Chapter 1

Boosting and drifting games

In this chapter we describe a different class of boosting algorithm. The first algorithm in this class is the boost-by-majority algorithm published by Freund in 1995 [?]. These algorithms are significantly more complex than Adaboost, which is probably one of the reasons that they have received less attention by practitioners so far. However, they have some interesting and potentially useful properties, which might make them attractive to practitioners in the future.

To start, let us restrict ourselves to a boosting algorithm which produces an unweighted majority rule. In other words, the weights of the weak rules are all set to one. Furthermore, let us make the assumption that the number of boosting iterations (which is the number of weak rules combined in the final rule) is fixed ahead of time.

In this setup it is useful to think of boosting as a game between two players. One player represents the boosting algorithm and is called the booster or the weightor, the other player represents the weak learning algorithm and is called the learner or the chooser (the reason for the names weightor and chooser will become clear shortly.) We consider a fixed training set of size \( n \). The game proceeds in iterations, corresponding to the iterations of the boosting algorithm. In each iteration the booster assigns weights to the training examples and the learner responds by
generating a classification rule that is slightly better than random guessing with respect to the weights selected by the booster. After a pre-specified number of iterations the game stops and the booster outputs the (unweighted) majority vote over all of the weak rules that were generated. The booster wins if this final rule is correct on all \( n \) training examples.

To get some intuition into the game, let us consider some simple cases. First, consider a lazy booster that always uses the uniform distribution over the training set. The response of the learner to this strategy is simple, it chooses some weak rule, which is correct on \( 1/2 + \gamma \) of the training examples and always outputs this same rule. The majority rule is equal to this fixed rule, and the booster loses. Clearly, the booster has to change the distribution in order to prevent the learner from always outputting the same classification rule.

As a second example, consider a lazy learner which uses the following simple strategy. At each iteration it picks a classification rule which is correct with probability \( 1/2 + \gamma \) on each point in the instance space independently at random. In other words, conditional on the correct label, the prediction of the weak rules on the examples are independent of each other. While it is possible that this lazy choice yields a rule such that the total weight of the examples on which it is correct is smaller than \( 1/2 + \gamma \). However, it is easy to see that this has constant probability and therefore all the learner needs to do is try again a few times until it gets a good enough classifier.

If the learner uses this lazy strategy the booster is guaranteed to win, with high probability, if the number of iterations is sufficiently large. It is not hard to show that if the number of iterations is \( O(\log(n/\epsilon)\gamma^2) \) the probability that the majority vote rule is incorrect on any of the \( n \) training examples is at most \( \epsilon \). As it turns out, the lazy strategy is a min/max optimal strategy for the learner. This means that on the one hand, there is no strategy for the booster to win in a smaller number of iterations. On the other hand, there is a strategy for the booster that is guaranteed
to win in this number of iterations for any strategy that is used by the learner. This strategy was described in [] and the corresponding algorithm is called the boost-by-majority or BBM algorithm. In the next section we describe the strategy and the algorithm.

1.1 The boost by majority algorithm

In order to simplify the description of the game we rename the elements in a way which abstracts away the details but preserves the mathematical structure. We replace the set of $n$ training examples with a set of $m$ chips. We place the chips into bins which are indexed by integers. The chips that are in bin $i$ at iteration $t$ correspond to those examples for which the difference between the number of correct classifications (by the rules generated so far in the game) minus the number of incorrect classifications is equal to $i$. We denote by $n_i^t$ the number of chips that are in bin $i$ at iteration $t$. Bin 0 contains all of the chips at the start of the game, i.e. $n_0^1 = n$.

To hide the learning aspects of the game we rename the booster as the weightor and the learner as the chooser. We can now define the game formally. At each round $t = 1, \ldots, T$ the following sequence of steps takes place:

1. The weightor assigns a weighting function to the reachable bins $w_i^t \geq 0$ for all $i$.

2. The chooser selects, for each bin, the number of chips it will move upwards: $n_i^t \geq m_i^t \geq 0$. The selection has to be such that

$$\frac{\sum_i m_i^t w_i^t}{\sum_i n_i^t w_i^t} \geq \frac{1}{2} + \gamma$$

(1.1)

3. The locations of the chips are updated:

$$n_i^{t+1} = m_i^{t-1} + (n_i^{t+1} - m_i^{t+1})$$

(1.2)
See Figure 1.1 for an example of a single round of the game.

The weightor wins if, at the end of the game, there are no chips in bins below bin 1. In other words, if \( \sum_{i=-T}^{-1} n_i^{T+1} = 0 \). We assume that \( T \) is an odd number.\(^1\)

Before we give the complete strategy for the weightor, let us consider the best choice for the weightor on the last round, round \( T \). At this round the fate of most of the chips is already determined. Chips cannot “cross over” from negative bins to positive bins. The only chips whose fate is still to be determined are those in bin 0. It thus stands to reason that the weightor should put all of the weight on that bin, i.e. set \( w_0^T = 1 \) and \( w_i^T = 0 \) for all \( i \neq 0 \). This will force the chooser to move up \( 1/2 + \gamma \) of the chips in bin 0.

It is clear to see what the weightor should do on round \( T \), but what about earlier rounds? As it turns out, the min/max optimal weighting function has a very simple equation.

As we argued the weighting function at the last iteration is

\[
w_i^T = \begin{cases} 
1 & \text{if } i = 0 \\
0 & \text{otherwise}
\end{cases} \quad (1.3)
\]

The weight at earlier stages is defined using the following backwards recursion:

\[
\forall 1 \leq t < T \quad w_i^t = \left( \frac{1}{2} + \gamma \right) w_{i+1}^{t+1} + \left( \frac{1}{2} - \gamma \right) w_{i-1}^{t+1} \quad (1.4)
\]

---

\(^1\)If \( T \) is even then we have to make an arbitrary choice as to whether or not to include \( n_0^{T+1} \) in the sum and this choice complicates the notation without adding anything substantial to the analysis.
1.1. THE BOOST BY MAJORITY ALGORITHM

Note that because all chips are in bin \( i = 0 \) on iteration \( t = 1 \) the only bins that are occupied at iteration \( t \) are those whose index \( i \) has the opposite parity to that of \( t \). As \( T \) is odd we get that the definition given above works correctly in the sense that the same bins get non-zero weights.

The recursion can be solved and the weights can be described using the following explicit equation:

\[
w^t_i = \begin{cases} 
\left( \frac{T-t}{2} \right) \left( \frac{1}{2} + \gamma \right)^{\frac{T-t-1}{2}} \left( \frac{1}{2} - \gamma \right)^{\frac{T-t+1}{2}} & \text{if } |i| \leq T - t \text{ and } (i - t) : 2 = 1 \\
0 & \text{otherwise}
\end{cases}
\]

(1.5)

To analyze this weighting strategy we introduce the central concept of this chapter, the potential function \( \Phi^t_i \). The potential function maps the pair \( i, t \) which corresponds to bin \( i \) on round \( t \), to a real value. Intuitively, this value quantifies the potential loss associated with the chips that are in the bin \( i \) on round \( t \). We will soon justify this intuition.

The potential function is defined inductively as follows

\[
t = T + 1, i:2 = 1 \quad \Phi^{T+1}_i = \begin{cases} 
1 & \text{if } i < 0 \\
0 & \text{otherwise}
\end{cases}.
\]

(1.6)

\[
\forall 1 \leq t \leq T, (i - t):2 = 1 \quad \Phi^t_i = \left( \frac{1}{2} + \gamma \right) \Phi^{t+1}_{i+1} + \left( \frac{1}{2} - \gamma \right) \Phi^{t+1}_{i-1}
\]

(1.7)

There is a simple relationship between the potential function and the weights function and the potential function:

\[
w^t_i = \Phi^{t+1}_{i-1} - \Phi^{t+1}_{i+1}.
\]

(1.8)

We can combine Equations (1.8) and (1.5) and get the following explicit expression for the potential is²

\[
\forall 1 \leq t \leq T + 1, (i - t):2 = 1
\]

²Note that we don’t have to worry about summing only over bins with the correct parity as the weights associated with the other bins are zero.
Given: $\epsilon > 0, \gamma > 0,$ 
$(x_1, y_1), \ldots, (x_m, y_m)$ where $x_j \in \mathcal{X}, y_j \in \mathcal{Y} = \{-1, +1\}$ 
Initialize $c(j) := 0$ for all $1 \leq j \leq m.$ 
Set the number of iterations: $T \geq \frac{1}{2\gamma^2} \ln \frac{1}{\epsilon}$ 
For $t = 1, \ldots, T$: 

- Define the distribution $D_t$ over the $m$ training examples as follows: 
  $$w^t_{c(j)} = \begin{cases} 
  \left(\frac{T-t}{T-\epsilon(j)}\right)^{\frac{T-t-\epsilon(j)}{2}} \left(\frac{1}{2} + \gamma\right)^{\frac{T-t-\epsilon(j)}{2}} \left(\frac{1}{2} - \gamma\right)^{\frac{T-t+\epsilon(j)}{2}} & \text{if } |c(j)| \leq T - t \\
  0 & \text{otherwise}
  \end{cases}$$
  $$D_t(j) = \frac{w^t_{c(j)}}{Z}, \quad Z = \sum_{j=1}^{m} w^t_{c(j)}$$

- Get weak hypothesis $h_t: \mathcal{X} \rightarrow \{-1, +1\}$ with sufficiently small error: 
  $$\Pr_{i \sim D_t} [h_t(x_i) \neq y_i] \leq \frac{1}{2} - \gamma$$

- update $c(j), \forall 1 \leq j \leq m:$ 
  $$c(j) := c(j) + y_j h_t(x_j)$$

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

Figure 1.2: The boost-by-majority algorithm.
1.1. THE BOOST BY MAJORITY ALGORITHM

\[ \Phi_i^t = \sum_{j=i+1}^{\infty} w_j^{t-1} = \sum_{j=i+1}^{T-t+1} w_j^{t-1} = \sum_{j=0}^{T-t+1} w_{i+2j+1}^t \]

\[ = \sum_{j=0}^{T-t+1} \left( T - t + 1 \right) \left( \frac{1}{2} + \gamma \right) \frac{T-t-j}{2} - j \left( \frac{1}{2} - \gamma \right) \frac{T-t+j+1}{2} \] \hspace{1cm} (1.9)

The range of values that we sum over is the part of the list \( j = t - T - 1, t - T + 1, \ldots, T - t + 1 \) which also satisfies \( j \leq i - 1 \). I would like to give the erf based expression that approximates this sum.

We can now state the main theorem for the weighting scheme of the boost by majority algorithm:

**Theorem 1.1** If the weightor uses the weighting scheme defined in Equation (1.5) then the following sequence of inequalities holds:

\[ \sum_i n_i^1 \Phi_i^1 \geq \sum_i n_i^2 \Phi_i^2 \geq \cdots \geq \sum_i n_i^{T+1} \Phi_i^{T+1} \] \hspace{1cm} (1.10)

**Proof:** The proof follows from this sequence of inequalities:

\[ \sum_i n_i^t \Phi_i^t - \sum_i n_i^{t+1} \Phi_i^{t+1} = \sum_i \left( n_i^t \Phi_i^t - n_i^{t+1} \Phi_i^{t+1} \right) \] \hspace{1cm} (1.11)

\[ = \sum_i \left( n_i^t \left[ \Phi_i^{t+1} \left( \frac{1}{2} + \gamma \right) + \Phi_i^{t+1} \left( \frac{1}{2} - \gamma \right) \right] - \left[ n_i^t + m_i^{t-2} - m_i^t \right] \Phi_i^{t+1} \right) \] \hspace{1cm} (1.12)

\[ = - \left( \frac{1}{2} + \gamma \right) \sum_i n_i^t \left[ \Phi_i^{t+1} - \Phi_i^{t+1} \right] + \sum_i m_i^t \left[ \Phi_i^{t+1} - \Phi_i^{t+1} \right] \] \hspace{1cm} (1.13)

\[ \geq 0 \] \hspace{1cm} (1.14)

\[ \geq \sum_i \] \hspace{1cm} (1.15)

The equality in (1.11) is a reindexing of the second sum. Equality with expression (1.12) follows from substituting Equation (1.7) in the first term and Equation (1.2) in the second term.

Equality with expression (1.13) follows from combining the terms involving \( n_i^t \) and reindexing the term \( m_i^{t-2} \Phi_i^{t+1} \).
CHAPTER 1. BOOSTING AND DRIFTING GAMES

Inequality (1.13) follows from combining Equations (1.1) and (1.8) into the following inequality:

\[
\sum_i m_t^i (\Phi_{i-1}^{t+1} - \Phi_{i+1}^{t+1}) \geq \left( \frac{1}{2} + \gamma \right) \sum_i n_t^i (\Phi_{i-1}^{t+1} - \Phi_{i+1}^{t+1})
\]  

(1.16)

The following corollary follows directly from the theorem and shows that the proposed weighting scheme is indeed a boosting algorithm.

**Corollary 1.2** If we have a weak learner which can generate hypotheses whose weighted error is smaller than \( \frac{1}{2} - \gamma \) for some \( \gamma > 0 \) and if we set the number of boosting iterations to

\[
T \geq \frac{1}{2\gamma^2} \ln \frac{1}{2\epsilon}, \quad \epsilon > 0
\]

(1.17)

Then the weighting scheme given in Equation (1.5) defines a boosting algorithm that produces a final hypothesis whose training error is at most \( \epsilon \).

**Proof:** xxx ■

1.2 A probabilistic interpretation

One natural question that arises from Theorem 1.1 is whether this strategy for the weightor is optimal. The answer to this question is no, there are better strategies. However, the performance gap between the optimal weighting strategy and the weighting strategy defined by Equation (1.5) vanishes as the number of chips increases to infinity.

Instead of letting the number of chips increase to infinity, let us assume that the chips are a continuous mass. In other words, we assume that it is possible to divide the chips in any bin in any desired way. More precisely, for any \( 0 \leq r \leq 1 \), and any \( t \) and \( i \), it is possible to set \( m_t^i = rn_t^i \). Under this assumption, it is very easy to define the optimal strategy for the chooser. It is simply to set the fraction \( r \) to be \( \frac{1}{2} + \gamma \) for all \( i \) and \( t \).
This strategy has a natural probabilistic interpretation. The interpretation is that to say that the chooser chooses each chip with probability \( \frac{1}{2} + \gamma \) regardless of the past history of the chip. Suppose we associate with each chip a binary sequence that is 1 on iterations where the chip is selected and 0 otherwise. These sequences have the same distribution as sequences generated by independent draws from a binary distribution where the probability of 1 is \( \frac{1}{2} + \gamma \). Note that this is the strategy we described as the “lazy chooser” strategy in earlier in this chapter. We arrive at the surprising conclusion that the lazy strategy is the optimal strategy for the chooser.

This interpretation of the optimal strategy for the chooser yields a natural explanation of the potential loss \( \Phi^t_i \) defined in Equation (1.7) and its relationship to the optimal strategy for the weightor. The potential function is simply the conditional probability of losing a chip given that it is in bin \( i \) at time \( t \), assuming that the chooser will play optimally in all following iteration. In turn, this interpretation of the potential function gives an intuitive explanation of the relation between the weights and the potentials defined in Equation (1.8). The weight \( w^t_i \) is associated with \( n^t_i \) chips that are in bin \( i \) on iteration \( t \). Of these chips, \( m^t_i \) will move to bin \( i + 1 \) and \( n^t_i - m^t_i \) will move to the \( i - 1 \). If we assume that on all iterations after \( t \) the chooser will use the optimal strategy, then the probability of losing a chip from bin \( i - 1 \) at the end of the game is \( \Phi^{t+1}_{i-1} \) (\( \Phi^{t+1}_i \)). It is thus clear that the relative impact of a unit change in \( m^t_i \) is \( \Phi^{t+1}_{i-1} - \Phi^{t+1}_i \), which explains Equation (1.8).

### 1.3 Making boost by majority adaptive

The first step in transforming the BBM algorithm into a practical algorithm that can compete with Adaboost is to make it *adaptive*. In other words, we wish to remove the requirement that the algorithm has prior knowledge of an upper bound on the
error of the weak rules and combine them with equal weights. Instead, we want to have an algorithm that can use any rule whose error is smaller than $\frac{1}{2}$ and use a weighted majority where rules with small error are assigned more weight than rules with large error.

If we ignore computational complexity, this can be achieved quite easily. Recall that the requirement placed by the BBM algorithm on the weak learner is that the weighted error of the weak rules be at most $\frac{1}{2} - \gamma$. If we set $\gamma$ to be extremely small then it becomes essentially equivalent to the requirement that the error be strictly smaller than $1/2$, which is the same requirement that Adaboost places on the weak learner. Another effect of setting $\gamma$ small is that the changes in the weights assigned to examples on consecutive iterations are all extremely small. As a result, if the weighted error of some rule $h_1$ on iteration 1 is significantly smaller than $\frac{1}{2}$, say $\frac{1}{4}$, then the weighted error of $h_1$ will be continue to be smaller than $\frac{1}{2} - \gamma$ on iterations 2, 3, 4, ..., which means that $h_1$ can used as the weak rule on all of these iterations. After some number of iterations $k_1$, the weighted error of $h_1$ with respect to $w_i^{k_1}$ will be larger than $\frac{1}{2} - \gamma$, at which point $h_1$ can no longer be used. At that point a new weak rule, $h_2$ has to be found, if $h_2$ has error significantly smaller than $\frac{1}{2} - \gamma$ then it will “survive” from iteration $k_1$ till iteration $k_1 + k_2$ etc. The result of this is that in the final majority vote rule $h_1$ appears $k_1$ times, $h_2$ appears $k_2$ times etc. In other words, we can represent the final rule as a weighted majority over the rules $h_1, h_2, \ldots$ with corresponding weights $k_1, k_2, \ldots$.

Thus, in principle, we can use BBM as an adaptive boosting algorithm. However, the straightforward implementation of this idea is prohibitively expensive in computation time. What we would like to do instead is to somehow calculate the weight $k_i$ in a way that does not require time $k_i$ but rather something much smaller, such as $\log k_i$. We will now show how the problem of calculating the weight $k_i$ can be restated as a problem involving the solution of two nonlinear equations in two unknowns. This allows us to use standard numerical techniques, such as the
Newton-Raphson method to find the weights associated with the weak rules.

The first step in transforming the BBM algorithm into an algorithm that operates in continuous time is to rewrite Equation (1.7), the inductive definition of the potential function, in a slightly different form.

\[
\Phi_i^t - \Phi_i^{t-1} = \left( \Phi_i^t - \frac{1}{2} \Phi_i^t - 1 - \frac{1}{2} \Phi_i^t + 1 \right) - \gamma \left( \Phi_i^{t+1} - \Phi_i^{t-1} \right) \tag{1.18}
\]

Next, we rewrite this equation in the continuous domain, replacing the potential function defined over the integers \( \Phi_i^t \) with a potential function defined over the reals \( \Phi(s, t) \), where the real valued \( s \) takes the place of the integer \( i \). Also we generalize the time step from 1 to \( \Delta t \) and the location step from 1 to \( \Delta s \). After making these notational changes, Equation (1.18) is transformed to the following:

\[
\Phi(s, t) - \Phi(s, t - \Delta t) = -\frac{1}{2} \left( \Phi(s + \Delta s, t) - 2\Phi(s, t) + \Phi(s - \Delta s, t) \right) - \gamma \left( \Phi(s + \Delta s, t) - \Phi(s - \Delta s, t) \right) \tag{1.19}
\]

We are now ready to let the advantage \( \gamma \) decrease do zero while increasing the number of steps to infinity. From Corollary 1.2 we know that if we wish to keep the error of the final hypothesis fixed we need to increase the number of steps at the rate \( T = \beta / \gamma^2 \) for the \( \beta = -2/ \ln(2\epsilon) \). If we set the initial time to be 0 and the final time to be 1 we get that \( \Delta t = \gamma^2 / \beta \). If we set \( \Delta s = \sqrt{\Delta t} = \gamma / \sqrt{\beta} \) and divide both sides of Equation (1.19) by \( \Delta t \) we arrive at the following difference equation

\[
\frac{\Phi(s, t) - \Phi(s, t - \Delta t)}{\Delta t} = -\frac{1}{2} \frac{\Phi(s + \Delta s, t) - 2\Phi(s, t) + \Phi(s - \Delta s, t)}{\Delta s^2} - \frac{\sqrt{\beta} \Phi(s + \Delta s, t) - \Phi(s - \Delta s, t)}{\Delta s} \tag{1.20}
\]

Taking the limit of Equation 1.20 as \( \gamma \) decreases to zero we get the following partial differential equation that describes the time evolution of a brownian process, which is the continuous time limit of a random walk process.

\[
\frac{\partial \Phi(s, t)}{\partial t} = -\frac{1}{2} \frac{\partial^2 \Phi(s, t)}{\partial s^2} - 2\sqrt{\beta} \frac{\partial \Phi(s, t)}{\partial s} \tag{1.21}
\]
CHAPTER 1. BOOSTING AND DRIFTING GAMES

Given the boundary condition which defines the potential function at the end of the game

\[ \forall s, \quad \Phi(s, 1) = \begin{cases} 1 & \text{if } s < 0 \\ 0 & \text{otherwise} \end{cases}. \]  

(1.22)

We can express the solution of the PDE (1.21) in closed form for all \( t \leq 1 \):

\[ \Phi(s, t) = \frac{1}{2} \left( 1 - \operatorname{erf} \left( \frac{s + 2\sqrt{\beta}(1 - t)}{\sqrt{2(1 - t)}} \right) \right) \]  

(1.23)

and the weight function is

\[ w(s, t) = \frac{\partial}{\partial s} \Phi(s, t) = \exp \left( - \frac{(s + 2\sqrt{\beta}(1 - t))^2}{2(1 - t)} \right) \]  

(1.24)

Given this closed form definition of the potential function, we describe adaptive boost by majority algorithm in Figure 1.3.

1.4 Making boost by majority confidence rated

One of the deficiencies of the boost by majority algorithm is that it allows only binary weak classifiers, i.e. classifiers whose output is \(-1\) or \(+1\). There are important advantages to allowing the weak classifiers to be “confidence rated”, i.e. predict using values in the range \([-1, +1]\). A prediction of zero corresponds to no prediction while predictions close to \(-1\) or \(+1\) correspond to a confident prediction of \(-1\) or \(+1\) respectively.

We can generalize the majority vote game to this setting. The analysis of this case is technically quite involved. However, as it turns out, the analysis becomes considerably simpler when in the continuous time limit.

The upshot of this analysis is that the change we need to introduce to the adaptive boost by majority algorithm is quite simple. We need to replace the potential and weight functions given in Equations (1.23) and (1.24). The confidence rated potential function is

\[ \Phi(s, t) = \min \left( 1, \left( 1 - \operatorname{erf} \left( \frac{s + 2\sqrt{\beta}(1 - t)}{\sqrt{2(1 - t)}} \right) \right) \right) \]  

(1.25)
Given: $\epsilon > 0$

$(x_1, y_1), \ldots, (x_m, y_m)$ where $x_j \in \mathcal{X}, y_j \in \mathcal{Y} = \{-1,+1\}$

Set $\beta = -2/\ln(2\epsilon)$

Initialize $t_1 = 0$ and $s(j) := 0$ for all $1 \leq j \leq m$.

Repeat for $k = 1, 2, \ldots$

- Define the distribution $D_k$ over the $m$ training examples by normalizing $w(s,t)$ defined in Equation (1.24)

$$D_k(j) = \frac{w(s(j), t_k)}{Z}, \quad Z = \sum_{j=1}^{m} w(s(j), t_k)$$

- Get weak hypothesis $h_k : \mathcal{X} \rightarrow [-1,+1]$ which is slightly correlated with the label:

$$E_{j \sim D_k} [y_j h_k(x_j)] > 0$$

- **Find** $\Delta s_k > 0, \Delta t_k > 0$ such that the two following equations, defined using Equations (1.25) and (1.26), are solved simultaneously:

$$\sum_{j=1}^{m} y_j h_k(x_j) w(s(j) + y_j h_k(x_j) \Delta s_k + \Delta t_k) = 0$$

$$\sum_{j=1}^{m} \Phi(s(j), t_k) = \sum_{j=1}^{m} \Phi(s(j) + y_j h_k(x_j) \Delta s_k, t_k + \Delta t_k)$$

- **update:** $t_{k+1} := t_k + \Delta t_k, \forall 1 \leq j \leq m, \quad s(j) := s(j) + y_j h_k(x_j) \Delta s_k$

- **break if** $t_{k+1} \geq 1$.

Output the final hypothesis:

$$H(x) = \text{sign} \left( \sum_k \Delta s_k h_k(x) \right).$$

Figure 1.3: The adaptive boost-by-majority algorithm.
and the confidence rated weight function is

\[ w(s, t) = \begin{cases} 
\exp \left( -\frac{(s+2\sqrt{\beta (1-t)})^2}{2(1-t)} \right) & \text{if } s > -2\sqrt{\beta (1-t)} \\
0 & \text{if } s \leq -2\sqrt{\beta (1-t)}
\end{cases} \]  \hspace{1cm} (1.26)

### 1.5 Boosting the normalized margin

So far we have concentrated on boosting where the loss function we wish to optimize is the number of classification mistakes on the training set. This was is captured by setting the potential function at the end of the game, defined in Equation (1.22), to be the step function with the step placed at \( s = 0 \).

As we now know, there is more to boosting than minimizing the training error. Specifically, we know from chapter XX that YYY.

We would therefore like to replace the final potential function with a different potential function that will cause the boosting algorithm to increase the margins of example beyond the point at which they are classified correctly. We can try to do that by making the final potential function be a step function centered at \( s = 1 \) instead of \( s = 0 \). Doing this gives rise to some of the desired effect. However, we run into a new problem.

The problem is that the quantity that the theory suggests we should optimize is not the margin: \( y \sum_k \Delta s_k h_k(x) \) but rather the normalized margin: \( y (\sum_k \Delta s_k h_k(x)) / (\sum_k |\Delta s_k|) \). In order to address this issue, we need to go back the basic definitions of the voting games and come up with a different game that will capture the goal of maximizing the normalized margins.

The way we do it is by changing the way we map the majority vote game into the continuous domain. In BrownBoost examples from \((s, t)\) move to either \((s-\Delta s, t+\Delta t)\) or \((s+\Delta s, t+\Delta t)\). Instead, we now move examples from \((s, t)\) to \((s(1-\Delta t) - \Delta s, t + \Delta t)\) or \((s(1-\Delta t) + \Delta s, t + \Delta t)\). This changes Equation (1.20)
1.5. BOOSTING THE NORMALIZED MARGIN

\[ \Phi(s, t) - \Phi(s, t - \Delta t) \]
\[ = - \frac{\Delta t}{2} \Phi(s(1 - \Delta t) + \Delta s, t) - 2\Phi(s, t) + \Phi(s(1 - \Delta t) - \Delta s, t) \]
\[ - \frac{\Delta t}{\Delta s} \Phi(s(1 - \Delta t) + \Delta s, t) - \Phi(s(1 - \Delta t) - \Delta s, t) \]

Setting \( \Delta s^2 = \Delta t \) the last equation which can be rewritten as:

\[ \frac{\Phi(s, t) - \Phi(s, t - \Delta t)}{\Delta t} = - \frac{\Delta s^2}{2} \frac{\partial^2 \Phi(s, t)}{\partial s^2} \]
\[ + (s - 2\rho) \frac{\partial \Phi(s, t)}{\partial s} \]  \hspace{1cm} (1.27)

This differential equation corresponds to the backwards Kolmogorov Equation for the Ornstein-Uhlenbeck diffusion process.

A family of solutions of the PDE (1.28) is:

\[ \Phi(s, t) = \frac{1}{2} \left( 1 - \text{erf} \left( \frac{s - \mu(t)}{\sigma(t)} \right) \right) \] \hspace{1cm} (1.29)

Where

\[ \sigma^2(t) = c_1 e^{-2t} - 1 \]
\[ \mu(t) = c_2 e^{-t} + 2\rho \]

and \( c_1, c_2 \) are real valued constants. We set these constants so that the final potential function, which is the target loss function, has the form that we want. Specifically, we set \( \mu(1) = \theta \) and \( \sigma(1) = \sigma_f \). Solving for these constraints we get

\[ \sigma^2(t) = (\sigma_f^2 + 1)e^{2(1-t)} - 1 \] \hspace{1cm} (1.30)
\[ \mu(t) = (\theta - 2\rho)e^{1-t} + 2\rho \] (1.31)

We set the value of \( \rho \) according to our target loss \( \epsilon \). Specifically we find \( \rho \) that satisfies
\[ \epsilon = \Phi(0, 0) = \frac{1}{2} \left( 1 - \text{erf} \left( \frac{2(e - 1)\rho - e\theta}{\sqrt{e^2(\sigma_f^2 + 1) - 1}} \right) \right) \]

We can apply the same arguments used in Section 1.4 to show that if we allow confidence rated weak classifiers we get
\[ \Phi(s, t) = \min \left( 1, \left( 1 - \text{erf} \left( \frac{s - \mu(t)}{\sigma(t)} \right) \right) \right) \] (1.32)
and the confidence rated weight function is
\[ w(s, t) = \begin{cases} \exp \left( -\frac{(s - \mu(t))^2}{2\sigma(t)^2} \right) & \text{if } s > \mu(t) \\ 0 & \text{if } s \leq \mu(t) \end{cases} \] (1.33)

The time evolution of \( \mu \) and \( \sigma \) remain the same, as defined in Equations (1.30,1.31). The only change is in the equation that we solve to set \( \rho \), which is now:
\[ \epsilon = \Phi(0, 0) = 1 - \text{erf} \left( \frac{2(e - 1)\rho - e\theta}{\sqrt{e^2(\sigma_f^2 + 1) - 1}} \right) \] (1.34)

See Figure 1.4 for an example of the evolution of the potential function. See Figure 1.5 for the pseudo-code of NormalBoost.

### 1.5.1 Tuning the parameters

In order to apply NormalBoost to a particular dataset we need to specify how to choose the parameters \( \theta, \sigma_f \) and \( \epsilon \). The role of \( \sigma_f \) and \( \theta \) is to control the margin distribution of the examples in order to avoid over-fitting. As overfitting decreases with the size of the training set \( n \), we suggest setting \( \sigma = \sigma_0/\sqrt{n} \) and \( \theta = \theta_0/\sqrt{n} \). The choice \( \sigma_0 \) and \( \theta_0 \) depends on the richness (or dimension) of the base classifiers, a richer set of base classifiers is more prone to over-fitting and thus requires setting \( \theta_0 \) and \( \sigma_0 \) higher. In any case, the choice should be made prior to training because
1.5. **BOOSTING THE NORMALIZED MARGIN**

choosing $\sigma$ and $\theta$ according to the training set increases the danger of over-fitting. **To be completed with rigorous bounds on the generalization error given the margin distribution.**

The role of $\epsilon$ is to control the target error rate for the learning process. Setting $\epsilon$ to a large value makes the learning problem easier. Setting $\epsilon$ small makes the problem harder. If $\epsilon$ is set too low, the learning process will not terminate because the condition $t_k = 1$ is never satisfied. Setting $\epsilon$ according to the training set should not lead to over-fitting as over-fitting is controlled by $\theta$ and $\sigma_f$. We suggest selecting $\epsilon$ by performing a line search on the segment $[0, 1]$ to find the smallest value of $\epsilon$ for which the algorithm terminates. As a start value one can use the test error achieved by using Adaboost or LogitBoost on the same training set.

### 1.5.2 Justification of update rule

This section is very technical, probably not appropriate for the book.

The NormalBoost algorithm is defined in the same way as BrownBoost, using Figure 1.3, with two important changes:

1. Equations (1.32) and (1.33) replacing Equations (1.23) and (1.24).

2. Instead of updating position using $s(j) := s(j) + y_j h_k(x_j) \Delta s_k$ we use the formula

$$s(j) := s(j) e^{-\alpha \Delta t_k} + y_j h_k(x_j) \Delta s_k$$

(1.35)

The justification is as follows. Suppose that we divide the time step $\Delta t$ into $n$ equal parts of length $\Delta t/n$. At each of these steps $i = 1, \ldots, n$ we solve for $\Delta s_i$ and update $s$, assuming that $\Delta t$ is sufficiently small, we know that $|\Delta s_i| \leq \sqrt{\Delta t/n}$. The step from $s_0$ to $s_1$.

$$s_1 = s_0 (1 - \alpha \Delta t/n) + \Delta s_0$$
if we expand this formula to two steps we get

\[ s_2 = (s_0(1 - \alpha \Delta t / n) + \Delta s_0)(1 - \alpha \Delta t / n) + \Delta s_1 = s_0(1 - \alpha \Delta t / n)^2 + \Delta s_0 - \alpha \Delta s_0 \Delta t / n + \Delta s_1 \]

recalling that \( |\Delta s_0| \leq \sqrt{\Delta t / n} \), this means that the term \( \alpha \Delta s_0 \Delta t / n \) is smaller than \( \alpha (\Delta t)^{3/2} \) thus as \( n \to \infty \) this term becomes negligible compared to \( \Delta s_0 \) and \( \Delta s_1 \). In other words

\[ s_2 = s_0(1 - \alpha \Delta t / n)^2 + \Delta s_0 + \Delta s_1 + O(n^{-3/2}) \]

We can now recurse over all \( n \) steps and get

\[ s_n = s_0(1 - \alpha \Delta t / n)^n + \sum_{i=1}^{n} \Delta s_i + O(n^{-1/2}) \]

Taking the limit \( n \to \infty \) we get Equation (1.35).

### 1.6 Asymmetric boosting

In many binary classification problems there is an inherent asymmetry between the two classes. There are two types of asymmetry:

- **Cost asymmetry**: A binary classification rule can make two types of mistakes: false positives, which are negative examples that are misclassified as positive, and false negatives, which are positive examples that are misclassified as negative. In some applications the different mistakes carry very different costs. For example, consider the problem of classifying people as healthy or sick. The consequences of mistaking a sick person to be healthy are much more dire than those of mistaking a healthy person to be sick. In the second case a healthy person might receive unnecessary treatment, while in the first case the sick person might die. We use \( C \) to denote the ratio between the cost of a false negative and the cost of a false positive. In the
1.6. ASSYMETRIC BOOSTING

Figure 1.4: The potential of the confidence-rated NormalBoost as a function of margin $y_s$ and time $t$. The different curves correspond to $t = 0, 0.1, \ldots, 0.9, 1$, where $t = 0$ is the light blue curve and $t = 1$ is the black curve. The parameters used are $\epsilon = 0.1, \theta = 1, \sigma_f = 1$. 
CHAPTER 1. BOOSTING AND DRIFTING GAMES

Given: $\epsilon > 0, \theta > 0, \sigma_f > 0$

$(x_1, y_1), \ldots, (x_m, y_m)$ where $x_j \in \mathcal{X}, y_j \in \mathcal{Y} = \{-1, +1\}$

Set $\rho$ to satisfy Equation (1.34).

Initialize $t_1 = 0, H_0 \equiv 0$ and $s(j) := 0$ for all $1 \leq j \leq m$.

Repeat for $k = 1, 2, \ldots$

- Define the distribution $D_k$ over the $m$ training examples by normalizing $w(s, t)$ defined in Equations (1.33,1.30,1.31)

  $$D_k(j) = \frac{w(s(j), t_k)}{Z}, \quad Z = \sum_{j=1}^{m} w(s(j), t_k)$$

- Get weak hypothesis $h_k : \mathcal{X} \rightarrow [-1, +1]$ which is slightly correlated with the label:

  $$E_{j \sim D_k}[y_j h_k(x_j)] > 0$$

- Find $\Delta s_k > 0, 1 - t_k \geq \Delta t_k > 0$ that simultaneously satisfy the following two equations:

  $$\sum_{j=1}^{m} y_j h_k(x_j) w(s'(j), t_k + \Delta t_k) = 0 \quad \text{or} \quad \Delta t_k = 1 - t_k$$

  $$\sum_{j=1}^{m} \Phi(s(j), t_k) = \sum_{j=1}^{m} \Phi(s'(j), t_k + \Delta t_k)$$

  where

  $$s'(j) = s(j) e^{-\Delta t_k} + y_j h_k(x_j) \Delta s_k$$

  and $\Phi(\cdot, \cdot), w(\cdot, \cdot)$ are defined by Equations (1.32,1.33,1.30,1.31).

- update: $t_{k+1} := t_k + \Delta t_k, \forall 1 \leq j \leq m, \quad s(j) := s'(j)$

  $$H_k = H_{k-1} e^{-\Delta t_k} + \Delta s_k h_k$$

- break if $t_{k+1} = 1$.

Output the final hypothesis $H_k$.

Figure 1.5: The Normal-Boost algorithm.
1.6. ASSYMETRIC BOOSTING

previous example $C$ is a very large number because the price of a false negative is much higher than that of a false positive. The goal of the learning algorithm in this case it to find a classification rule with a small expected cost

$$E_{(x,y) \sim D} [\text{cost}(h, x, y)] = C \Pr_{(x,y) \sim D} [h(x) = -1 \land y = +1] + \Pr_{(x,y) \sim D} [h(x) = +1 \land y = -1]$$

- frequency asymmetry: In some classification problems one of the classes is much more prevalent than the other. For example, in Viola and Jones work on detecting faces in images, the frequency of faces is four to five orders of magnitudes lower than the frequency of non faces. In this case the danger of overfitting the positive examples is much greater than that of overfitting the negative examples.\(^4\) We denote the number of positive training examples by $n_+$ and the number of negative examples by $n_-$. We assume that the positive label corresponds to the rare class, i.e. $n_+ < n_-$. Often problems exhibit both types of asymmetry, for example, sick people are (hopefully) less frequent than healthy people, at the same time, misdiagnosing a sick person as healthy is much more costly than misdiagnosing a healthy person as sick.

Adjusting NormalHedge for assymetric problems is straight-forward. We define different potential functions for the two classes. Specifically, we define

$$\Phi(s, t, y) = \begin{cases} 
C \min \left( 1, \left( 1 - \text{erf}\left( \frac{s - \mu_+(t)}{\sigma_+(t)} \right) \right) \right) & \text{if } y = +1 \\
\min \left( 1, \left( 1 - \text{erf}\left( \frac{s - \mu_-(t)}{\sigma_-(t)} \right) \right) \right) & \text{if } y = -1
\end{cases}$$

Where $\mu_+(t)$ and $\sigma_+(t)$ ($\mu_-(t)$ and $\sigma_-(t)$) are set according to Equations (1.30,1.31) with parameters $\theta_+, \sigma_f+$ and $\rho_+$ ($\theta_-, \sigma_f-$ and $\rho_-$ respectively.) We suggest the fol-

\(^4\)Viola and Jones approach to this problem was to give higher weight to the positive examples, in effect, using an assymetric cost. It is not clear whether this approach was effective. I would like to show experimentally that the approach presented here is more effective than re-weighting
lowing heuristic for setting the over-fitting control parameters:

\[ \theta_+ = \frac{\theta_0}{\sqrt{n_+}}, \quad \sigma_f = \frac{\sigma_0}{\sqrt{n_+}} \]

\[ \theta_- = \frac{\theta_0}{\sqrt{n_-}}, \quad \sigma_f = \frac{\sigma_0}{\sqrt{n_-}} \]

The parameters \( \rho_+ \) and \( \rho_- \) are set by finding the solutions to the following two equations:

\[ \Phi(0, 0, +1) = \epsilon, \quad \Phi(0, 0, -1) = C\epsilon \]

where \( \epsilon \) is the target expected cost. When \( \epsilon \) is not known in advance, we suggest using a line search on the training set to find the smallest achievable value for \( \epsilon \).

Figures 1.6 and 1.7 illustrate the evolution of asymmetric potential functions for the two types of asymmetry described above. As the potential functions for positive and negative examples are different, we use the score itself, rather than the margin (which is score times correct label) as the horizontal axis in these figures.
Figure 1.6: asymmetric costs The asymmetric potentials of the confidence-rated NormalBoost as a function of score $s$ and time $t$. The different curves correspond to $t = 0, 0.1, \ldots, 0.9, 1$, where $t = 0$ is the light blue curve and $t = 1$ is the black curve. The parameters used are $C = 10, \epsilon = 0.01, n_+ = n_- = 10000, \sigma_0 = \theta_0 = 10$. 
Figure 1.7: **asymmetric costs** The asymmetric potentials of the confidence-rated NormalBoost as a function of score $s$ and time $t$. The different curves correspond to $t = 0, 0.1, \ldots, 0.9, 1$, where $t = 0$ is the light blue curve and $t = 1$ is the black curve. The parameters used are $C = 1, \epsilon = 0.01, n_+ = 100, n_- = 10000, \sigma_0 = \theta_0 = 10$. 