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).

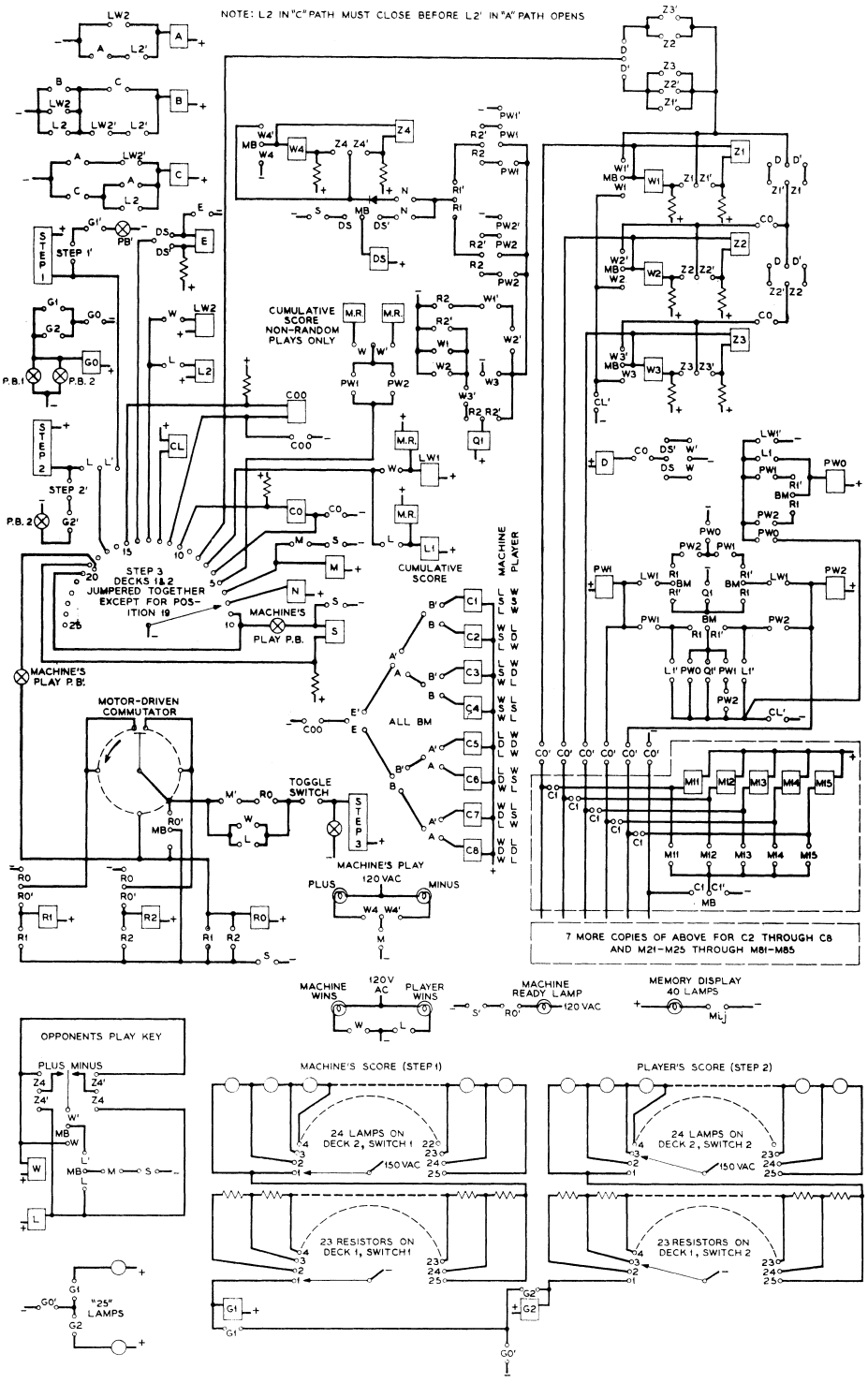
So as an alternative to AdaBoost or the variants given in Section 5.4.2, we see that the simpler algorithm α-Boost can be used for the purpose of maximizing the minimum margin. However, in addition to the caveats of Section 5.4.2, we expect this procedure in practice to be slow since α must be small,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



Copyrighted material — do not circulate

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:

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

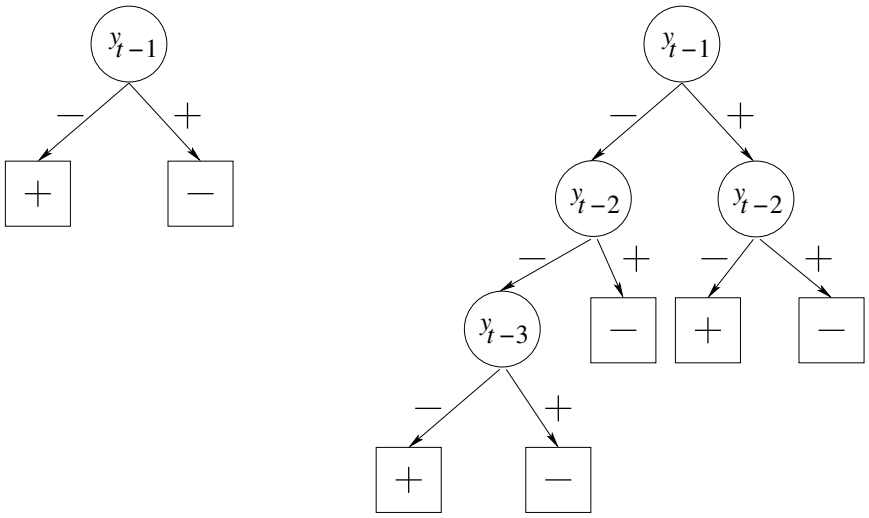


Figure 6.4: Two example context trees for predicting the human's next bit y_t based on those previously played $y_1, \ldots, y_{t-1}$. The trees are evaluated as in Section 1.3, which in this case means working our way backward through the history of played bits until a leaf node is reached providing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

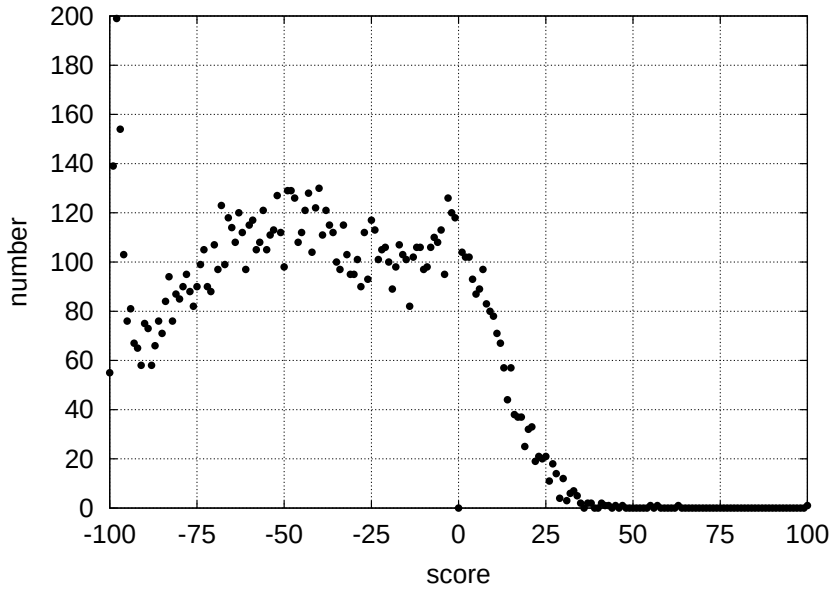


Figure 6.5: A histogram of the number of games played (out of 11,882) for each possible final score between -100 and 100, where the score is the number of rounds won by the human minus the number won by the computer, so games with negative scores were won by the computer. (No games had a score of zero 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.

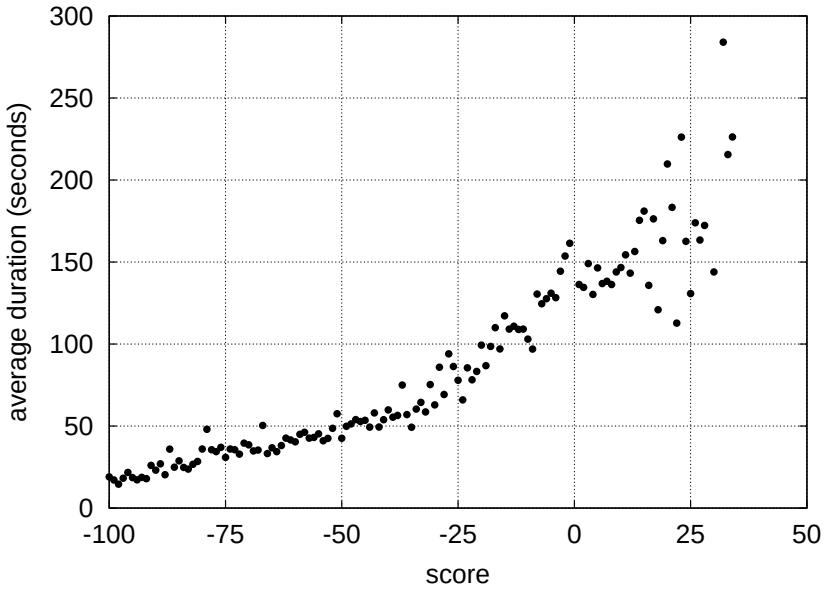


Figure 6.6: A plot of the average duration (in seconds) of the game compared to the final score of the game. For every possible score, a point is plotted whose x-value is the score, and whose y-value is the average of the durations of all games which ended with that particular final score. (No point 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.)