$ be a probability vector in $\mathbb{R}^m$ (that is, all elements of $\mathbf{D}$ are nonnegative and $\sum_i D_i = 1$). The weak learner we describe later receives $\mathbf{D}$ and S and outputs a decision stump $h : \mathcal{X} \to \mathcal{Y}$ that minimizes the risk w.r.t. $\mathbf{D}$,

$$L_{\mathbf{D}}(h) = \sum_{i=1}^{m} D_i \mathbb{1}_{[h(\mathbf{x}_i) \neq y_i]}.$$

Note that if $\mathbf{D} = (1/m, \ldots, 1/m)$ then $L_{\mathbf{D}}(h) = L_S(h)$.

Recall that each decision stump is parameterized by an index $j \in [d]$ and a threshold θ. Therefore, minimizing $L_{\mathbf{D}}(h)$ amounts to solving the problem

$$\min_{j \in [d]} \min_{\theta \in \mathbb{R}} \left(\sum_{i : y_i = 1} D_i \mathbb{1}_{[x_{i,j} > \theta]} + \sum_{i : y_i = -1} D_i \mathbb{1}_{[x_{i,j} \leq \theta]} \right). \tag{10.1}$$

Fix $j \in [d]$ and let us sort the examples so that $x_{1,j} \leq x_{2,j} \leq \ldots \leq x_{m,j}$. Define $\Theta_j = \{\frac{x_{i,j} + x_{i+1,j}}{2} : i \in [m-1]\} \cup \{(x_{1,j} - 1), (x_{m,j} + 1)\}$. Note that for any $\theta \in \mathbb{R}$ there exists $\theta' \in \Theta_j$ that yields the same predictions for the sample S as the

threshold θ. Therefore, instead of minimizing over $\theta \in \mathbb{R}$ we can minimize over $\theta \in \Theta_j$.

This already gives us an efficient procedure: Choose $j \in [d]$ and $\theta \in \Theta_j$ that minimize the objective value of Equation (10.1). For every j and $\theta \in \Theta_j$ we have to calculate a sum over m examples; therefore the runtime of this approach would be $O(dm^2)$. We next show a simple trick that enables us to minimize the objective in time $O(dm)$.

The observation is as follows. Suppose we have calculated the objective for $\theta \in (x_{i-1,j}, x_{i,j})$. Let $F(\theta)$ be the value of the objective. Then, when we consider $\theta' \in (x_{i,j}, x_{i+1,j})$ we have that

$$F(\theta') = F(\theta) - D_i \mathbb{1}_{[y_i=1]} + D_i \mathbb{1}_{[y_i=-1]} = F(\theta) - y_i D_i.$$

Therefore, we can calculate the objective at θ' in a constant time, given the objective at the previous threshold, θ. It follows that after a preprocessing step in which we sort the examples with respect to each coordinate, the minimization problem can be performed in time $O(dm)$. This yields the following pseudocode.

ERM for Decision Stumps

input:
  training set $S = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$
  distribution vector $\mathbf{D}$
goal: Find $j^\star, \theta^\star$ that solve Equation (10.1)
initialize: $F^\star = \infty$
for $j = 1, \ldots, d$
  sort S using the j'th coordinate, and denote
    $x_{1,j} \leq x_{2,j} \leq \cdots \leq x_{m,j} \leq x_{m+1,j} \stackrel{\text{def}}{=} x_{m,j} + 1$
  $F = \sum_{i:y_i=1} D_i$
  if $F < F^\star$
    $F^\star = F$, $\theta^\star = x_{1,j} - 1$, $j^\star = j$
  for $i = 1, \ldots, m$
    $F = F - y_i D_i$
    if $F < F^\star$ and $x_{i,j} \neq x_{i+1,j}$
      $F^\star = F$, $\theta^\star = \frac{1}{2}(x_{i,j} + x_{i+1,j})$, $j^\star = j$
output $j^\star, \theta^\star$

10.2 AdaBoost

AdaBoost (short for Adaptive Boosting) is an algorithm that has access to a weak learner and finds a hypothesis with a low empirical risk. The AdaBoost algorithm receives as input a training set of examples $S = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$, where for each i, $y_i = f(\mathbf{x}_i)$ for some labeling function f. The boosting process proceeds in a sequence of consecutive rounds. At round t, the booster first defines

a distribution over the examples in S, denoted $\mathbf{D}^{(t)}$. That is, $\mathbf{D}^{(t)} \in \mathbb{R}_+^m$ and $\sum_{i=1}^m D_i^{(t)} = 1$. Then, the booster passes the distribution $\mathbf{D}^{(t)}$ and the sample S to the weak learner. (That way, the weak learner can construct i.i.d. examples according to $\mathbf{D}^{(t)}$ and f.) The weak learner is assumed to return a "weak" hypothesis, h_t, whose error,

$$\epsilon_t \stackrel{\text{def}}{=} L_{\mathbf{D}^{(t)}}(h_t) \stackrel{\text{def}}{=} \sum_{i=1}^m D_i^{(t)} \mathbb{1}_{[h_t(\mathbf{x}_i) \neq y_i]},$$

is at most $\frac{1}{2} - \gamma$ (of course, there is a probability of at most δ that the weak learner fails). Then, AdaBoost assigns a weight for h_t as follows: $w_t = \frac{1}{2} \log\left(\frac{1}{\epsilon_t} - 1\right)$. That is, the weight of h_t is inversely proportional to the error of h_t. At the end of the round, AdaBoost updates the distribution so that examples on which h_t errs will get a higher probability mass while examples on which h_t is correct will get a lower probability mass. Intuitively, this will force the weak learner to focus on the problematic examples in the next round. The output of the AdaBoost algorithm is a "strong" classifier that is based on a weighted sum of all the weak hypotheses. The pseudocode of AdaBoost is presented in the following.

AdaBoost

input:
training set $S = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$
weak learner WL
number of rounds T
initialize $\mathbf{D}^{(1)} = (\frac{1}{m}, \ldots, \frac{1}{m})$.
for $t = 1, \ldots, T$:
invoke weak learner $h_t = \text{WL}(\mathbf{D}^{(t)}, S)$
compute $\epsilon_t = \sum_{i=1}^m D_i^{(t)} \mathbb{1}_{[y_i \neq h_t(\mathbf{x}_i)]}$
let $w_t = \frac{1}{2} \log\left(\frac{1}{\epsilon_t} - 1\right)$
update $D_i^{(t+1)} = \frac{D_i^{(t)} \exp(-w_t y_i h_t(\mathbf{x}_i))}{\sum_{j=1}^m D_j^{(t)} \exp(-w_t y_j h_t(\mathbf{x}_j))}$ for all $i = 1, \ldots, m$
output the hypothesis $h_s(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^T w_t h_t(\mathbf{x})\right)$.

The following theorem shows that the training error of the output hypothesis decreases exponentially fast with the number of boosting rounds.

THEOREM 10.2 *Let S be a training set and assume that at each iteration of AdaBoost, the weak learner returns a hypothesis for which $\epsilon_t \leq 1/2 - \gamma$. Then, the training error of the output hypothesis of AdaBoost is at most*

$$L_S(h_s) = \frac{1}{m} \sum_{i=1}^m \mathbb{1}_{[h_s(\mathbf{x}_i) \neq y_i]} \;\leq\; \exp(-2\,\gamma^2\,T)\ .$$

Proof For each t, denote $f_t = \sum_{p \leq t} w_p h_p$. Therefore, the output of AdaBoost

is f_T. In addition, denote

$$Z_t = \frac{1}{m}\sum_{i=1}^{m} e^{-y_i f_t(x_i)}.$$

Note that for any hypothesis we have that $\mathbb{1}_{[h(x)\neq y]} \leq e^{-yh(x)}$. Therefore, $L_S(f_T) \leq Z_T$, so it suffices to show that $Z_T \leq e^{-2\gamma^2 T}$. To upper bound Z_T we rewrite it as

$$Z_T = \frac{Z_T}{Z_0} = \frac{Z_T}{Z_{T-1}} \cdot \frac{Z_{T-1}}{Z_{T-2}} \cdots \frac{Z_2}{Z_1} \cdot \frac{Z_1}{Z_0}, \tag{10.2}$$

where we used the fact that $Z_0 = 1$ because $f_0 \equiv 0$. Therefore, it suffices to show that for every round t,

$$\frac{Z_{t+1}}{Z_t} \leq e^{-2\gamma^2}. \tag{10.3}$$

To do so, we first note that using a simple inductive argument, for all t and i,

$$D_i^{(t+1)} = \frac{e^{-y_i f_t(x_i)}}{\sum_{j=1}^{m} e^{-y_j f_t(x_j)}}.$$

Hence,

$$\begin{aligned}
\frac{Z_{t+1}}{Z_t} &= \frac{\sum_{i=1}^{m} e^{-y_i f_{t+1}(x_i)}}{\sum_{j=1}^{m} e^{-y_j f_t(x_j)}} \\
&= \frac{\sum_{i=1}^{m} e^{-y_i f_t(x_i)} e^{-y_i w_{t+1} h_{t+1}(x_i)}}{\sum_{j=1}^{m} e^{-y_j f_t(x_j)}} \\
&= \sum_{i=1}^{m} D_i^{(t+1)} e^{-y_i w_{t+1} h_{t+1}(x_i)} \\
&= e^{-w_{t+1}} \sum_{i:y_i h_{t+1}(x_i)=1} D_i^{(t+1)} + e^{w_{t+1}} \sum_{i:y_i h_{t+1}(x_i)=-1} D_i^{(t+1)} \\
&= e^{-w_{t+1}}(1-\epsilon_{t+1}) + e^{w_{t+1}}\epsilon_{t+1} \\
&= \frac{1}{\sqrt{1/\epsilon_{t+1}-1}}(1-\epsilon_{t+1}) + \sqrt{1/\epsilon_{t+1}-1}\;\epsilon_{t+1} \\
&= \sqrt{\frac{\epsilon_{t+1}}{1-\epsilon_{t+1}}}(1-\epsilon_{t+1}) + \sqrt{\frac{1-\epsilon_{t+1}}{\epsilon_{t+1}}}\epsilon_{t+1} \\
&= 2\sqrt{\epsilon_{t+1}(1-\epsilon_{t+1})}.
\end{aligned}$$

By our assumption, $\epsilon_{t+1} \leq \frac{1}{2} - \gamma$. Since the function $g(a) = a(1-a)$ is monotonically increasing in $[0, 1/2]$, we obtain that

$$2\sqrt{\epsilon_{t+1}(1-\epsilon_{t+1})} \leq 2\sqrt{\left(\frac{1}{2}-\gamma\right)\left(\frac{1}{2}+\gamma\right)} = \sqrt{1-4\gamma^2}.$$

Finally, using the inequality $1 - a \leq e^{-a}$ we have that $\sqrt{1-4\gamma^2} \leq e^{-4\gamma^2/2} = e^{-2\gamma^2}$. This shows that Equation (10.3) holds and thus concludes our proof. □

Each iteration of AdaBoost involves $O(m)$ operations as well as a single call to the weak learner. Therefore, if the weak learner can be implemented efficiently (as happens in the case of ERM with respect to decision stumps) then the total training process will be efficient.

Remark 10.2 Theorem 10.2 assumes that at each iteration of AdaBoost, the weak learner returns a hypothesis with weighted sample error of at most $1/2-\gamma$. According to the definition of a weak learner, it can fail with probability δ. Using the union bound, the probability that the weak learner will not fail at all of the iterations is at least $1 - \delta T$. As we show in Exercise 1, the dependence of the sample complexity on δ can always be logarithmic in $1/\delta$, and therefore invoking the weak learner with a very small δ is not problematic. We can therefore assume that δT is also small. Furthermore, since the weak learner is only applied with distributions over the training set, in many cases we can implement the weak learner so that it will have a zero probability of failure (i.e., $\delta = 0$). This is the case, for example, in the weak learner that finds the minimum value of $L_{\mathbf{D}}(h)$ for decision stumps, as described in the previous section.

Theorem 10.2 tells us that the empirical risk of the hypothesis constructed by AdaBoost goes to zero as T grows. However, what we really care about is the true risk of the output hypothesis. To argue about the true risk, we note that the output of AdaBoost is in fact a composition of a halfspace over the predictions of the T weak hypotheses constructed by the weak learner. In the next section we show that if the weak hypotheses come from a base hypothesis class of low VC-dimension, then the estimation error of AdaBoost will be small; namely, the true risk of the output of AdaBoost would not be very far from its empirical risk.

10.3 Linear Combinations of Base Hypotheses

As mentioned previously, a popular approach for constructing a weak learner is to apply the ERM rule with respect to a base hypothesis class (e.g., ERM over decision stumps). We have also seen that boosting outputs a composition of a halfspace over the predictions of the weak hypotheses. Therefore, given a base hypothesis class B (e.g., decision stumps), the output of AdaBoost will be a member of the following class:

$$L(B,T) = \left\{ x \mapsto \operatorname{sign}\left(\sum_{t=1}^{T} w_t h_t(x)\right) : \mathbf{w} \in \mathbb{R}^T,\ \forall t,\quad h_t \in B \right\}. \tag{10.4}$$

That is, each $h \in L(B,T)$ is parameterized by T base hypotheses from B and by a vector $w \in \mathbb{R}^T$. The prediction of such an h on an instance x is obtained by first applying the T base hypotheses to construct the vector $\psi(x) =$

$(h_1(x), \ldots, h_T(x)) \in \mathbb{R}^T$, and then applying the (homogenous) halfspace defined by $\mathbf{w}$ on $\psi(x)$.

In this section we analyze the estimation error of $L(B, T)$ by bounding the VC-dimension of $L(B, T)$ in terms of the VC-dimension of B and T. We will show that, up to logarithmic factors, the VC-dimension of $L(B, T)$ is bounded by T times the VC-dimension of B. It follows that the estimation error of AdaBoost grows linearly with T. On the other hand, the empirical risk of AdaBoost decreases with T. In fact, as we demonstrate later, T can be used to decrease the approximation error of $L(B, T)$. Therefore, the parameter T of AdaBoost enables us to control the bias-complexity tradeoff.

To demonstrate how the expressive power of $L(B, T)$ increases with T, consider the simple example, in which $\mathcal{X} = \mathbb{R}$ and the base class is Decision Stumps,

$$\mathcal{H}_{\mathrm{DS1}} = \{x \mapsto \mathrm{sign}(x - \theta) \cdot b : \quad \theta \in \mathbb{R}, b \in \{\pm 1\}\}.$$

Note that in this one dimensional case, $\mathcal{H}_{\mathrm{DS1}}$ is in fact equivalent to (nonhomogenous) halfspaces on $\mathbb{R}$.

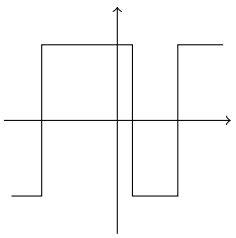
Now, let $\mathcal{H}$ be the rather complex class (compared to halfspaces on the line) of piece-wise constant functions. Let g_r be a piece-wise constant function with at most r pieces; that is, there exist thresholds $-\infty = \theta_0 < \theta_1 < \theta_2 < \cdots < \theta_r = \infty$ such that

$$g_r(x) = \sum_{i=1}^{r} \alpha_i \mathbb{1}_{[x \in (\theta_{i-1}, \theta_i]]} \quad \forall i, \quad \alpha_i \in \{\pm 1\}.$$

Denote by $\mathcal{G}_r$ the class of all such piece-wise constant classifiers with at most r pieces.

In the following we show that $\mathcal{G}_T \subseteq L(\mathcal{H}_{\mathrm{DS1}}, T)$; namely, the class of halfspaces over T decision stumps yields all the piece-wise constant classifiers with at most T pieces.

Indeed, without loss of generality consider any $g \in \mathcal{G}_T$ with $\alpha_t = (-1)^t$. This implies that if x is in the interval $(\theta_{t-1}, \theta_t]$, then $g(x) = (-1)^t$. For example:



Now, the function

$$h(x) = \mathrm{sign}\left(\sum_{t=1}^{T} w_t \, \mathrm{sign}(x - \theta_{t-1})\right), \tag{10.5}$$

where $w_1 = 0.5$ and for $t > 1$, $w_t = (-1)^t$, is in $L(\mathcal{H}_{\mathrm{DS1}}, T)$ and is equal to g (see Exercise 2).

From this example we obtain that $L(\mathcal{H}_{\mathrm{DS1}}, T)$ can shatter any set of $T+1$ instances in $\mathbb{R}$; hence the VC-dimension of $L(\mathcal{H}_{\mathrm{DS1}}, T)$ is at least $T+1$. Therefore, T is a parameter that can control the bias-complexity tradeoff: Enlarging T yields a more expressive hypothesis class but on the other hand might increase the estimation error. In the next subsection we formally upper bound the VC-dimension of $L(B, T)$ for any base class B.

10.3.1 The VC-Dimension of $L(B, T)$

The following lemma tells us that the VC-dimension of $L(B, T)$ is upper bounded by $\tilde{O}(\mathrm{VCdim}(B)\, T)$ (the $\tilde{O}$ notation ignores constants and logarithmic factors).

LEMMA 10.3 *Let B be a base class and let $L(B, T)$ be as defined in Equation (10.4). Assume that both T and $\mathrm{VCdim}(B)$ are at least 3. Then,*

$$\mathrm{VCdim}(L(B, T)) \leq T\,(\mathrm{VCdim}(B) + 1)\,(3\log(T\,(\mathrm{VCdim}(B) + 1)) + 2).$$

Proof Denote $d = \mathrm{VCdim}(B)$. Let $C = \{x_1, \ldots, x_m\}$ be a set that is shattered by $L(B, T)$. Each labeling of C by $h \in L(B, T)$ is obtained by first choosing $h_1, \ldots, h_T \in B$ and then applying a halfspace hypothesis over the vector $(h_1(x), \ldots, h_T(x))$. By Sauer's lemma, there are at most $(em/d)^d$ different dichotomies (i.e., labelings) induced by B over C. Therefore, we need to choose T hypotheses, out of at most $(em/d)^d$ different hypotheses. There are at most $(em/d)^{dT}$ ways to do it. Next, for each such choice, we apply a linear predictor, which yields at most $(em/T)^T$ dichotomies. Therefore, the overall number of dichotomies we can construct is upper bounded by

$$(em/d)^{dT}\,(em/T)^T \leq m^{(d+1)T},$$

where we used the assumption that both d and T are at least 3. Since we assume that C is shattered, we must have that the preceding is at least 2^m, which yields

$$2^m \leq m^{(d+1)T}.$$

Therefore,

$$m \leq \log(m)\frac{(d+1)T}{\log(2)}\ .$$

Lemma A.1 in Chapter A tells us that a necessary condition for the above to hold is that

$$m \leq 2\frac{(d+1)T}{\log(2)}\log\frac{(d+1)T}{\log(2)} \leq (d+1)T(3\log((d+1)T) + 2),$$

which concludes our proof. □

In Exercise 4 we show that for some base classes, B, it also holds that $\mathrm{VCdim}(L(B, T)) \geq \Omega(\mathrm{VCdim}(B)\, T)$.

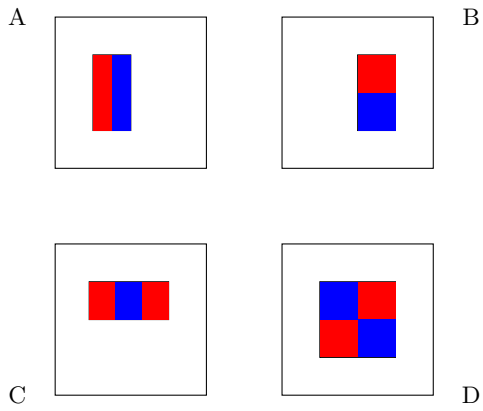


Figure 10.1 The four types of functions, g, used by the base hypotheses for face recognition. The value of g for type A or B is the difference between the sum of the pixels within two rectangular regions. These regions have the same size and shape and are horizontally or vertically adjacent. For type C, the value of g is the sum within two outside rectangles subtracted from the sum in a center rectangle. For type D, we compute the difference between diagonal pairs of rectangles.

10.4 AdaBoost for Face Recognition

We now turn to a base hypothesis that has been proposed by Viola and Jones for the task of face recognition. In this task, the instance space is images, represented as matrices of gray level values of pixels. To be concrete, let us take images of size 24×24 pixels, and therefore our instance space is the set of real valued matrices of size 24×24. The goal is to learn a classifier, $h : \mathcal{X} \to \{\pm 1\}$, that given an image as input, should output whether the image is of a human face or not.

Each hypothesis in the base class is of the form $h(x) = f(g(x))$, where f is a decision stump hypothesis and $g : \mathbb{R}^{24,24} \to \mathbb{R}$ is a function that maps an image to a scalar. Each function g is parameterized by

- An axis aligned rectangle R. Since each image is of size 24×24, there are at most 24^4 axis aligned rectangles.
- A type, $t \in \{A, B, C, D\}$. Each type corresponds to a mask, as depicted in Figure 10.1.

To calculate g we stretch the mask t to fit the rectangle R and then calculate the sum of the pixels (that is, sum of their gray level values) that lie within the red rectangles and subtract it from the sum of pixels in the blue rectangles.

Since the number of such functions g is at most $24^4 \cdot 4$, we can implement a weak learner for the base hypothesis class by first calculating all the possible outputs of g on each image, and then apply the weak learner of decision stumps described in the previous subsection. It is possible to perform the first step very

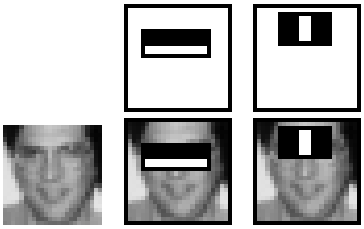


Figure 10.2 The first and second features selected by AdaBoost, as implemented by Viola and Jones. The two features are shown in the top row and then overlaid on a typical training face in the bottom row. The first feature measures the difference in intensity between the region of the eyes and a region across the upper cheeks. The feature capitalizes on the observation that the eye region is often darker than the cheeks. The second feature compares the intensities in the eye regions to the intensity across the bridge of the nose.

efficiently by a preprocessing step in which we calculate the integral image of each image in the training set. See Exercise 5 for details.

In Figure 10.2 we depict the first two features selected by AdaBoost when running it with the base features proposed by Viola and Jones.

10.5 Summary

Boosting is a method for amplifying the accuracy of weak learners. In this chapter we described the AdaBoost algorithm. We have shown that after T iterations of AdaBoost, it returns a hypothesis from the class $L(B, T)$, obtained by composing a linear classifier on T hypotheses from a base class B. We have demonstrated how the parameter T controls the tradeoff between approximation and estimation errors. In the next chapter we will study how to tune parameters such as T, based on the data.

10.6 Bibliographic Remarks

As mentioned before, boosting stemmed from the theoretical question of whether an efficient weak learner can be "boosted" into an efficient strong learner (Kearns & Valiant 1988) and solved by Schapire (1990). The AdaBoost algorithm has been proposed in Freund & Schapire (1995).

Boosting can be viewed from many perspectives. In the purely theoretical context, AdaBoost can be interpreted as a negative result: If strong learning of a hypothesis class is computationally hard, so is weak learning of this class. This negative result can be useful for showing hardness of agnostic PAC learning of a class B based on hardness of PAC learning of some other class $\mathcal{H}$, as long as