Chapter 6

Game theory, On-line Learning and Boosting

Having studied methods of analyzing boosting's training and generalization errors, we turn now to some of the other ways that boosting can be thought about, understood and interpreted. We begin with the fundamental and beautiful connection between boosting and game theory. Using mathematical abstractions, *game theory* studies ordinary games, like chess and checkers, but more generally, the field also attempts to model all forms of interactions between people, animals, corporations, nations, software agents, etc. In boosting, there is a natural interaction between two agents, namely, the boosting algorithm and the weak learning algorithm. As we will see, these two agents are in fact playing a game repeatedly in a standard game-theoretic fashion. Moreover, we will see that some of the key concepts of boosting, including margin, edge, the weak learning assumption, etc. all have direct and natural interpretations in the game-theoretic context. Indeed, the principle that boosting should be possible at all, given the weak learning assumption, is very closely related to von Neumann's famous minmax theorem, the fundamental theorem of zero-sum games. Moreover, the learning framework presented in this chapter allows us to give a very simple proof of this classic theorem.

AdaBoost and its simplified variants turn out to be special cases of a more general algorithm for playing general repeated games. We devote much of this chapter to a description of this general game-playing algorithm. Later, we will see how boosting can be achieved and understood as a special case for an appropriately chosen game. Moreover, by reversing the roles of the two players, we will see that a solution is obtained for a different learning problem, namely, the well-studied on-line prediction model in which a learning agent predicts the classifications of a sequence of instances while attempting to minimize the number of prediction mis-

takes. Thus, an extremely tight connection between boosting and on-line learning is revealed by placing them both in a general game-theoretic context.

We end the chapter with an application to a simple game which involves an element of "mind-reading."

6.1 Game theory

We begin with a review of basic game theory. We study two-person games in so-called *normal form*. Such a game is defined by a matrix $\mathbf{M}$. There are two players called the row player and the column player. To play the game, the row player chooses a row i, and, simultaneously, the column player chooses a column j. The selected entry $\mathbf{M}(i, j)$ is the *loss* suffered by the row player. (Although it is common in game theory for the players' purpose to be specified in terms of a "gain" or "reward" to be maximized, we use an equivalent, if gloomier, formulation based on "loss" for the sake of consistency with the rest of the book.)

As an example, the loss matrix for the children's game[1] "Rock-Paper-Scissors" is given by:

	Rock	Paper	Scissors
Rock	$\frac{1}{2}$	1	0
Paper	0	$\frac{1}{2}$	1
Scissors	1	0	$\frac{1}{2}$

For instance, if the row player plays Paper and the column player plays Scissors then the row player loses, suffering a loss of 1.

The row player's goal is to minimize its loss, and we will generally focus mainly on this player's perspective of the game. Often, the goal of the column player is to maximize this loss, in which case the game is said to be *zero-sum*, so named because the column player's loss can be viewed as the negative of the row player's so that the losses of the two players always add up to exactly zero. Most of our results are given in the context of a zero-sum game. However, the results also apply when no assumptions are made about the goal or strategy of the column player, who might possibly have some other purpose in mind. We return to this point below.

[1] In this game, each of two children simultaneously throw down hand signals indicating Rock, Paper or Scissors. If one plays Scissors, for instance, and the other Paper, then the former wins since "Scissors cut Paper." Similarly, Paper beats Rock, and Rock beats Scissors. A tie occurs if the same object is chosen by both children.

6.1.1 Randomized play

As described above, the players each choose a single row or column. Usually, this choice of play is allowed to be randomized. That is, the row player chooses a distribution P over the rows of $\mathbf{M}$, and (simultaneously) the column player chooses a distribution Q over columns. The two distributions P and Q define the random choice of row or column. The row player's expected loss is then easily computed as

$$\mathbf{M}(P, Q) \doteq \sum_{i,j} P(i)\mathbf{M}(i, j)Q(j) = P^\top \mathbf{M}Q,$$

where we sometimes, as in this expression, regard P and Q as vectors. For ease of notation, we denote this quantity by $\mathbf{M}(P, Q)$, as above, and refer to it simply as the loss (rather than expected loss). In addition, if the row player chooses a distribution P but the column player chooses a single column j, then the (expected) loss is $\sum_i P(i)\mathbf{M}(i, j)$ which we denote by $\mathbf{M}(P, j)$. The notation $\mathbf{M}(i, Q)$ is defined analogously.

Individual (deterministically chosen) rows i and columns j are called *pure strategies*. Randomized plays defined by distributions P and Q over rows and columns are called *mixed strategies*. The number of rows of the matrix $\mathbf{M}$ will be denoted by m.

6.1.2 Sequential play

Up until now, we have assumed that the players choose their (pure or mixed) strategies simultaneously. Suppose now that play instead is sequential. That is, suppose that the column player chooses its strategy Q *after* the row player has chosen and announced its strategy P. Assume further that the column player's goal is to maximize the row player's loss (that is, that the game is zero-sum). Then given knowledge of P, such a "worst-case" or "adversarial" column player will choose Q to maximize $\mathbf{M}(P, Q)$; that is, if the row player plays mixed strategy P, then its loss will be

$$\max_Q \mathbf{M}(P, Q). \tag{6.1}$$

(It is understood here and throughout the chapter that $\max_Q$ denotes maximum over all probability distributions over columns; similarly, $\min_P$ will always denote minimum over all probability distributions over rows. These extrema exist because the set of distributions over a finite space is compact.)

Eq. (6.1) can be viewed as a function of P that specifies what the loss will be for the row player if it chooses to play that particular strategy. Knowing this, the row player should choose P to minimize this expression. Doing so will result in a

loss for the row player of exactly

$$\min_P \max_Q \mathbf{M}(P, Q). \tag{6.2}$$

Thus, this quantity represents the loss that will be suffered when the row player plays first, followed by the column player, and assuming both play optimally. Note that the order of the $\min \max$ in Eq. (6.2) matches the order of play (although, of course, the minimum and maximum would be evaluated mathematically from the inside out).

A mixed strategy P^* realizing the minimum in Eq. (6.2) is called a *minmax strategy*, and is optimal in this particular setting.

If now we reverse the order of play so that the column player plays first and the row player can choose its play with the benefit of knowing the column player's chosen strategy Q, then by a symmetric argument, the loss of the row player will be

$$\max_Q \min_P \mathbf{M}(P, Q).$$

A strategy Q^* realizing the maximum is called a *maxmin strategy*.

Note that, because

$$\mathbf{M}(P, Q) = \sum_{j=1}^{n} \mathbf{M}(P, j) Q(j),$$

the maximum over distributions Q in Eq. (6.1) will always be realized when Q is concentrated on a single column j. In other words, for any P,

$$\max_Q \mathbf{M}(P, Q) = \max_j \mathbf{M}(P, j) \tag{6.3}$$

and similarly, for any Q,

$$\min_P \mathbf{M}(P, Q) = \min_i \mathbf{M}(i, Q) \tag{6.4}$$

(where $\min_i$ and $\max_j$ will always denote minimum over rows i or maximum over columns j). On the other hand, the minmax strategy P^* which realizes the minimum in Eq. (6.2) will not, in general, be a pure strategy (likewise for Q^*).

6.1.3 The minmax theorem

Intuitively, we expect the player who chooses its strategy last to have the advantage since it plays knowing its opponent's strategy exactly—at least, we expect there to

be no disadvantage in playing second. Thus, we expect the row player's loss to be no greater when playing second than when playing first so that

$$\max_Q \min_P \mathbf{M}(P,Q) \leq \min_P \max_Q \mathbf{M}(P,Q). \tag{6.5}$$

Indeed, this is true in general. We might go on naively to conjecture that there is a real advantage to playing last in some games so that, at least in some cases, the inequality in Eq. (6.5) is strict. Surprisingly, it turns out not to matter which player plays first. Von Neumann's well-known *minmax theorem* states that the outcome is the same in either case so that

$$\max_Q \min_P \mathbf{M}(P,Q) = \min_P \max_Q \mathbf{M}(P,Q) \tag{6.6}$$

for every matrix $\mathbf{M}$. The common value v of the two sides of the equality is called the *value* of the game $\mathbf{M}$. A proof of the minmax theorem will be given in Section 6.2.4.

In words, Eq. (6.6) means that the row player has a (minmax) strategy P^* such that regardless of the strategy Q played by the column player, even if chosen with knowledge of P^*, the loss suffered $\mathbf{M}(P^*,Q)$ will be *at most* v. Moreover, P^* is optimal in the sense that by playing Q^*, the column player can force a loss of *at least* v for any strategy P played by the row player, including P^*. The (maxmin) strategy Q^* is symmetrically optimal.

For instance, for Rock-Paper-Scissors, the optimal minmax strategy is to play each of the three possible moves with equal probability $1/3$. Regardless of what the opponent does, the expected loss for this strategy will always be exactly $1/2$, the value of this game. In this case, playing any other (mixed) strategy, if known to an optimal opponent, will result in a strictly higher loss.

Thus, classical game theory says that given a (zero-sum) game $\mathbf{M}$, one should play using a minmax strategy. Computing such a strategy, a problem called *solving* the game, can be accomplished using linear programming, that is, using standard techniques for maximizing a linear function subject to linear inequality constraints. However, there are a number of problems with this approach. For instance,

- $\mathbf{M}$ may be unknown;
- $\mathbf{M}$ may be so large that computing a minmax strategy using linear programming becomes infeasible;
- the column player may not be truly adversarial and may behave in a manner that admits loss significantly smaller than the game value v.

Regarding the last point, consider again the example of Rock-Paper-Scissors. Suppose, as happened on one episode of *The Simpsons*, that Bart is playing against his sister Lisa. Lisa thinks, "Poor predictable Bart, always takes Rock," while Bart thinks, "Good old Rock, nothing beats that." If Lisa were to follow the supposedly "optimal" minmax strategy given above, she would still suffer loss of $1/2$, and would miss an obvious opportunity to beat Bart every single time by always playing Paper. This is because a minmax strategy is intended for use against a fully adversarial opponent (in this case, one much smarter than Bart), and will perform suboptimally if played against a suboptimal opponent.

6.2 Learning in repeated game playing

If playing a game only once, we cannot hope to overcome a lack of prior knowledge about either the game $\mathbf{M}$ or the opponent's intentions and abilities. However, in *repeated* play in which the same game is played over and over again against the same opponent, one can hope to *learn* to play the game well against the particular opponent being faced. This is the main topic of this section.

6.2.1 The learning model

We begin by formalizing a model of repeated play. To simplify the presentation, we assume for the remainder of this chapter that all of the losses appearing in the matrix $\mathbf{M}$ are in the range $[0, 1]$. This does not at all limit the generality of the results since any matrix, having only a finite number of entries, can be shifted and scaled to satisfy this assumption, without fundamentally altering the game.

To emphasize the roles of the two players, we here refer to the row player as the *learner* and the column player as the *environment*. As before, $\mathbf{M}$ is a game matrix, possibly unknown to the learner. This game is played repeatedly in a sequence of *rounds*. On round $t = 1, \ldots, T$:

1. the learner chooses mixed strategy P_t;

2. the environment chooses mixed strategy Q_t (which may be chosen with knowledge of P_t);

3. the learner is permitted to observe the loss $\mathbf{M}(i, Q_t)$ for each row i; this is the loss it would have suffered had it played using pure strategy i;

4. the learner suffers loss $\mathbf{M}(P_t, Q_t)$.

The basic goal of the learner is to minimize its total *cumulative loss*:

$$\sum_{t=1}^{T} \mathbf{M}(P_t, Q_t). \tag{6.7}$$

If the environment is adversarial then a related goal is to approximate the performance of the optimal, minmax strategy P^*. However, for more benign environments, the goal may be to suffer the minimum loss possible, which may be much better than the value of the game. Thus, the goal of the learner is to do almost as well as the best strategy against the actual sequence of plays $Q_1, \ldots, Q_T$ which were chosen by the environment. That is, the learner's goal is to suffer cumulative loss which is "not much worse" than the cumulative loss of the *best* (fixed) strategy *in hindsight*, namely,

$$\min_P \sum_{t=1}^{T} \mathbf{M}(P, Q_t). \tag{6.8}$$

6.2.2 The basic algorithm

We now describe an algorithm for achieving this goal in repeated play, which we call MW for "multiplicative weights." The learning algorithm MW starts with some initial mixed strategy P_1 which it uses for the first round of the game. After each round t, the learner computes a new mixed strategy P_{t+1} by a simple multiplicative rule:

$$P_{t+1}(i) = \frac{P_t(i) \exp(-\eta \mathbf{M}(i, Q_t))}{Z_t} \tag{6.9}$$

where Z_t is a normalization factor:

$$Z_t = \sum_{i=1}^{m} P_t(i) \exp(-\eta \mathbf{M}(i, Q_t)), \tag{6.10}$$

and $\eta > 0$ is a parameter of the algorithm. This is a very intuitive rule which has the effect of increasing the chance of playing strategies with low loss on the preceding round (for which $\mathbf{M}(i, Q_t)$ is small), while similarly decreasing the chance of playing strategies with high loss. Later, we discuss the choice of P_1 and η.

6.2.3 Analysis

The main theorem concerning this algorithm is given next. Roughly speaking, this general theorem gives a bound on the learner's cumulative loss (Eq. (6.7)) in terms of the cumulative loss of the best strategy in hindsight (Eq. (6.8)), plus an additional

term which will be shown later to be relatively insignificant for large T. As we will see, this result will have many implications.

The theorem and its proof make use of a measure of distance (or "divergence") between two probability distributions P and P' over $\{1, \ldots, m\}$ called *relative entropy*, also known as *Kullback-Leibler divergence*:

$$\mathrm{RE}\left(P \parallel P'\right) \doteq \sum_{i=1}^{m} P(i) \ln\left(\frac{P(i)}{P'(i)}\right). \tag{6.11}$$

This measure, though not a metric, is always nonnegative, and is equal to zero if and only if $P = P'$. See Section 8.1.2 for further background.

Theorem 6.1 *For any matrix $\mathbf{M}$ with m rows and entries in $[0, 1]$, and for any sequence of mixed strategies $Q_1, \ldots, Q_T$ played by the environment, the sequence of mixed strategies $P_1, \ldots, P_T$ produced by algorithm MW with parameter η satisfies:*

$$\sum_{t=1}^{T} \mathbf{M}(P_t, Q_t) \leq \min_{P} \left[a_\eta \sum_{t=1}^{T} \mathbf{M}(P, Q_t) + c_\eta \mathrm{RE}\left(P \parallel P_1\right) \right]$$

where

$$a_\eta = \frac{\eta}{1 - e^{-\eta}} \qquad c_\eta = \frac{1}{1 - e^{-\eta}}.$$

Our proof uses a kind of "amortized analysis" in which relative entropy is used as a "potential" function, or measure of progress. The heart of the proof is in the following lemma, which bounds the change in potential before and after a single round. Note that the potential is measured relative to an arbitrary *reference distribution* $\tilde{P}$, which can be thought of as the "best" distribution, although the analysis actually applies simultaneously to all possible choices of $\tilde{P}$. In words, the lemma shows that whenever the learner suffers significant loss relative to $\tilde{P}$, the potential must drop substantially. Since the potential can never become negative, this will allow us to bound the learner's cumulative loss relative to that of $\tilde{P}$.

Lemma 6.2 *For any iteration t where MW is used with parameter η, and for any mixed strategy $\tilde{P}$,*

$$\mathrm{RE}\left(\tilde{P} \parallel P_{t+1}\right) - \mathrm{RE}\left(\tilde{P} \parallel P_t\right) \leq \eta \mathbf{M}(\tilde{P}, Q_t) + \ln\left(1 - (1 - e^{-\eta})\mathbf{M}(P_t, Q_t)\right).$$

Proof: We have the following sequence of inequalities:

$$\begin{aligned}
&\mathrm{RE}\left(\tilde{P} \parallel P_{t+1}\right) - \mathrm{RE}\left(\tilde{P} \parallel P_t\right) \\
&= \sum_{i=1}^{m} \tilde{P}(i) \ln\left(\frac{\tilde{P}(i)}{P_{t+1}(i)}\right) - \sum_{i=1}^{m} \tilde{P}(i) \ln\left(\frac{\tilde{P}(i)}{P_t(i)}\right) && (6.12)\\
&= \sum_{i=1}^{m} \tilde{P}(i) \ln\left(\frac{P_t(i)}{P_{t+1}(i)}\right) \\
&= \sum_{i=1}^{m} \tilde{P}(i) \ln\left(\frac{Z_t}{\exp(-\eta \mathbf{M}(i, Q_t))}\right) && (6.13)\\
&= \eta \sum_{i=1}^{m} \tilde{P}(i) \mathbf{M}(i, Q_t) + \ln Z_t \\
&= \eta \sum_{i=1}^{m} \tilde{P}(i) \mathbf{M}(i, Q_t) + \ln\left[\sum_{i=1}^{m} P_t(i) \exp(-\eta \mathbf{M}(i, Q_t))\right] && (6.14)\\
&\leq \eta \mathbf{M}(\tilde{P}, Q_t) + \ln\left[\sum_{i=1}^{m} P_t(i) \left(1 - (1 - e^{-\eta}) \mathbf{M}(i, Q_t)\right)\right] && (6.15)\\
&= \eta \mathbf{M}(\tilde{P}, Q_t) + \ln\left[1 - (1 - e^{-\eta}) \mathbf{M}(P_t, Q_t)\right].
\end{aligned}$$

Eq. (6.12) follows from the definition of relative entropy. Eq. (6.13) follows from the update rule of MW given in Eq. (6.9). Eq. (6.14) uses the definition of Z_t in Eq. (6.10). And Eq. (6.15) uses the fact that, by convexity of e^x, for $q \in [0, 1]$,

$$e^{-\eta q} = \exp(q(-\eta) + (1 - q) \cdot 0) \leq q e^{-\eta} + (1 - q) e^0 = 1 - (1 - e^{-\eta}) q.$$

■

Proof of Theorem 6.1: Let $\tilde{P}$ be any mixed row strategy. We first simplify the last term in the inequality of Lemma 6.2 by using the fact that $\ln(1 - x) \leq -x$ for any $x < 1$ which implies that

$$\mathrm{RE}\left(\tilde{P} \parallel P_{t+1}\right) - \mathrm{RE}\left(\tilde{P} \parallel P_t\right) \leq \eta \mathbf{M}(\tilde{P}, Q_t) - (1 - e^{-\eta}) \mathbf{M}(P_t, Q_t).$$

Summing this inequality over $t = 1, \ldots, T$ we get

$$\mathrm{RE}\left(\tilde{P} \parallel P_{T+1}\right) - \mathrm{RE}\left(\tilde{P} \parallel P_1\right) \leq \eta \sum_{t=1}^{T} \mathbf{M}(\tilde{P}, Q_t) - (1 - e^{-\eta}) \sum_{t=1}^{T} \mathbf{M}(P_t, Q_t).$$

Rearranging the inequality and noting that $\mathrm{RE}\left(\tilde{P} \parallel P_{T+1}\right) \geq 0$ gives

$$\begin{aligned}(1-e^{-\eta})\sum_{t=1}^{T}\mathbf{M}(P_t,Q_t) &\leq \eta\sum_{t=1}^{T}\mathbf{M}(\tilde{P},Q_t)+\mathrm{RE}\left(\tilde{P} \parallel P_1\right)-\mathrm{RE}\left(\tilde{P} \parallel P_{T+1}\right)\\ &\leq \eta\sum_{t=1}^{T}\mathbf{M}(\tilde{P},Q_t)+\mathrm{RE}\left(\tilde{P} \parallel P_1\right).\end{aligned}$$

Since $\tilde{P}$ was chosen arbitrarily, this gives the statement of the theorem. ■.

In order to use MW, we need to choose the initial distribution P_1 and the parameter η. We start with the choice of P_1. In general, the closer P_1 is to a good mixed strategy $\tilde{P}$, the better the bound on the total loss MW. However, even if we have no prior knowledge about the good mixed strategies, we can achieve reasonable performance by using the uniform distribution over the rows as the initial strategy. This gives us a performance bound that holds uniformly for all games with m rows. Note that there is *no* explicit dependence in this bound on the number of columns and only logarithmic dependence on the number of rows. Later, we exploit both these properties in the applications that follow.

Corollary 6.3 *If MW is used with P_1 set to the uniform distribution then its total loss is bounded by*

$$\sum_{t=1}^{T}\mathbf{M}(P_t,Q_t) \leq a_\eta \min_P \sum_{t=1}^{T}\mathbf{M}(P,Q_t)+c_\eta \ln m$$

where a_η and c_η are as defined in Theorem 6.1.

Proof: If $P_1(i)=1/m$ for all i then $\mathrm{RE}\left(P \parallel P_1\right) \leq \ln m$ for all P. ■

Next we discuss the choice of the parameter η. As η approaches zero, a_η approaches 1 from above while c_η increases to infinity. On the other hand, if we fix η and let the number of rounds T increase, the second term $c_\eta \ln m$ becomes negligible (since it is fixed) relative to T. Thus, by choosing η as a function of T which approaches 0 for $T \to \infty$, the learner can ensure that its average per-trial loss will not be much worse than the loss of the best strategy. This is formalized in the following corollary:

Corollary 6.4 *Under the conditions of Theorem 6.1 and with η set to*

$$\ln\left(1+\sqrt{\frac{2\ln m}{T}}\right),$$

the average per-trial loss suffered by the learner is

$$\frac{1}{T}\sum_{t=1}^{T}\mathbf{M}(P_t, Q_t) \leq \min_P \frac{1}{T}\sum_{t=1}^{T}\mathbf{M}(P, Q_t) + \Delta_T$$

where

$$\Delta_T \doteq \sqrt{\frac{2\ln m}{T}} + \frac{\ln m}{T} = O\left(\sqrt{\frac{\ln m}{T}}\right).$$

Proof: By Corollary 6.3,

$$\begin{aligned}
\sum_{t=1}^{T}\mathbf{M}(P_t, Q_t) &\leq \min_P \sum_{t=1}^{T}\mathbf{M}(P, Q_t) + (a_\eta - 1)T + c_\eta \ln m \qquad (6.16)\\
&= \min_P \sum_{t=1}^{T}\mathbf{M}(P, Q_t) + \left[\left(\frac{\eta}{1-e^{-\eta}} - 1\right)T + \frac{\ln m}{1-e^{-\eta}}\right].\\
&\leq \min_P \sum_{t=1}^{T}\mathbf{M}(P, Q_t) + \left[\left(\frac{e^{\eta}-e^{-\eta}}{2(1-e^{-\eta})} - 1\right)T + \frac{\ln m}{1-e^{-\eta}}\right] \qquad (6.17)
\end{aligned}$$

In Eq. (6.16), we used our assumption that the losses in $\mathbf{M}$ are bounded in $[0, 1]$, which implies that the loss in any sequence of T plays cannot exceed T. In Eq. (6.17), we used the approximation $\eta \leq (e^{\eta} - e^{-\eta})/2$ which holds for any $\eta \geq 0$ since, by Taylor series expansion,

$$\frac{e^{\eta}-e^{-\eta}}{2} = \eta + \frac{\eta^3}{3!} + \frac{\eta^5}{5!} + \cdots \geq \eta.$$

Minimizing the bracketed expression on the right-hand side of Eq. (6.17) gives the stated choice of η. Plugging in this choice gives the stated bound. ■

Since $\Delta_T \to 0$ as $T \to \infty$, we see that the amount by which the average per-trial loss of the learner exceeds that of the best mixed strategy can be made arbitrarily small for large T. In other words, even with no prior knowledge of $\mathbf{M}$ or the environment, we see that the learner plays almost as well on-line as if it knew ahead of time both the matrix $\mathbf{M}$ and the exact sequence of plays of the environment $Q_1, \ldots, Q_T$ (assuming the learner is restricted to use a fixed mixed strategy for the entire sequence).

Note that in the analysis we made no assumptions at all about the environment. Theorem 6.1 guarantees that the learner's cumulative loss is not much larger than that of *any* fixed mixed strategy. As shown in the next corollary, this implies that the loss cannot be much larger than the game value. However, this is a considerable

weakening of the general result: if the environment is non-adversarial, there might be a better row strategy, in which case the algorithm is guaranteed to be almost as good as this better strategy.

Corollary 6.5 *Under the conditions of Corollary 6.4,*

$$\frac{1}{T}\sum_{t=1}^{T}\mathbf{M}(P_t, Q_t) \leq v + \Delta_T$$

where v is the value of the game $\mathbf{M}$.

Proof: Let P^* be a minmax strategy for $\mathbf{M}$ so that for all column strategies Q, $\mathbf{M}(P^*, Q) \leq v$. Then, by Corollary 6.4,

$$\frac{1}{T}\sum_{t=1}^{T}\mathbf{M}(P_t, Q_t) \leq \frac{1}{T}\sum_{t=1}^{T}\mathbf{M}(P^*, Q_t) + \Delta_T \leq v + \Delta_T.$$

■

6.2.4 Proof of the minmax theorem

More interestingly, Corollary 6.4 can be used to derive a very simple proof of von Neumann's minmax theorem as discussed in Section 6.1.3. To prove this theorem, we need to show that

$$\min_P \max_Q \mathbf{M}(P, Q) = \max_Q \min_P \mathbf{M}(P, Q).$$

Proving that

$$\min_P \max_Q \mathbf{M}(P, Q) \geq \max_Q \min_P \mathbf{M}(P, Q), \tag{6.18}$$

as suggested earlier, is straightforward: For any $\tilde{P}$ and any Q, $\mathbf{M}(\tilde{P}, Q) \geq \min_P \mathbf{M}(P, Q)$. Thus, $\max_Q \mathbf{M}(\tilde{P}, Q) \geq \max_Q \min_P \mathbf{M}(P, Q)$. Since this holds for all $\tilde{P}$, we get Eq. (6.18). So the hard part of proving the minmax theorem is showing that

$$\min_P \max_Q \mathbf{M}(P, Q) \leq \max_Q \min_P \mathbf{M}(P, Q). \tag{6.19}$$

Suppose that we run algorithm MW (with η set as in Corollary 6.4) against a maximally adversarial environment which always chooses strategies that maximize the learner's loss. That is, on each round t, the environment chooses

$$Q_t = \arg\max_Q \mathbf{M}(P_t, Q). \tag{6.20}$$

Let $\overline{P}$ and $\overline{Q}$ be the average of the strategies played by each side:

$$\overline{P} \doteq \frac{1}{T}\sum_{t=1}^{T} P_t \qquad \overline{Q} \doteq \frac{1}{T}\sum_{t=1}^{T} Q_t. \tag{6.21}$$

Clearly, $\overline{P}$ and $\overline{Q}$ are probability distributions.

Then we have:

$$\begin{aligned}
\min_P \max_Q P^\top \mathbf{M} Q &\leq \max_Q \overline{P}^\top \mathbf{M} Q \\
&= \max_Q \frac{1}{T}\sum_{t=1}^{T} {P_t}^\top \mathbf{M} Q && \text{by definition of } \overline{P} \\
&\leq \frac{1}{T}\sum_{t=1}^{T} \max_Q {P_t}^\top \mathbf{M} Q \\
&= \frac{1}{T}\sum_{t=1}^{T} {P_t}^\top \mathbf{M} Q_t && \text{by definition of } Q_t \\
&\leq \min_P \frac{1}{T}\sum_{t=1}^{T} P^\top \mathbf{M} Q_t + \Delta_T && \text{by Corollary 6.4} \\
&= \min_P P^\top \mathbf{M} \overline{Q} + \Delta_T && \text{by definition of } \overline{Q} \\
&\leq \max_Q \min_P P^\top \mathbf{M} Q + \Delta_T.
\end{aligned}$$

Since Δ_T can be made arbitrarily close to zero, this proves Eq. (6.19) and the minmax theorem.

6.2.5 Approximately solving a game

Aside from yielding a proof for a famous theorem that by now has many proofs, the preceding derivation shows that algorithm MW can be used to find an approximate minmax or maxmin strategy, that is, for approximately solving the game $\mathbf{M}$.

Skipping the first inequality of the sequence of equalities and inequalities given above, we see that

$$\max_Q \mathbf{M}(\overline{P}, Q) \leq \max_Q \min_P \mathbf{M}(P, Q) + \Delta_T = v + \Delta_T.$$

Thus, the mixed strategy $\overline{P}$ is an *approximate minmax strategy* in the sense that for all column strategies Q, $\mathbf{M}(\overline{P}, Q)$ does not exceed the game value v by more

than Δ_T. Since Δ_T can be made arbitrarily small, this approximation can be made arbitrarily tight.

Similarly, ignoring the last inequality of this derivation, we have that

$$\min_P \mathbf{M}(P, \overline{Q}) \geq v - \Delta_T$$

so $\overline{Q}$ also is an *approximate maxmin strategy*. Furthermore, by Eq. (6.3), Q_t satisfying Eq. (6.20) can always be chosen to be a pure strategy (that is, a mixed strategy concentrated on a single column of $\mathbf{M}$). Therefore, the approximate maxmin strategy $\overline{Q}$ has the additional favorable property of being *sparse* in the sense that at most T of its entries will be nonzero.

Viewing MW as a method for approximately solving a game will be central to our derivation of a boosting algorithm (Section 6.4).

6.3 On-line prediction

So far in this book, we have only considered learning in a "batch" setting in which the learner is provided with a random batch of training examples, and must formulate a single hypothesis which is then used for making predictions on new random test examples. Once training is complete, no further changes are made to the chosen prediction rule.

In contrast, in the *on-line prediction model*, the learner instead observes a sequence of examples and predicts their labels one at a time. Thus, at each of a series of time steps $t = 1, \ldots, T$, the learner is presented with an instance x_t, predicts the label for x_t, and then immediately gets to observe the correct label for x_t. A mistake occurs if the predicted and correct labels disagree. The learner's goal is to minimize the total number of mistakes made.

As a concrete example of where on-line learning might be appropriate, suppose we wish to predict stock market behavior over the course of some time period. Each morning, based on current market conditions, we make a prediction whether or not some market indicator will go up or down that day. Then each evening, we find out whether that morning's prediction was right or not. Our goal is to learn over time how to make accurate predictions so that the total number of mistakes will be small.

There are important differences between the on-line and batch models. In the batch model, there is a strict division between the training phase and the testing phase. In the on-line model, training and testing occur together all at the same time since every example acts as both a test of what was learned in the past, and as a training example for improving our predictions going forward into the future.

A second key difference relates to the generation of examples. In the batch setting, we always assumed all examples to be random, independently and identically distributed. In the on-line model, *no* assumptions are made about the generation of examples. The sequence of examples is entirely arbitrary, and might even be under the control of an adversary.

The labels also might be adversarially chosen, in which case there is no way to limit the number of mistakes that might be forced on the learner. To provide meaningful results in such a setting, we therefore seek learning algorithms that perform well relative to the best fixed prediction rule or hypothesis in some possibly very large class of hypotheses. Thus, if there is any one hypothesis in this class that makes accurate predictions, then our learning algorithm should do so as well.

Historically, the game-playing algorithm MW presented above was a direct generalization of an on-line prediction algorithm called the *Weighted Majority Algorithm*. It is not surprising, therefore, that an on-line prediction algorithm can be derived from the more general game-playing algorithm by an appropriate choice of game $\mathbf{M}$. In this section, we make this connection explicit as a step toward exposing the fundamental link that exists between on-line learning and boosting under the common umbrella of game theory.

To formalize the learning problem, let X be a finite set of instances, and let $\mathcal{H}$ be a finite space of hypotheses $h : X \to \{-1, +1\}$. These represent the fixed set of prediction rules against which the performance of the learning algorithm is to be compared. Let $c : X \to \{-1, +1\}$ be an unknown *target*, not necessarily in $\mathcal{H}$, defining the true and correct labels of each instance. This target may be entirely arbitrary. Even so, we are here implicitly introducing the mild assumption that the same instance never appear twice with different labels; this assumption is entirely unnecessary, but it does simplify the presentation.

Learning takes place in a sequence of rounds. On round $t = 1, \ldots, T$:

1. the learner observes an instance $x_t \in X$, selected arbitrarily;
2. the learner makes a randomized prediction $\hat{y}_t \in \{-1, +1\}$ of the label associated with x_t;
3. the learner observes the correct label $c(x_t)$.

The expected number of mistakes made by the learner is:

$$\mathrm{E}\left[\sum_{t=1}^{T} \mathbf{1}\{\hat{y}_t \neq c(x_t)\}\right] = \sum_{t=1}^{T} \Pr\left[\hat{y}_t \neq c(x_t)\right]. \tag{6.22}$$

Note that all expectations and probabilities in this context are taken with respect to the learner's own randomization, *not* the presentation of examples and labels

which need not be random in any way. The number of mistakes made by any fixed hypothesis h is equal to:

$$\sum_{t=1}^{T} \mathbf{1}\{h(x_t) \neq c(x_t)\}\,.$$

The goal of the learner is to minimize the expected number of mistakes made by the learner relative to the number of mistakes made by the *best* hypothesis in the space $\mathcal{H}$, determined in hindsight, that is,

$$\min_{h \in \mathcal{H}} \sum_{t=1}^{T} \mathbf{1}\{h(x_t) \neq c(x_t)\}\,.$$

Thus, we ask that the learner perform well whenever the target c is "close" to any one of the hypotheses in $\mathcal{H}$.

It is straightforward now to *reduce* the on-line prediction problem to a special case of the repeated game problem, that is, to show how an algorithm for repeated game-playing can be used as a "subroutine" to solve the on-line prediction problem. In this reduction, the environment's choice of a column will correspond to a choice of an instance $x \in X$ that is presented to the learner on a given iteration, while the learner's choice of a row will correspond to choosing a specific hypothesis $h \in \mathcal{H}$ which is then used to predict the label $h(x)$. More specifically, we define a game matrix that has $|\mathcal{H}|$ rows, indexed by $h \in \mathcal{H}$, and $|X|$ columns, indexed by $x \in X$. The matrix entry associated with hypothesis (row) h and instance (column) x is defined to be

$$\mathbf{M}(h, x) \doteq \mathbf{1}\{h(x) \neq c(x)\} = \begin{cases} 1 & \text{if } h(x) \neq c(x) \\ 0 & \text{otherwise.} \end{cases}$$

Thus, $\mathbf{M}(h, x)$ is 1 if and only if h disagrees with the target c on instance x. We call this the *mistake matrix*.

To derive an on-line learning algorithm, we apply MW to the mistake matrix $\mathbf{M}$. The reduction, shown schematically in Figure 6.1, creates an intermediary between MW and the on-line prediction problem. On each round, P_t from MW and the selected instance x_t are used to compute a prediction $\hat{y}_t$. Then after receiving $c(x_t)$, the matrix values $\mathbf{M}(\cdot, Q_t)$ can be computed and passed to MW for a suitable choice of Q_t.

More precisely, the distribution P_1 is first initialized by MW to be uniform over the hypotheses in $\mathcal{H}$. Next, on each round $t = 1, \ldots, T$, the on-line learning algorithm based on this reduction does the following:

1. receive instance $x_t \in X$;

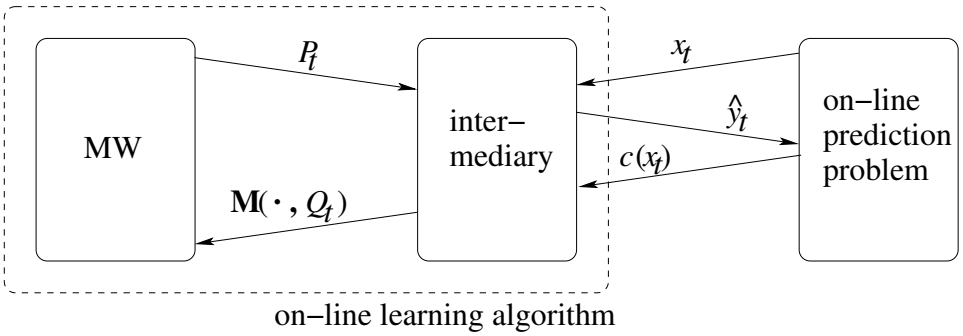

Figure 6.1: A schematic diagram of the reduction showing how MW can be used in the context of on-line prediction.

2. choose $h_t \in \mathcal{H}$ randomly according to the distribution P_t computed by MW;
3. predict $\hat{y}_t = h_t(x_t)$;
4. receive $c(x_t)$;
5. let Q_t be the pure strategy concentrated on x_t, and compute $\mathbf{M}(h, Q_t) = \mathbf{M}(h, x_t) = \mathbf{1}\{h(x_t) \neq c(x_t)\}$ for all $h \in \mathcal{H}$;
6. compute distribution P_{t+1} using algorithm MW; this reduces to the following update rule:

$$P_{t+1}(h) = \frac{P_t(h)}{Z_t} \times \begin{cases} e^{-\eta} & \text{if } h(x_t) \neq c(x_t) \\ 1 & \text{otherwise} \end{cases}$$

where Z_t is a normalization constant.

For the analysis, note that

$$\begin{aligned} \mathbf{M}(P_t, x_t) &= \sum_{h \in \mathcal{H}} P_t(h)\mathbf{M}(h, x_t) \\ &= \Pr_{h \sim P_t}[h(x_t) \neq c(x_t)] \\ &= \Pr[\hat{y}_t \neq c(x_t)]. \end{aligned} \tag{6.23}$$

By a direct application of Corollary 6.4 (for an appropriate choice of η), we have

$$\sum_{t=1}^{T} \mathbf{M}(P_t, x_t) \leq \min_{h \in \mathcal{H}} \sum_{t=1}^{T} \mathbf{M}(h, x_t) + O\left(\sqrt{T \ln |\mathcal{H}|}\right).$$

Rewriting using the definition of $\mathbf{M}$ and Eqs. (6.22) and (6.23) gives

$$\mathrm{E}\left[\sum_{t=1}^{T}\mathbf{1}\{\hat{y}_t \neq c(x_t)\}\right] \leq \min_{h\in\mathcal{H}}\sum_{t=1}^{T}\mathbf{1}\{h(x_t) \neq c(x_t)\} + O\left(\sqrt{T\ln|\mathcal{H}|}\right). \quad (6.24)$$

Thus, the expected number of mistakes made by the learner cannot exceed the number of mistakes made by the best hypothesis in $\mathcal{H}$ by more than $O\left(\sqrt{T\ln|\mathcal{H}|}\right)$. Equivalently, dividing both sides by T, we have

$$\mathrm{E}\left[\frac{1}{T}\sum_{t=1}^{T}\mathbf{1}\{\hat{y}_t \neq c(x_t)\}\right] \leq \min_{h\in\mathcal{H}}\frac{1}{T}\sum_{t=1}^{T}\mathbf{1}\{h(x_t) \neq c(x_t)\} + O\left(\sqrt{\frac{\ln|\mathcal{H}|}{T}}\right).$$

Since the last term vanishes as T becomes large, this says that the proportion of rounds where a mistake is made by the algorithm becomes very close to the best possible among all $h \in \mathcal{H}$.

These results can be straightforwardly generalized in many ways, for instance, to any bounded "loss" function (such as square loss rather than zero-one mistake loss), or to a setting in which the learner attempts to achieve performance comparable to that of the best among a (possibly changing) set of "experts" rather than a fixed set of hypotheses.

6.4 Boosting

Finally, we come to boosting, which we study here in a simplified form. We will see now how boosting is a special case of the general game-playing set-up of this chapter, and how this view leads not only to a (re-)derivation of an algorithm for boosting, but also to new insights into the very nature of the boosting problem.

As in Section 6.3, let X be a space of instances (typically, in this setting, the training set), $\mathcal{H}$ a space of (weak) hypotheses, and c some unknown target or labeling function, used here, again, for simplicity of presentation. We assume the availability of a weak learning algorithm such that, for some $\gamma > 0$, and for any distribution D over the set X, the algorithm is able to find a hypothesis $h \in \mathcal{H}$ with error at most $1/2 - \gamma$ with respect to the distribution D; this is the empirical γ-weak learning assumption of Section 2.3.3.

To review, in boosting, the weak learning algorithm is run many times on many distributions, and the selected weak hypotheses are combined into a final hypothesis whose error should be small, or even zero. Thus, boosting proceeds in rounds. On round $t = 1, \ldots, T$:

1. the booster constructs a distribution D_t on X which is passed to the weak learner;

2. the weak learner produces a hypothesis $h_t \in \mathcal{H}$ with error at most $1/2 - \gamma$:

$$\Pr_{x \sim D_t} [h_t(x) \neq c(x)] \leq \tfrac{1}{2} - \gamma.$$

After T rounds, the weak hypotheses $h_1, \ldots, h_T$ are combined into a final hypothesis H. As we know, the important issues for designing a boosting algorithm are: (1) how to choose distributions D_t, and (2) how to combine the h_t's into a final hypothesis.

6.4.1 Boosting and the minmax theorem

Before deriving our boosting algorithm, let us step back for a moment to consider the relationship between the mistake matrix $\mathbf{M}$ used in Section 6.3 and the minmax theorem. This relationship will turn out to be highly relevant to the design and understanding of the boosting algorithm that will follow.

Recall that the mistake matrix $\mathbf{M}$ has rows and columns indexed by hypotheses and instances, respectively, and that $\mathbf{M}(h, x) = 1$ if $h(x) \neq c(x)$ and is 0 otherwise. Assuming empirical γ-weak learnability as above, what does the minmax theorem say about $\mathbf{M}$? Suppose that the value of $\mathbf{M}$ is v. Then, together with Eqs. (6.3) and (6.4), the minmax theorem tells us that

$$\begin{aligned} \min_P \max_{x \in X} \mathbf{M}(P, x) &= \min_P \max_Q \mathbf{M}(P, Q) \\ &= v \\ &= \max_Q \min_P \mathbf{M}(P, Q) \\ &= \max_Q \min_{h \in \mathcal{H}} \mathbf{M}(h, Q). \end{aligned} \tag{6.25}$$

Note that, by $\mathbf{M}$'s definition,

$$\mathbf{M}(h, Q) = \Pr_{x \sim Q} [h(x) \neq c(x)].$$

Therefore, the right hand part of Eq. (6.25) says that there exists a distribution Q^* on X such that for every hypothesis h, $\mathbf{M}(h, Q^*) = \Pr_{x \sim Q^*} [h(x) \neq c(x)] \geq v$. However, because we assume γ-weak learnability, there must exist a hypothesis h such that

$$\Pr_{x \sim Q^*} [h(x) \neq c(x)] \leq \tfrac{1}{2} - \gamma.$$

Combining these facts gives that $v \leq 1/2 - \gamma$.

On the other hand, the left part of Eq. (6.25) implies that there exists a distribution P^* over the hypothesis space $\mathcal{H}$ such that for every $x \in X$:

$$\Pr_{h \sim P^*} [h(x) \neq c(x)] = \mathbf{M}(P^*, x) \leq v \leq \tfrac{1}{2} - \gamma < \tfrac{1}{2}. \tag{6.26}$$

In words, this says that every instance x is misclassified by less than $1/2$ of the hypotheses, as weighted by P^*. That is, a weighted majority vote classifier defined over $\mathcal{H}$ in which each hypothesis $h \in \mathcal{H}$ is assigned weight $P^*(h)$ will correctly classify all of the instances x; in symbols,

$$c(x) = \text{sign}\left(\sum_{h \in \mathcal{H}} P^*(h)\ h(x)\right)$$

for all $x \in X$. Thus, the weak learning assumption, together with the minmax theorem, implies that the target c must be functionally equivalent (on X) to some weighted majority of hypotheses in $\mathcal{H}$.

This reasoning tells us something even stronger about the *margins* for this weighted majority vote. Recall from Chapter 5 that the margin of an example is the difference between the weighted fraction of hypotheses voting for the correct label and the weighted fraction voting for an incorrect label. In this case, by Eq. (6.26), that difference will be at least

$$\left(\tfrac{1}{2} + \gamma\right) - \left(\tfrac{1}{2} - \gamma\right) = 2\gamma.$$

That is, the minimum margin over all examples will be at least 2γ. This is essentially the same result as in Section 5.4.3 where it was argued that empirical γ-weak learnability implies (and is in fact equivalent to) linear separability with margin 2γ, an important example of the tight relationship between edges and margins. Now, within a game-theoretic context, we can see that they are both manifestations of the value of a very natural game, and that their close connection is a direct and immediate consequence of the minmax theorem.

6.4.2 Idea for boosting

So the assumption of empirical γ-weak learnability implies that the target c can be computed exactly as a weighted majority of hypotheses in $\mathcal{H}$. Moreover, the weights used in this function (defined by distribution P^* above) are not just any old weights, but rather are a minmax strategy for the game $\mathbf{M}$. This is the basis of our boosting algorithm, namely, the idea of fitting the target labels c by approximating the weights P^* of this function. Since these weights are a minmax strategy of the game $\mathbf{M}$, our hope is to apply the method described in Section 6.2 for approximately solving a game using the MW algorithm.

The problem is that the resulting algorithm does not fit the boosting model if applied to the mistake matrix $\mathbf{M}$: Recall that on each round, algorithm MW computes a distribution over the rows of the game matrix (hypotheses, in the case

of matrix $\mathbf{M}$). However, in the boosting model, we want to compute on each round a distribution over instances (columns of $\mathbf{M}$).

Since we have an algorithm which computes distributions over rows, but need one that computes distributions over columns, the obvious solution is to reverse the roles of rows and columns. This is exactly the approach that we follow. That is, rather than using game $\mathbf{M}$ directly, we construct the *dual* of $\mathbf{M}$ which is the identical game except that the roles of the row and column players have been switched.

Constructing the dual $\mathbf{M}'$ of a game $\mathbf{M}$ is straightforward. First, we need to reverse row and column so we take the transpose $\mathbf{M}^\top$. This, however, is not enough since the column player of $\mathbf{M}$ wants to maximize the outcome, but the row player of $\mathbf{M}'$ wants to minimize the outcome (loss). Therefore, we also need to reverse the meaning of minimum and maximum which is easily done by negating the matrix yielding $-\mathbf{M}^\top$. Finally, to adhere to our convention of losses being in the range $[0, 1]$, we add the constant 1 to every outcome, which has no effect on the game. Thus, the dual $\mathbf{M}'$ of $\mathbf{M}$ is simply

$$\mathbf{M}' = \mathbf{1} - \mathbf{M}^\top \tag{6.27}$$

where $\mathbf{1}$ is an all 1's matrix of the appropriate dimensions.

In the case of the mistake matrix $\mathbf{M}$, the dual now has $|X|$ rows and $|\mathcal{H}|$ columns indexed by instances and hypotheses, respectively, and each entry is

$$\mathbf{M}'(x, h) \doteq 1 - \mathbf{M}(h, x) = \mathbf{1}\{h(x) = c(x)\} = \begin{cases} 1 & \text{if } h(x) = c(x) \\ 0 & \text{otherwise.} \end{cases}$$

Note that any minmax strategy of the game $\mathbf{M}$ becomes a maxmin strategy of the game $\mathbf{M}'$. Therefore, whereas before we were interested in finding an approximate minmax strategy of $\mathbf{M}$, we are now interested in finding an approximate maxmin strategy of $\mathbf{M}'$.

We can now apply algorithm MW to game matrix $\mathbf{M}'$ since, by the results of Section 6.2.5, this will lead to the construction of an approximate maxmin strategy. As shown in Figure 6.2, the reduction now creates an intermediary between MW and the weak learning algorithm, on each round using the distribution P_t received from MW to compute D_t, then using the hypothesis h_t received in response from the weak learner to compute $\mathbf{M}'(\cdot, Q_t)$ for an appropriate choice of Q_t. In more detailed terms, the reduction proceeds as follows: The distribution P_1 is initialized as in MW, that is, uniform over X. Then on each round of boosting $t = 1, \ldots, T$, the boosting algorithm under this reduction does the following:

1. set $D_t = P_t$ and pass D_t to the weak learning algorithm;

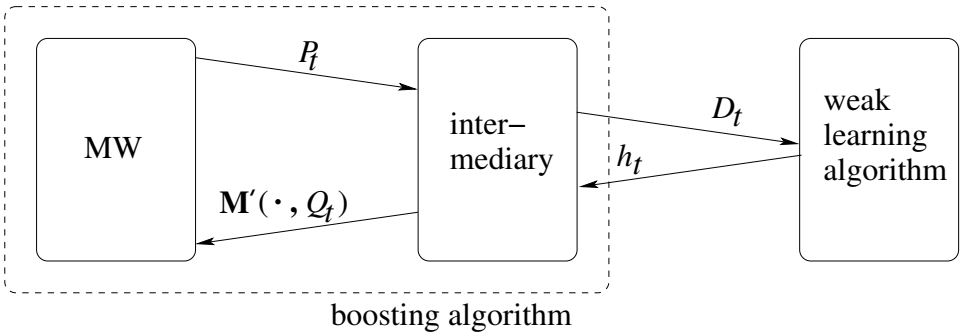

Figure 6.2: A schematic diagram of the reduction showing how MW can be used to derive a boosting algorithm.

2. the weak learner returns a hypothesis h_t satisfying

$$\Pr_{x \sim D_t}[h_t(x) = c(x)] \geq \tfrac{1}{2} + \gamma;$$

3. let Q_t be the pure strategy concentrated on h_t and compute $\mathbf{M}'(x, Q_t) = \mathbf{M}'(x, h_t) = \mathbf{1}\{h_t(x) = c(x)\}$ for all $x \in X$;

4. compute the new distribution P_{t+1} using algorithm MW; that is,

$$P_{t+1}(x) = \frac{P_t(x)}{Z_t} \times \begin{cases} e^{-\eta} & \text{if } h_t(x) = c(x) \\ 1 & \text{otherwise} \end{cases}$$

where Z_t is a normalization constant.

Our goal, again, is to find an approximate maxmin strategy of $\mathbf{M}'$ using the method of approximately solving a game given in Section 6.2.5. According to that method, on each round t, Q_t may be a pure strategy h_t and should be chosen to maximize

$$\mathbf{M}'(P_t, h_t) = \sum_x P_t(x)\mathbf{M}'(x, h_t) = \Pr_{x \sim P_t}[h_t(x) = c(x)].$$

In other words, h_t should have maximum accuracy with respect to distribution P_t. This is exactly the goal of the weak learner. (Although it is not guaranteed to succeed in finding the *best* h_t, finding one of accuracy $1/2 + \gamma$ turns out to be sufficient for our purposes.)

So the weak learner aims to maximize the weighted accuracy of the weak hypotheses, as is natural, but in this game-theoretic setting, the goal of the booster

is exactly the opposite, namely, to choose distributions D_t which make it as hard as possible for the weak learner to find an accurate hypothesis. Thus, although we have said informally that boosting focuses on hard examples, we see now that it would be more accurate to say that boosting focuses on finding the hardest *distribution* over examples.

Finally, the method from Section 6.2.5 suggests that $\overline{Q} = (1/T)\sum_{t=1}^{T} Q_t$ is an approximate maxmin strategy, and we know that the target c is equivalent to a majority of the hypotheses if weighted by a maxmin strategy of $\mathbf{M}'$. Since Q_t is in our case concentrated on pure strategy (hypothesis) h_t, this leads us to choose a final hypothesis H which is the (simple) majority of $h_1, \ldots, h_T$:

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} h_t(x)\right).$$

Note that as a side effect,

$$\overline{P} = \frac{1}{T}\sum_{t=1}^{T} P_t = \frac{1}{T}\sum_{t=1}^{T} D_t$$

will likewise converge to an approximate minmax solution of $\mathbf{M}'$. Thus, the (average of the) distributions D_t computed by boosting also have a natural game-theoretic interpretation.

6.4.3 Analysis

Indeed, the resulting boosting procedure will compute a final hypothesis H that is functionally equivalent to c for sufficiently large T. We show in this section how this follows from Corollary 6.4.

As noted earlier, for all t, by our assumption of γ-weak learnability,

$$\mathbf{M}'(P_t, h_t) = \Pr_{x\sim P_t}[h_t(x) = c(x)] \geq \tfrac{1}{2} + \gamma.$$

By Corollary 6.4, for an appropriate choice of η, this implies that

$$\tfrac{1}{2} + \gamma \leq \frac{1}{T}\sum_{t=1}^{T} \mathbf{M}'(P_t, h_t) \leq \min_{x\in X} \frac{1}{T}\sum_{t=1}^{T} \mathbf{M}'(x, h_t) + \Delta_T, \tag{6.28}$$

and so, for all x,

$$\frac{1}{T}\sum_{t=1}^{T} \mathbf{M}'(x, h_t) \geq \tfrac{1}{2} + \gamma - \Delta_T > \tfrac{1}{2} \tag{6.29}$$

where the last inequality holds for sufficiently large T (specifically, when $\Delta_T < \gamma$). Note that, by definition of $\mathbf{M}'$, $\sum_{t=1}^{T} \mathbf{M}'(x, h_t)$ is exactly the number of hypotheses h_t which agree with c on instance x. Therefore, in words, Eq. (6.29) says that more than half the hypotheses h_t are correct on x. This means, by definition of H, that $H(x) = c(x)$ for all x.

For the above to hold, we need only that $\Delta_T < \gamma$, which will be the case for $T = \Omega(\ln |X|/\gamma^2)$. Moreover, by the same argument as in Section 6.4.2 applied to Eq. (6.29), we see that every x will have margin at least $2\gamma - 2\Delta_T$. Thus, as T gets large and Δ_T approaches zero, a minimum margin of at least 2γ is obtained asymptotically. Together with the discussion in Section 5.4.3, this shows that the optimal margin is achieved asymptotically, assuming that the "best" (minimum weighted error) weak hypothesis h_t is chosen on every round.

When the game-playing subroutine MW is "compiled out," and when η is replaced by 2α (for cosmetic compatibility with earlier notation), the result of our reduction is a simplified version of AdaBoost (Algorithm 1.1) in which all of the α_t's are set equal to the fixed parameter α. We refer to this simplified algorithm as α-Boost, although it has sometimes also been called ε-boosting or ε-AdaBoost.

The analysis above shows that α-Boost converges to the maximum margin combined classifier when α and T are chosen together in concert according to the dictates of Corollary 6.4, suggesting that T must be delicately tuned as a function of α (or vice versa). In fact, a slightly different analysis shows that the same result holds true if α is simply chosen to be "very small" and the algorithm is then run for a "long time" (with no danger of running for too long). In particular, instead of using Corollary 6.4 in Eq. (6.28), we can apply Corollary 6.3. This gives

$$\tfrac{1}{2} + \gamma \leq \frac{1}{T}\sum_{t=1}^{T} \mathbf{M}'(P_t, h_t) \leq a_\eta \min_{x \in X} \frac{1}{T}\sum_{t=1}^{T} \mathbf{M}'(x, h_t) + \frac{c_\eta \ln m}{T}$$

where $\eta = 2\alpha$. Rearranging gives

$$\begin{aligned}
\min_{x \in X} \frac{1}{T}\sum_{t=1}^{T} \mathbf{M}'(x, h_t) &\geq \frac{1}{a_\eta}\left[\left(\tfrac{1}{2} + \gamma\right) - \frac{c_\eta \ln m}{T}\right] \\
&= \left(\tfrac{1}{2} + \gamma\right) - \left(1 - \frac{1 - e^{-2\alpha}}{2\alpha}\right)\left(\tfrac{1}{2} + \gamma\right) - \frac{\ln m}{2\alpha T} \\
&\geq \left(\tfrac{1}{2} + \gamma\right) - \alpha\left(\tfrac{1}{2} + \gamma\right) - \frac{\ln m}{2\alpha T}
\end{aligned}$$

where the last inequality uses the approximation $e^{-z} \leq 1 - z + z^2/2$ for all $z \geq 0$. Thus, by similar arguments as before, all examples x will have margin at least

$$2\gamma - \alpha(1 + 2\gamma) - \frac{\ln m}{\alpha T}.$$

When T is very large, the rightmost term becomes negligible, so that asymptotically the margins come within $\alpha(1 + 2\gamma) \leq 2\alpha$ of 2γ, the best possible margin for the given γ-weak learning assumption (see Section 5.4.3). Thus, this argument shows that a combined classifier can be found with a minimum margin that is arbitrarily close to optimal by using an appropriately small choice of α, followed by a long run of the algorithm (with specific rates of convergence as given above).

So as an alternative to AdaBoost or the variants given in Section 5.4.2, we see that the simpler algorithm α-Boost can be used for the purpose of maximizing the minimum margin. However, in addition to the caveats of Section 5.4.2, we expect this procedure in practice to be slow since α must be small, and T must be correspondingly large.

6.5 Application to a "mind-reading" game

We end this chapter with a brief description of an application of these ideas to a simple game called *penny-matching*, or *odds and evens*. One player is designated the "evens" player and the other is "odds." On every round of play, they both choose and then simultaneously reveal a single bit, either $+$ or $-$ (which we sometimes identify with $+1$ and -1). If the two bits match, then the evens player wins; otherwise, the odds player wins. The game is typically played for multiple rounds.

As in Rock-Paper-Scissors, the penny-matching game incorporates elements of a mind-reading contest in the sense that each player attempts to predict what the other player will do, and to act accordingly, while simultaneously trying to behave unpredictably. Of course, the players can in principle choose their bits entirely at random (which would be the minmax strategy for the game); however, unless provided with an external source of randomness, such as an actual coin or a computer, humans turn out to be very bad at behaving in a truly random fashion (we will see experimental evidence for this below). Moreover, players who can successfully discern their opponent's intentions will have a much better chance of winning.

In the 1950's, David Hagelbarger and later Claude Shannon created learning machines to play this game against a human in an early exploration of how to make "intelligent" computing devices. In those days, this meant literally building a machine—Figure 6.3 shows a schematic diagram of Hagelbarger's, which he called a "sequence extrapolating robot." (Shannon referred to his as a "mind-reading (?) machine" (*sic*).) Both their designs were very simple, keeping track of how the human has behaved in similar circumstances and then acting accordingly based on this history. On each round, their machines would consider the current "state of play" and how the human had previously behaved when this identical state

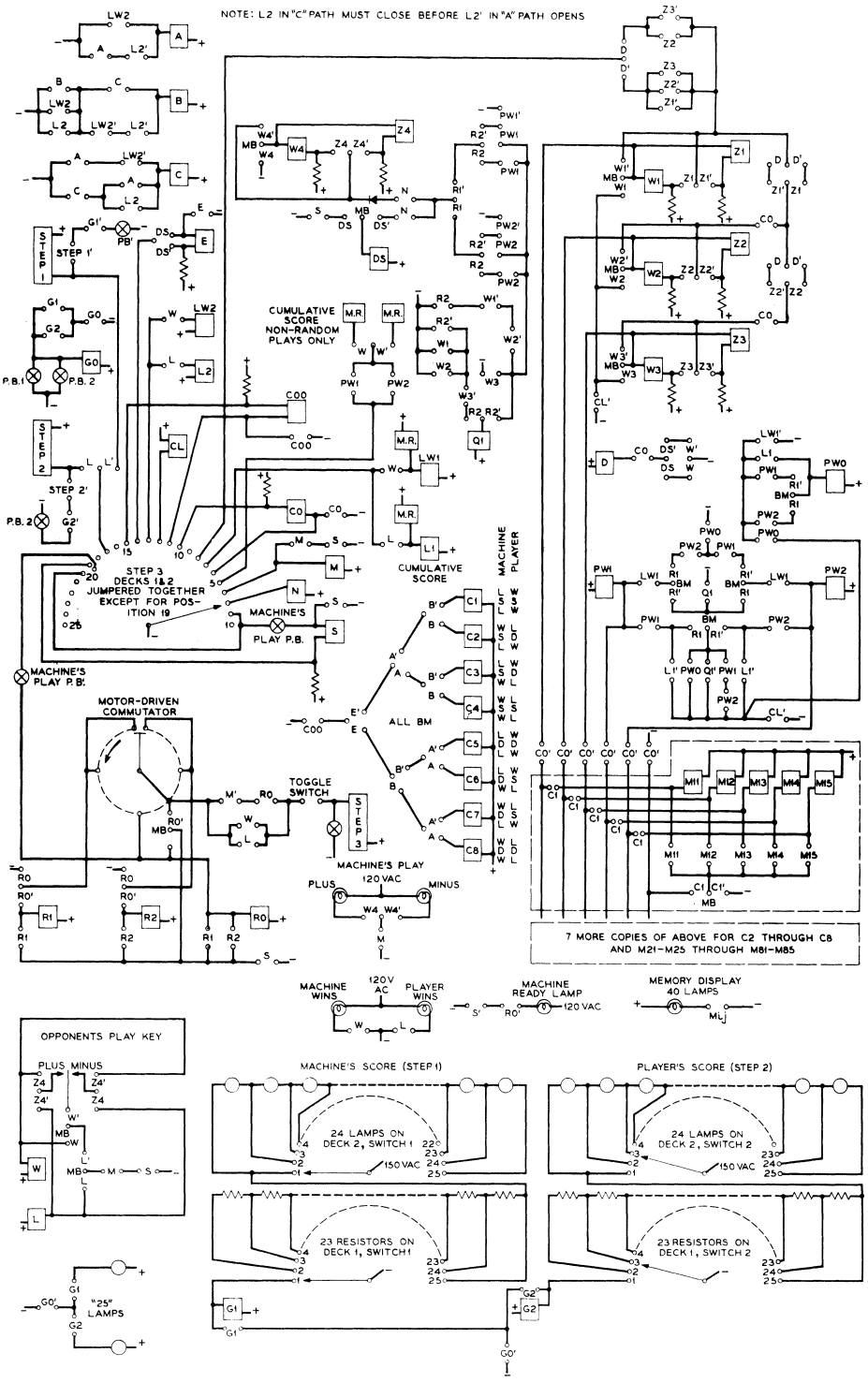

Figure 6.3: The circuit design of Hagelbarger's machine for playing penny-matching. [Copyright ©1956 IRE (now IEEE). Reprinted, with permission, from *IRE Transactions on Electronic Computers*, EC-5(1):1–7, March, 1956.]

of play had been encountered, formulating a prediction of the human's next play accordingly. In their machines, the notion of state of play was limited only to what happened on the last two rounds, specifically, whether the human won or lost the last round; whether the human won or lost the time before that; and whether the human played differently or the same on the last two rounds.

Here, we describe a more sophisticated approach to playing this game based on the on-line prediction framework of Section 6.3. As we have discussed, the essential problem in this game is that of predicting what one's opponent will do next. Moreover, these predictions must be made in an on-line fashion. And regarding the "data" as random in this adversarial setting seems entirely unreasonable. Given these attributes of the problem, the on-line learning model seems to be especially well-suited.

Recall that, in on-line prediction, on each round t, the learner receives an instance x_t, formulates a prediction $\hat{y}_t$, and observes an outcome, or label, $c(x_t)$ which we henceforth denote by y_t. The learner's goal is to minimize the number of mistakes, that is, rounds in which $\hat{y}_t \neq y_t$. To cast the penny-matching game in these terms, we first identify the learner with the "evens" player whose goal is to match the human opponent's bits. On round t, we identify y_t with the human's chosen bit on that round, and we take the learner's prediction $\hat{y}_t$ to be its own chosen bit. Then the learner loses the round if and only if $\hat{y}_t \neq y_t$. In other words, in this set-up, minimizing mistakes in on-line prediction is the same as minimizing the number of rounds lost in penny-matching.

As presented in Section 6.3, given an instance x_t, an on-line learning algorithm formulates its own prediction $\hat{y}_t$ based on the predictions $h(x_t)$ made by the rules h in a space $\mathcal{H}$. In the current setting, we take the instance x_t to be the entire history up until (but not including) round t; specifically, this means all of the plays made by both players on the first $t-1$ rounds. Given this history, each prediction rule h makes its own prediction of what the human will do next.

The algorithm presented in Section 6.3 gives a technique for combining the predictions of the rules in $\mathcal{H}$ so that the composite predictions $\hat{y}_t$ will be almost as good as those of the best rule in the space.

So all that remains is to choose a set $\mathcal{H}$ of predictors. Our bounds suggest that $\mathcal{H}$ can be rather large, and we only need to anticipate that one of the rules will be good. Clearly, there are many sorts of predictors we might imagine, and here we describe just one of many possible approaches.

As was done by Hagelbarger and Shannon, it seems especially natural to consider predictors that take into account the recent past. For instance, suppose the human tends to alternate between plays of $-$ and $+$ leading to runs like this:

$$- + - + - + - + - + - + - + - \cdots$$

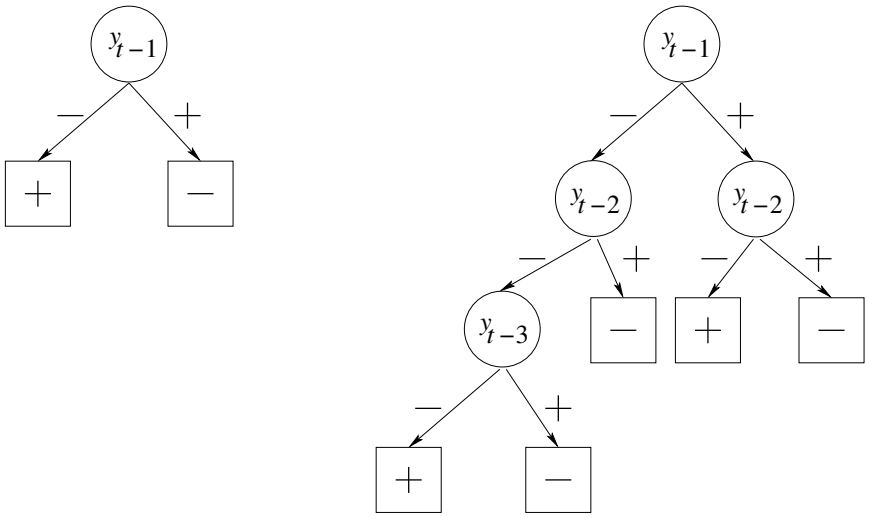

Figure 6.4: Two example context trees for predicting the human's next bit y_t based on those previously played $y_1, \ldots, y_{t-1}$. The trees are evaluated as in Section 1.3, which in this case means working our way backward through the history of played bits until a leaf node is reached providing the tree's prediction for the current context.

Such a pattern can be captured by a rule that says, "if the last bit was $-$, then predict $+$ for the current round; and if the last bit was $+$, then predict $-$." This simple rule can be represented by a decision tree like the stubby one on the left of Figure 6.4, where the nodes indicate the bit to be tested to determine which branch to follow, and the leaves provide the rule's prediction for the next bit.

Suppose now that the human instead tends to create more complicated patterns like this one:

$$++---+++---+++---\cdots$$

This pattern can be similarly captured by a decision tree as shown on the right of Figure 6.4. For instance, the tree tells us that if the last bit was $+$ and the one before that was $-$ then the next bit should be predicted $+$. But if the last two bits were $-$, then we actually need to look one more bit back to arrive at a prediction, according to this rule.

Note that, although we have motivated these rules with simple patterns like the ones above, such rules need not give perfect predictions to be useful in our setting. It is enough that they capture general tendencies that enable them to make predictions that are better than random.

Such decision trees are called *context trees* since each prediction is formulated based on the context of the recent past where we work our way back in time until the rule has enough information to make a prediction. The trees we have considered so far only take into account how the human has played, but in general, we may also wish to consider other aspects of the recent past, for instance, who won the round, and whether or not the human's predictions changed from one round to the next. Indeed, rules based on Hagelbarger and Shannon's "state of play" could be put into the form of such a tree as well.

So the idea is to identify the rules used by our on-line prediction algorithm with such context trees. This leads, of course, to the question of *which* trees to include in our rule space. To answer this, we begin by fixing an order in which the past is probed. For instance, the trees above, on round t, first test the last bit y_{t-1} played by the human, then the preceding bit y_{t-2}, then y_{t-3}, etc. This means that the trees we consider will all test the value of y_{t-1} at the root, then all nodes at the next level down will test y_{t-2}, and so on. The point is that the tests associated with particular nodes are fixed and the same for all trees in the family. (Although we focus on there being just a single, fixed ordering of the tests, these methods can be immediately generalized to the case in which there are instead a small number of orderings considered, each defining its own family of trees.)

Subject to this restriction on the ordering of the tests, we can now consider including in $\mathcal{H}$ *all possible* context trees, meaning all possible topologies, or ways of cutting off the tree, and all possible ways of labeling the leaves. For instance, Figure 6.4 shows two possible trees that are consistent with the specific restrictions we described above. In general, there will be an exponential number of such trees since there are exponentially many tree topologies and exponentially many leaf labelings to consider. As previously mentioned, this huge number of rules is not necessarily a problem in terms of performance since the bounds (such as in Eq. (6.24)) are only logarithmic in $|\mathcal{H}|$. Moreover, it is not implausible to expect at least one such rule to capture the kinds of patterns typically selected by humans.

On the other hand, computationally, having a very large number of rules is prohibitively expensive since a naive implementation of this algorithm requires space and time-per-round that are linear in $|\mathcal{H}|$. Nevertheless, for this particular well-structured family of rules, it turns out that the on-line learning algorithm of Section 6.3 can be implemented extremely efficiently both in terms of time and space. This is because the required tree-based computations collapse into a form in which a kind of dynamic programming can be applied.

These ideas were implemented into a "mind-reading game" that is publicly available on the internet (`seed.ucsd.edu/~mindreader`) in which the computer and the human play against each other until one player has won a hundred rounds.

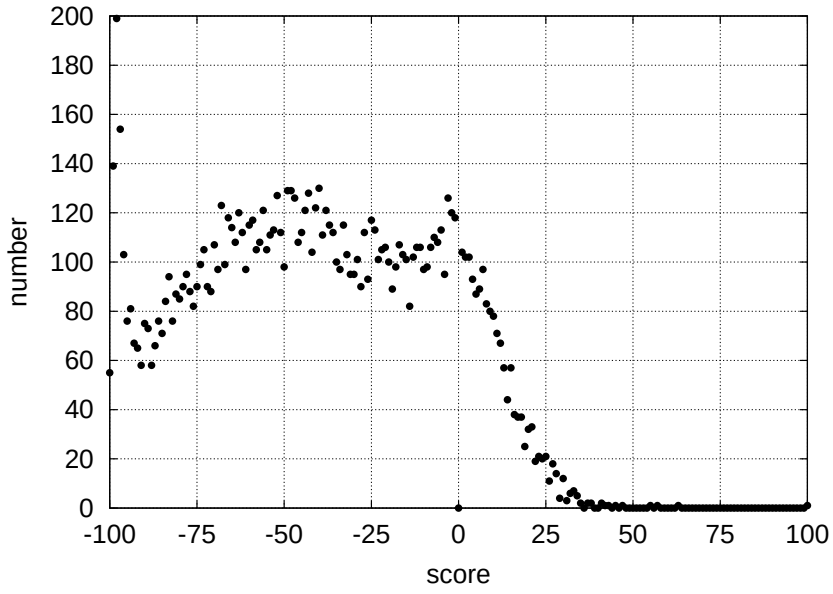

Figure 6.5: A histogram of the number of games played (out of 11,882) for each possible final score between -100 and 100, where the score is the number of rounds won by the human minus the number won by the computer, so games with negative scores were won by the computer. (No games had a score of zero since ties are not possible under the rules of this game.)

Figure 6.5 shows a histogram of the final scores for 11,882 games recorded between March, 2006 and June 2008. The score is the number of rounds won by the human minus the number won by the computer (so it is positive if and only if the human won the entire game). The figure shows that the computer usually wins, and often by a wide margin. In fact, the computer won 86.6% of these games. The average score of all the games was -41.0 with a median of -42, meaning that on half the games, the human had won 58 or fewer rounds by the time the computer had won 100. Of course, a purely random player would do much better than humans against the computer, necessarily winning 50% of the games, and achieving an average and median score of zero (in expectation).

A curious phenomenon revealed by this data is shown in Figure 6.6. Apparently, the faster humans play, the more likely they are to lose. Presumably, this is because faster play tends to be more predictable, often leading to rhythmic and patterned key-banging that the learning algorithm can quickly pick up on.

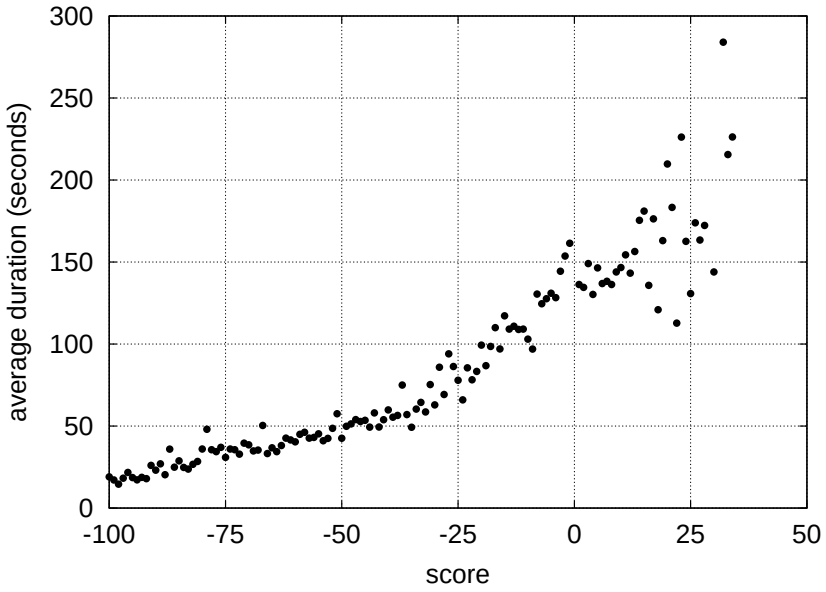

Figure 6.6: A plot of the average duration (in seconds) of the game compared to the final score of the game. For every possible score, a point is plotted whose x-value is the score, and whose y-value is the average of the durations of all games which ended with that particular final score. (No point was plotted for scores with fewer than five such games. Also, to mitigate the effect of outliers, the few games that lasted more than ten minutes were treated as if they had lasted exactly ten minutes.)

Summary

To summarize, we have seen in this chapter how AdaBoost (or at least, a simplified variant) can be viewed as a special case of a more general algorithm for solving games through repeated play. This has allowed us to understand AdaBoost more deeply, showing, for instance, that

1. the weights on the weak hypotheses in the combined classifier must converge to an approximate maxmin strategy for the (dual) mistake-matrix game associated with boosting;

2. the (average of the) distributions D_t over examples must converge to an approximate minmax strategy for this same game; and

3. the notions of edge and margin are intimately connected via the minmax theorem.

Moreover, we have seen how on-line learning is the dual problem of boosting.

Bibliographic notes

Our development of basic game theory in Section 6.1 is standard. Further background can be found in any of a number of introductory texts, such as refs. [103, 173, 178, 179]. The minmax theorem of Section 6.1.3 is due to von Neumann [174].

The algorithm, analysis and proof of the minmax theorem appearing in Section 6.2 are all taken from Freund and Schapire [94, 96] whose work is a direct generalization of Littlestone and Warmuth's [155]. Algorithms with this same "no-regret" property (also called "Hannan consistency" or "universal consistency"), whose loss is guaranteed to be not much worse than that of the best fixed strategy, date back to the 1950's with the work of Hannan [117] and Blackwell [23, 24]. Other methods include those of Foster and Vohra [86], and Hart and Mas-Colell [118], as well as Fudenberg and Levine [101] whose method of "exponential fictitious play" strongly resembles the Weighted Majority Algorithm.

The on-line prediction model studied in Section 6.3 was first considered by Littlestone and Warmuth [155], and Vovk [228], although its roots connect it with work in numerous other fields such as game theory and data compression. The Weighted Majority Algorithm and its analysis, as presented here, are originally due to Littlestone and Warmuth [155]. Its re-derivation is due to Freund and Schapire [94]. Better bounds than those presented here for the on-line prediction problem were obtained by Cesa-Bianchi et al. [45] and Vovk [228].

For further background on no-regret algorithms and on-line learning, see Cesa-Bianchi and Lugosi's excellent book [47]. A somewhat different perspective on learning and game theory is given in Fudenberg and Levine's book [102].

The connection between boosting and game playing described in Section 6.4 is due to Freund and Schapire [94]. However, from the very beginning, AdaBoost was linked with on-line learning, having been originally derived directly from a generalization of the Weighted Majority Algorithm called "Hedge" [95].

The α-Boost algorithm of Section 6.4.3 in which all of the α_t's are held fixed to a small constant α was suggested by Friedman [100]. The convergence and margin-maximizing properties of this algorithm were studied by Rosset, Zhu and Hastie [191], and by Zhang and Yu [235]. The proof given here is similar to the one given by Xi et al. [232].

Any game can be solved using linear programming, and conversely, it is known that the solution of any linear program can be obtained by solving an appropriate zero-sum game [62]. This equivalence points also to the close relationship between boosting and linear programming, and indeed, the problem of finding the maximum-margin classifier can be formulated as a linear program. This connec-

tion is studied in depth by Grove and Schuurmans [111], and Demiriz, Bennett and Shawe-Taylor [65].

A method for combining on-line learning and boosting—specifically, for running AdaBoost in an on-line fashion—is given by Oza and Russell [180].

Early machines for learning to play penny-matching, as in Section 6.5, were invented by Hagelbarger [115] and later by Shannon [212]. Figure 6.3 is reprinted from the former. The technique of combining the predictions of all possible context trees is due to Helmbold and Schapire [122] in a direct adaptation of Willems, Shtarkov and Tjalkens's method for weighting context trees [230]. The internet implementation was created by the authors with Anup Doshi.

The episode of *The Simpsons* quoted in Section 6.1.3 first aired on April 15, 1993 (episode #9F16).

Some of the exercises in this chapter are based on material from [62, 96, 153].

Exercises

In the exercises below, assume all game matrices have entries only in $[0, 1]$, except where noted otherwise.

6-1. Show that the minmax theorem (Eq. (6.6)) is false when working with pure strategies. In other words, give an example of a game $\mathbf{M}$ for which

$$\min_i \max_j \mathbf{M}(i,j) \neq \max_j \min_i \mathbf{M}(i,j).$$

6-2. Suppose MW is used to play against itself on an $m \times n$ game matrix $\mathbf{M}$. That is, on each round, the row player selects its mixed strategy P_t using MW, and the column player selects Q_t using another copy of MW applied to the dual matrix $\mathbf{M}'$ (Eq. (6.27)). Use Corollary 6.4, applied to these two copies of MW, to give an alternative proof of the minmax theorem. Also, show that $\overline{P}$ and $\overline{Q}$, as defined in Eq. (6.21), are approximate minmax and maxmin strategies.

6-3. What is the relationship between the value v of a game $\mathbf{M}$ and the value v' of its dual $\mathbf{M}'$ (Eq. (6.27))? In particular, if $\mathbf{M}$ is *symmetric* (that is, equal to its dual), what must its value be? Justify your answers.

6-4. Let $S = \langle x_1, \ldots, x_m \rangle$ be any sequence of m distinct points in $\mathcal{X}$. Referring to the definitions in Section 5.3, prove that

$$R_S(\mathcal{H}) \leq O\left(\sqrt{\frac{\ln |\mathcal{H}|}{m}}\right) \tag{6.30}$$

by applying the analysis in Section 6.3 to an appropriately constructed presentation of examples and labels. (Eq. (6.30) is the same as Eq. (5.22), but with possibly weaker constants.)

[*Hint:* Consider choosing all labels uniformly at random.]

6-5. The vMW algorithm is a variant of MW in which the parameter η varies from round-to-round. Let $u \in [0,1]$ be a given "estimate" of the value of the game $\mathbf{M}$. The vMW algorithm starts with an arbitrary initial strategy P_1. On each round t, the new mixed strategy P_{t+1} is computed from P_t as in Eq. (6.9) but with η replaced by η_t where

$$\eta_t \doteq \max\left\{0, \ln\left(\frac{u(1-\ell_t)}{(1-u)\ell_t}\right)\right\},$$

and where $\ell_t \doteq \mathbf{M}(P_t, Q_t)$. Let

$$\hat{Q} \doteq \frac{\sum_{t=1}^{T}\eta_t Q_t}{\sum_{t=1}^{T}\eta_t}.$$

For parts (a) and (b), assume $\ell_t \geq u$ for all t.

(a) Show that if $\mathbf{M}(\tilde{P}, \hat{Q}) \leq u$ then

$$\mathrm{RE}\left(\tilde{P} \parallel P_{T+1}\right) - \mathrm{RE}\left(\tilde{P} \parallel P_1\right) \leq -\sum_{t=1}^{T}\mathrm{RE}_b\left(u \parallel \ell_t\right).$$

(b) Show that

$$\Pr_{i\sim P_1}\left[\mathbf{M}(i,\hat{Q}) \leq u\right] = \sum_{i:\mathbf{M}(i,\hat{Q})\leq u} P_1(i) \leq \exp\left(-\sum_{t=1}^{T}\mathrm{RE}_b\left(u \parallel \ell_t\right)\right).$$

(c) Suppose the value v of the game is at most u, and suppose vMW is run with P_1 chosen to be the uniform distribution. For all $\varepsilon > 0$, show that the number of rounds t on which $\ell_t \geq u + \varepsilon$ cannot exceed

$$\frac{\ln m}{\mathrm{RE}_b\left(u \parallel u+\varepsilon\right)}.$$

(d) Show how AdaBoost can be derived from vMW and how its analysis in Theorem 3.1 follows as a special case of part (b). (You can assume that $\epsilon_t \leq 1/2$ on every round of AdaBoost.)

(e) Likewise, show how the version of AdaBoost given in Section 5.4.2 with α_t set as in Eq. (5.33) can be derived from vMW, and how the bound in Eq. (5.34) follows as a special case of part (b). (Assume $\gamma_t \geq \theta/2$ on every round t.)

(f) Explain in precise terms using the language of boosting what the implications are of part (c) for the boosting algorithms considered in parts (d) and (e).

6-6. In the on-line learning framework of Section 6.3, suppose the target c is linearly separable with margin θ; that is, suppose there exists a weight vector $\mathbf{w}$ on the classifiers in $\mathcal{H}$ such that $\|\mathbf{w}\|_1 = 1$, and for all $x \in X$,

$$c(x)\left(\sum_{h\in\mathcal{H}} w_h h(x)\right) \geq \theta$$

where $\theta > 0$ is known (but not $\mathbf{w}$). Derive and analyze an on-line algorithm that makes at most

$$O\left(\frac{\ln|\mathcal{H}|}{\theta^2}\right)$$

mistakes on any sequence of examples. Give explicit constants. [*Hint:* Use Ex. 6-5.]

6-7. In the on-line learning framework of Section 6.3, we assumed that each instance x_t is labeled by $c(x_t)$, where c is some unknown target function. This means that the same instance x cannot appear twice with different labels. Suppose we remove this assumption so that the "correct" label associated with each instance x_t is now allowed to be an arbitrary value $y_t \in \{-1, +1\}$. For this relaxed setting, show that the bound in Eq. (6.24), with $c(x_t)$ replaced by y_t, holds for an appropriate modification of the algorithm given in Section 6.3.

6-8. In the modified on-line learning setting of the last exercise, suppose that the instance space X is not too large, and that $\mathcal{H}$ is chosen to be all possible binary functions on X, that is, all functions $h : X \to \{-1, +1\}$. Since $|\mathcal{H}| = 2^{|X|}$, a naive implementation of the (modified) algorithm of Section 6.3 would require time and space that are exponential in $|X|$. Devise an alternative algorithm (1) which is equivalent in terms of its input-output behavior (so that it makes identical (randomized) predictions when given identical observations), but (2) whose space requirements are only linear in $|X|$, and whose per-round time requirements are even better. Show that your algorithm has both these properties.

6-9. This exercise explores a kind of equivalence that exists between linear programming and solving zero-sum games. A *linear program* is an optimization problem in which the goal is to maximize a linear objective function subject to linear inequality constraints. Thus, the problem has the following form:

$$\begin{aligned} &\text{maximize:} && \mathbf{c}\cdot\mathbf{x} \\ &\text{subject to:} && \mathbf{A}\mathbf{x} \leq \mathbf{b} \text{ and } \mathbf{x} \geq \mathbf{0}. \end{aligned} \tag{6.31}$$

Here, our aim is to solve for $\mathbf{x} \in \mathbb{R}^n$ given $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^m$ and $\mathbf{c} \in \mathbb{R}^n$. Also, in this exercise, we use inequality between vectors to mean component-wise inequality (that is, $\mathbf{u} \geq \mathbf{v}$ if and only if $u_i \geq v_i$ for all i).

(a) Show that the problem of solving a game, that is, finding a minmax strategy P^*, can be formulated as a linear program.

Every linear program in its *primal* form, as in Program (6.31), has a corresponding *dual* program over dual variables $\mathbf{y} \in \mathbb{R}^m$:

$$\begin{aligned} &\text{minimize:} && \mathbf{b} \cdot \mathbf{y} \\ &\text{subject to:} && \mathbf{A}^\top \mathbf{y} \geq \mathbf{c} \text{ and } \mathbf{y} \geq \mathbf{0}. \end{aligned} \tag{6.32}$$

(b) What is the dual of the linear program that was found in part (a)? In game-theoretic terms, what is the meaning of its solution?

Returning to the general linear program above and its dual (Programs (6.31) and (6.32)), a vector $\mathbf{x} \in \mathbb{R}^n$ is said to be *feasible* if $\mathbf{A}\mathbf{x} \leq \mathbf{b}$ and $\mathbf{x} \geq \mathbf{0}$; likewise, $\mathbf{y} \in \mathbb{R}^m$ is feasible if $\mathbf{A}^\top \mathbf{y} \geq \mathbf{c}$ and $\mathbf{y} \geq \mathbf{0}$.

(c) Show that if $\mathbf{x}$ and $\mathbf{y}$ are both feasible, then $\mathbf{c} \cdot \mathbf{x} \leq \mathbf{b} \cdot \mathbf{y}$. Further, if $\mathbf{x}$ and $\mathbf{y}$ are feasible and $\mathbf{c} \cdot \mathbf{x} = \mathbf{b} \cdot \mathbf{y}$, show that $\mathbf{x}$ and $\mathbf{y}$ are solutions of the primal and dual problems, respectively. [*Hint:* Consider $\mathbf{y}^\top \mathbf{A}\mathbf{x}$.]

Consider the $(m + n + 1) \times (m + n + 1)$ game matrix

$$\mathbf{M} \doteq \begin{pmatrix} \mathbf{0} & \mathbf{A}^\top & -\mathbf{c} \\ -\mathbf{A} & \mathbf{0} & \mathbf{b} \\ \mathbf{c}^\top & -\mathbf{b}^\top & \mathbf{0} \end{pmatrix}.$$

Here, entries of $\mathbf{M}$ that are matrices or vectors stand for entire blocks of entries, and each $\mathbf{0}$ is a matrix composed of all 0's of the appropriate size. (Note that we have here dropped our convention of requiring that all entries of $\mathbf{M}$ be in $[0, 1]$.)

(d) What is the value of the game $\mathbf{M}$?

(e) Let P^* be a minmax solution of the game $\mathbf{M}$ which, being a vector in $\mathbb{R}^{m+n+1}$, can be written in the form

$$\begin{pmatrix} \mathbf{x} \\ \mathbf{y} \\ z \end{pmatrix}$$

where $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{y} \in \mathbb{R}^m$, and $z \in \mathbb{R}$. Show that if $z \neq 0$ then $\mathbf{x}/z$ and $\mathbf{y}/z$ are solutions of the primal and dual problems, respectively.