Boost Your Knowledge of AdaBoost

Mark Stamp*
Department of Computer Science
San Jose State University

December 15, 2019

1 Introduction

Boosting is a process whereby we can combine multiple (weak) classifiers into one (much stronger) classifier [1]. Of course, many machine learning techniques can be applied in a somewhat similar fashion. For example, a meta-scoring approach can be implemented by using an SVM to construct a classifier based on a variety of different scores [3]. But the beauty of boosting is that the individual classifiers can be extremely weak—in fact, anything that is better than a coin flip can be used. And provided that we have a sufficient number of usable classifiers, boosting enables us to construct an arbitrarily strong classifier. Classifiers don't get much better than "arbitrarily strong."

The best-known boosting algorithm is *AdaBoost*, which is a clever shorthand for "adaptive boosting." At each iteration of AdaBoost, we use a greedy strategy, in the sense that we select the individual classifier—and its associated weight—that improves our overall classifier the most. It's an adaptive approach, since we build each intermediate classifier based on the classifier that was determined at the previous step of the algorithm.

It is worth noting that AdaBoost is not a hill climb. Yes, we select the best available classifier at each step, but there is no guarantee that this selection will improve our overall classifier. We'll see that it is generally advantageous not to stop just because the classifier gets worse at some particular iteration.

*Email: mark.stamp@sjsu.edu. This tutorial was originally published online in 2017, with a few minor corrections having been made since. This is an expanded (and slightly corrected) version of Section 7.4.2 of my book, *Introduction to Machine Learning with Applications in Information Security* [3], a section which is itself based on Rojas' excellent paper, "AdaBoost and the Super Bowl of Classifiers: A Tutorial Introduction to Adaptive Boosting" [1].

2 Football Analogy

To illustrate the AdaBoost process, let's consider the problem of building the best football team possible from a collection of average players.[1] One reasonable way to organize a team would be to select the best player at each position from your collection of players. However, suppose that your best quarterback is terrible, while your best receiver is also one of your best players at several other positions. If you put your best receiver at receiver, he won't catch any passes, because your quarterback is terrible. So, it would seem to make more sense to put your best receiver at some other position where he is also very good. In other words, the best overall team might not have each player playing at his best position.

The following adaptive strategy could be used to try to assemble the best possible team from a collection of average players.

1. Select the best player and determine his role on the team.
2. Determine the biggest weakness remaining on the team.
3. From the remaining players, choose the one who can best improve on the weakness identified in step 2.
4. Decide exactly what role the newly selected player will fill.
5. If the team is not yet complete, goto 2.

Note that this strategy does not assure us of obtaining the best possible team—that might require an exhaustive search over all possible teams. But, this adaptive approach is likely to produce a better team than the naïve approach of simply selecting the best player at each position. And the weaker the individual players, the better this algorithm is likely to perform, as compared to the naïve strategy.

3 AdaBoost

AdaBoost is somewhat analogous to building your football team using the adaptive strategy outlined in the previous section. At each iteration, we'll identify the biggest weakness in the classifier that we have constructed so far. Then we'll determine which of the remaining available classifiers will help the most

[1] If you don't know much about American football, don't worry. All you need to know is that it's like rugby, except that a ball is used instead of a watermelon. Also, in football the quarterback throws the ball to a receiver on his same team, who is supposed to catch it.

with respect to this weakness. Finally, we determine the best way to merge this newly-selected classifier into our overall classifier.

Using this adaptive approach, we can combine a large number of weak classifiers to obtain a result that is much stronger than any of the individual classifiers. The AdaBoost algorithm that we describe below is efficient and relatively easy to implement.

But before discussing AdaBoost in more detail, we note that in practice, boosting may not achieve the seemingly too-good-to-be-true results that we promised above. Unfortunately, boosting algorithms tend to be extremely sensitive to noise. The reason for this sensitivity should become clear as we discuss AdaBoost.

As mentioned above, AdaBoost is the most popular boosting technique and it uses an iterative, greedy approach. In the remainder of this section, we describe the AdaBoost algorithm in detail.

Suppose that we have a labeled training set of size n, denoted (X_i, z_i), for $i = 1, 2, \ldots, n$, where X_i is the i^{th} data point, with corresponding label (i.e., classification) z_i. Since we are dealing with a binary classification problem, we'll assume that each $z_i \in \{-1, +1\}$. In addition, we assume that we have L (weak) classifiers, denoted c_ℓ, for $\ell = 1, 2, \ldots, L$, where each c_ℓ assigns a label (either $+1$ or -1) to each X_i. An example of such training data is illustrated in Table 1. Note that an entry of $+1$ in row i, column ℓ in Table 1 indicates that classifier c_ℓ classifies data point X_i as type $+1$, while a -1 indicates that c_ℓ classifies the data point as type -1. By comparing $c_\ell(X_i)$ to z_i, for $i = 1, 2, \ldots, n$, we can easily determine the number of "hits" (i.e., correct classifications) and the number of "misses" (i.e., incorrect classifications) for classifier c_ℓ. Note that for an c_ℓ, the sum of the hits and misses is n.

Table 1: Classifiers

Data	Label	Classifiers			
		c_1	c_2	$\cdots$	c_L
X_1	z_1	-1	$+1$	$\cdots$	$+1$
X_2	z_2	$+1$	-1	$\cdots$	-1
X_3	z_3	-1	-1	$\cdots$	$+1$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
X_n	z_n	-1	$+1$	$\cdots$	-1

Our goal is to construct a classifier $C(X)$ as a weighted combination of the c_ℓ. Of course, we want $C(X)$ to be as strong as possible, that is, we want $C(X)$ to correctly classify as many of the training vectors X_i as possible.

The classifiers c_ℓ can be weak, that is, the number of hits might be only marginally greater than the number of misses. Also, note that if the number of misses exceeds the number of hits for a given c_ℓ, we simply reverse the sense of the classifier (i.e., replace each $+1$ with -1, and vice versa, in column ℓ of Table 1) to obtain a classifier with more hits than misses. However, a classifier cannot be perfectly random, that is, we cannot use a classifier for which the number of hits and misses are exactly equal. In other words, the only classifiers that are not usable for boosting are those that are indistinguishable from flipping a fair coin.

AdaBoost is an iterative algorithm whereby we generate a series of classifiers, $C_1, C_2, \ldots, C_M$, with the desired final classifier being $C = C_M$. Furthermore, each classifier is of the form

$$C_m(X_i) = \alpha_1 k_1(X_i) + \alpha_2 k_2(X_i) + \cdots + \alpha_m k_m(X_i)$$

where each k_j is a distinct classifier selected from the set $\{c_1, c_2, \ldots, c_L\}$, and each α_j is a non-negative weight. Further-furthermore, we'll select the classifiers and weights so that

$$C_m(X_i) = C_{m-1}(X_i) + \alpha_m k_m(X_i).$$

That is, at the m^{th} iteration, we include another classifier k_m from the set of unused c_ℓ and determine a new weight α_m. In terms of the football analogy discussed above, we select one of the remaining players (corresponding to k_m), and we determine his role (corresponding to α_m), without changing any of the previously selected players or their roles. This is a greedy approach, since we choose k_m and α_m to maximize the improvement at each step.[2]

In AdaBoost, we use an exponential *loss function* to determine the best classifier at each step. A loss function is like a score, except that the smaller the loss, the better, whereas larger scores are better. Specifically, the loss (or error) function at step m is defined as

$$E_m = \sum_{i=1}^{n} e^{-z_i \left(C_{m-1}(X_i) + \alpha_m k_m(X_i) \right)}$$

where k_m is to be chosen from among the unused classifiers c_ℓ and $\alpha_m > 0$ is to be determined. Equivalently, we can write the loss function as

$$E_m = \sum_{i=1}^{n} w_i \, e^{-z_i \alpha_m k_m(X_i)} \tag{1}$$

[2]The maximum "improvement" at a given iteration might actually be an "unimprovement," that is, the number of correct classifications can decrease. It follows that AdaBoost is not a hill climb algorithm.

where

$$w_i = e^{-z_i C_{m-1}(X_i)}. \tag{2}$$

Using equation (2), equation (1) can be rewritten as

$$E_m = \sum_{z_i = k_m(X_i)} w_i \, e^{-\alpha_m} + \sum_{z_i \neq k_m(X_i)} w_i \, e^{\alpha_m}$$

which we write as

$$E_m = W_1 e^{-\alpha_m} + W_2 \, e^{\alpha_m} \tag{3}$$

where

$$W_1 = \sum_{z_i = k_m(X_i)} w_i \text{ and } W_2 = \sum_{z_i \neq k_m(X_i)} w_i. \tag{4}$$

Then

$$e^{\alpha_m} E_m = W_1 + W_2 \, e^{2\alpha_m}$$

and it follows that

$$e^{\alpha_m} E_m = (W_1 + W_2) + W_2\big(e^{2\alpha_m} - 1\big).$$

Letting $W = W_1 + W_2$, we have

$$e^{\alpha_m} E_m = W + W_2\big(e^{2\alpha_m} - 1\big).$$

From the definitions of W_1 and W_2, we see that any increase in W_1 is offset by precisely the same decrease in W_2, and vice versa. Consequently, W is fixed for each iteration. Therefore, regardless of the value of α_m, we'll want to choose k_m so that W_2 is minimized.

Once we have selected a k_m that minimizes W_2, we must determine the corresponding coefficient α_m. Note that once k_m is specified, both W_1 and W_2 are known. From equation (3), we see that

$$\frac{dE_m}{d\alpha_m} = -W_1 e^{-\alpha_m} + W_2 \, e^{\alpha_m}.$$

Setting this derivative equal to zero and solving for α_m, we find that the desired minimum occurs when

$$\alpha_m = \frac{1}{2} \, \ln\left(\frac{1 - r_m}{r_m}\right) \tag{5}$$

where

$$r_m = \frac{W_2}{W_1 + W_2} = \frac{W_2}{W}.$$

AdaBoost is specified in detail here as Algorithm 3.1. In this algorithm, we are assuming that for each classifier c_ℓ, the number of hits is larger than the number of misses. If this is not the case, we would preprocess the classifiers to remove any that have an equal number of hits and misses, and we would reverse the sense of any classifier that has more misses than hits.

Note that for the initial $m = 1$ iteration, we define $C_{m-1} = C_0 = 0$, which implies that the weights in equation (2) are given by $w_i = 1$, for $i = 1, 2, \ldots, n$. Consequently, when determining the initial classifier C_1, we simply choose a classifier c_ℓ that minimizes the number of misses. At every iteration—including the initial step—we compute α_m using equation (5).

Algorithm 3.1 AdaBoost

Given:
  Labeled data (X_i, z_i) for $i = 1, 2, \ldots, n$
  Classifiers $c_\ell(X_i)$ for $\ell = 1, 2, \ldots, L$ and $i = 1, 2, \ldots, n$
Initialize:
  $C_0(X_i) = 0$ for $i = 1, 2, \ldots, n$
  $u_j = 0$ for $j = 1, 2, \ldots, L$
for $m = 1, 2, \ldots, L$ **do**
  $w_i = e^{-z_i C_{m-1}(X_i)}$ for $i = 1, 2, \ldots, n$
  $W = \sum w_i$
  $W_2 = \infty$
  for $j = 1, 2, \ldots, L$ **do**
    if $u_j = 0$ **then** // classifier c_j has not yet been used
      $Y = \sum_{z_i \neq c_j(X_i)} w_i$
      if $Y < W_2$ **then**
        $W_2 = Y$
        $t = j$
      end if
    end if
  end for
  $k_m = c_t$
  $u_t = 1$ // marks classifier c_t as used
  $r_m = W_2/W$
  $\alpha_m = \frac{1}{2} \ln\left(\frac{1 - r_m}{r_m}\right)$
  $C_m(X_i) = C_{m-1}(X_i) + \alpha_m k_m(X_i)$
end for

Again, in the football analogy, choosing k_m from among the unused c_ℓ corresponds to selecting the player—from among those not yet selected—who can help the team the most. And determining α_m corresponds (roughly) to putting the selected player at the position that does the most good for the team.

As previously mentioned, errors in the training data can be an Achilles heel for AdaBoost. This is fairly clear from the algorithm, as an error in one step will tend to snowball in subsequent iterations. It is also interesting to note that due to the exponential weighting function, outliers can have excessive influence on the final classifier.

4 Examples

In this section, we consider examples that illustrate interesting and important aspects of AdaBoost. We start with a small example that serves to illustrate the workings of Algorithm 3.1. Then we consider a much bigger example that illustrates the ability of AdaBoost to generate a strong classifier from a collection of extremely weak classifiers. In this latter case, we also experiment with the number of available weak classifiers. All of the data used in these examples is available at [2].

4.1 A Small Example

Here, we consider a small example in some detail. Specifically, for this example, we have a set of $n = 25$ labeled data points and $L = 30$ classifiers. The data we used is given in Table 2.

For the first iteration of AdaBoost, as given in Algorithm 3.1, we select a classifier c_ℓ with the minimum number of misses (or, equivalently, the maximum number of hits). For the data in Table 2, we could select any of the classifiers with 17 hits. By following Algorithm 3.1, we choose c_8 as our initial classifier. We then have $W_2 = 8$ and $W = 25$, which gives us $r_1 = 8/25 = 0.32$ and

$$\alpha_1 = \frac{1}{2} \ln\left(\frac{1-0.32}{0.32}\right) = 0.376886.$$

Thus, our first constructed classifier is

$$C_1(X_i) = \alpha_1 c_8(X_i) = 0.376886\, c_8(X_i).$$

At the next iteration, $m = 2$ and we compute the weights as in line 4 of Algorithm 3.1. Due to the particularly simple form of C_1, we have

$$w_i = \begin{cases} e^{0.376886} & \text{if } C_1(X_i) \text{ is a hit} \\ e^{-0.376886} & \text{if } C_1(X_i) \text{ is a miss.} \end{cases} \tag{6}$$

Table 2: Data for the example in Section 4.1

Data	z_i	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}	c_{16}	c_{17}	c_{18}	c_{19}	c_{20}	c_{21}	c_{22}	c_{23}	c_{24}	c_{25}	c_{26}	c_{27}	c_{28}	c_{29}	c_{30}
X_1	−1	+1	−1	−1	+1	+1	+1	−1	−1	−1	+1	+1	−1	−1	−1	−1	+1	+1	+1	+1	+1	−1	−1	−1	−1	−1	+1	+1	+1	−1	+1
X_2	+1	+1	−1	+1	−1	+1	−1	+1	+1	+1	+1	+1	+1	+1	+1	−1	+1	−1	+1	+1	+1	+1	+1	+1	−1	−1	+1	+1	+1	+1	+1
X_3	−1	+1	+1	+1	+1	−1	+1	−1	+1	+1	+1	+1	−1	−1	+1	−1	+1	+1	−1	−1	−1	−1	−1	−1	−1	+1	−1	−1	−1	−1	−1
X_4	+1	+1	+1	−1	+1	+1	+1	+1	+1	+1	+1	−1	−1	−1	−1	+1	+1	+1	+1	+1	+1	−1	+1	+1	+1	−1	−1	+1	+1	+1	+1
X_5	−1	−1	+1	+1	+1	−1	+1	−1	+1	−1	−1	−1	−1	−1	−1	−1	−1	+1	−1	+1	+1	−1	−1	+1	−1	−1	+1	−1	−1	−1	+1
X_6	+1	+1	−1	−1	+1	−1	−1	−1	−1	−1	+1	+1	+1	+1	−1	+1	−1	+1	+1	−1	+1	−1	−1	−1	+1	+1	−1	+1	−1	−1	+1
X_7	−1	+1	−1	+1	+1	−1	−1	−1	+1	+1	−1	+1	−1	−1	+1	−1	−1	−1	−1	+1	−1	+1	+1	−1	+1	−1	−1	−1	−1	−1	+1
X_8	+1	−1	+1	+1	−1	+1	+1	+1	−1	+1	−1	+1	−1	−1	−1	+1	−1	−1	−1	+1	−1	−1	−1	+1	+1	+1	+1	+1	−1	−1	+1
X_9	−1	−1	+1	−1	+1	−1	+1	+1	−1	+1	+1	−1	+1	−1	−1	+1	−1	−1	+1	+1	−1	−1	−1	+1	−1	−1	−1	−1	−1	+1	+1
X_{10}	+1	+1	−1	+1	+1	+1	−1	+1	−1	−1	−1	−1	−1	+1	−1	+1	−1	−1	+1	−1	−1	−1	+1	+1	+1	−1	+1	+1	+1	−1	−1
X_{11}	−1	+1	−1	−1	−1	+1	−1	−1	+1	−1	+1	+1	−1	−1	−1	+1	−1	−1	−1	+1	+1	−1	+1	+1	+1	−1	+1	+1	+1	−1	−1
X_{12}	+1	+1	+1	−1	−1	−1	+1	−1	+1	+1	−1	−1	−1	−1	+1	−1	−1	−1	−1	−1	+1	+1	−1	−1	+1	+1	+1	+1	−1	+1	+1
X_{13}	−1	−1	+1	+1	−1	−1	−1	+1	+1	+1	−1	+1	+1	+1	+1	+1	−1	−1	+1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	+1	+1
X_{14}	+1	+1	+1	−1	+1	−1	+1	+1	+1	+1	−1	−1	+1	−1	+1	−1	−1	+1	−1	+1	+1	−1	−1	+1	−1	−1	+1	+1	+1	+1	+1
X_{15}	−1	−1	+1	−1	−1	+1	+1	+1	−1	−1	−1	+1	−1	−1	−1	+1	+1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	+1	−1	−1	−1
X_{16}	+1	+1	−1	+1	−1	+1	+1	−1	+1	+1	−1	+1	+1	+1	+1	−1	+1	+1	+1	−1	−1	+1	−1	−1	−1	+1	−1	+1	−1	+1	+1
X_{17}	−1	+1	−1	−1	−1	−1	−1	+1	−1	+1	−1	+1	+1	−1	−1	−1	+1	−1	+1	−1	+1	−1	−1	+1	−1	+1	−1	−1	−1	−1	−1
X_{18}	+1	−1	−1	+1	−1	−1	+1	−1	+1	+1	−1	−1	−1	+1	+1	+1	−1	+1	+1	−1	−1	+1	−1	−1	−1	−1	−1	+1	+1	+1	+1
X_{19}	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	+1	−1	−1	+1	+1	−1	+1	−1	+1	−1	−1	−1	−1	+1	+1	+1	+1	+1	−1	+1	+1
X_{20}	+1	−1	+1	−1	+1	−1	+1	+1	+1	−1	−1	+1	−1	+1	+1	−1	−1	−1	−1	+1	+1	−1	+1	−1	−1	−1	−1	−1	+1	+1	+1
X_{21}	−1	+1	−1	−1	+1	+1	+1	−1	−1	−1	+1	−1	+1	−1	−1	−1	−1	+1	−1	+1	+1	−1	+1	+1	−1	−1	−1	−1	−1	+1	−1
X_{22}	+1	−1	+1	−1	+1	+1	−1	−1	+1	+1	+1	−1	−1	+1	−1	+1	−1	+1	−1	+1	−1	−1	−1	+1	+1	+1	−1	−1	−1	+1	−1
X_{23}	−1	+1	−1	+1	+1	+1	−1	+1	+1	+1	−1	−1	−1	−1	+1	−1	−1	−1	−1	+1	+1	−1	−1	+1	−1	+1	−1	+1	−1	−1	−1
X_{24}	+1	+1	−1	+1	+1	+1	−1	+1	+1	+1	+1	+1	−1	−1	−1	+1	+1	+1	+1	+1	−1	+1	+1	+1	+1	−1	−1	−1	+1	+1	+1
X_{25}	−1	+1	+1	+1	−1	−1	−1	−1	−1	−1	+1	+1	−1	+1	+1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	+1	+1	+1	+1
	Hits	13	13	13	13	14	16	13	17	15	13	13	13	17	14	13	13	17	14	14	13	16	16	13	14	14	14	15	17	17	16

Recall that the sign of $C_j(X_i)$ determines the classification, so $C_1(X_i)$ is a hit whenever the sign of $C_1(X_i)$ and the sign of z_i agree.

Next, following Algorithm 3.1 we use the weights w_i that have been computed according to equation (6) to determine that c_{13} gives us the smallest (weighted) sum of misses, with $W_2 = 7.031442$. We also compute the sum of the weights to find $W = 23.323808$, so that $r_2 = 7.031442/23.323808 = 0.301471$. It follows that

$$\alpha_2 = \frac{1}{2}\ln\left(\frac{1-0.301471}{0.301471}\right) = 0.420152.$$

We construct classifier $C_2(X_i)$ as given in line 20 of Algorithm 3.1, which for this specific case gives us

$$C_2(X_i) = C_1(X_i) + \alpha_2 c_{13}(X_i) = 0.376886\, c_8(X_i) + 0.420152\, c_{13}(X_i).$$

At the next iteration, we find that the minimum weighted sum of misses is $W_2 = 6.677488$, which occurs for classifier c_6. The sum of the all of the weights is $W = 21.406432$, and hence $r_2 = 6.677488/21.406432 = 0.311938$. We therefore have

$$\alpha_3 = \frac{1}{2}\ln\left(\frac{1-0.311938}{0.311938}\right) = 0.395536.$$

The classifier determined at this step of the algorithm is

$$C_3(X_i) = C_2(X_i)+\alpha_3 c_6(X_i) = 0.376886\, c_8(X_i)+0.420152\, c_{13}(X_i)+0.395536\, c_6(X_i).$$

We then continue with Algorithm 3.1 until all classifiers c_ℓ have been used. Of course, if all X_i are classified correctly at some iteration, then there is no need to continue beyond that point.

The classifiers C_m discussed here are summarized in Table 3. We have also included the first classifier that classifies all $n = 25$ samples correctly, namely, C_{21}.

4.2 A Bigger Example

We now consider a case where all of the available classifiers are extremely weak. In this example, we have $n = 100$ labeled samples, that is, we have (X_i, z_i), for $i = 1, 2, \ldots, 100$. Furthermore, we have $L = 1000$ classifiers c_ℓ. Of these 1000 classifiers, there are 482 for which $c_\ell(X_i) = z_i$ for exactly 51 of the 100 data points X_i, while each of the remaining 518 classifiers satisfies $c_\ell(X_i) = z_i$ for exactly 52 of the 100 data points X_i. In other words, each of the classifiers c_ℓ has an accuracy of 51% or 52%, with approximately the same number of classifiers having 51% accuracy as the number that have 52% accuracy.

Table 3: Classifiers C_1, C_2, C_3, and C_{21}, based on the data in Table 2

z_i	C_1	C_2	C_3	$\cdots$	C_{21}
-1	-0.376886	-0.797038	-0.401502	$\cdots$	-0.566786
$+1$	$+0.376886$	$+0.797038$	$+1.192575$	$\cdots$	$+1.604748$
-1	-0.376886	-0.797038	-1.192575	$\cdots$	-1.715969
$+1$	$+0.376886$	-0.043267	$+0.352270$	$\cdots$	$+3.077440$
-1	$+0.376886$	-0.043267	$+0.352270$	$\cdots$	-1.774858
$+1$	-0.376886	$+0.043267$	-0.352270	$\cdots$	$+0.026848$
-1	$+0.376886$	-0.043267	-0.438803	$\cdots$	-0.491584
$+1$	-0.376886	-0.797038	-0.401502	$\cdots$	$+0.299065$
-1	-0.376886	-0.797038	-0.401502	$\cdots$	-1.065321
$+1$	-0.376886	$+0.043267$	-0.352270	$\cdots$	$+0.081761$
-1	-0.376886	-0.797038	-1.192575	$\cdots$	-0.843112
$+1$	$+0.376886$	-0.043267	$+0.352270$	$\cdots$	$+0.291551$
-1	$+0.376886$	$+0.797038$	$+0.401502$	$\cdots$	-1.291802
$+1$	$+0.376886$	-0.043267	$+0.352270$	$\cdots$	$+2.225882$
-1	$+0.376886$	-0.043267	$+0.352270$	$\cdots$	-2.368660
$+1$	$+0.376886$	$+0.797038$	$+1.192575$	$\cdots$	$+2.369160$
-1	-0.376886	-0.797038	-1.192575	$\cdots$	-2.456064
$+1$	$+0.376886$	$+0.797038$	$+1.192575$	$\cdots$	$+1.363484$
-1	-0.376886	$+0.043267$	-0.352270	$\cdots$	-1.437239
$+1$	$+0.376886$	$+0.797038$	$+1.192575$	$\cdots$	$+0.800087$
-1	-0.376886	-0.797038	-0.401502	$\cdots$	-2.643773
$+1$	$+0.376886$	$+0.797038$	$+0.401502$	$\cdots$	$+0.580473$
-1	$+0.376886$	-0.043267	-0.438803	$\cdots$	-1.287581
$+1$	$+0.376886$	-0.043267	-0.438803	$\cdots$	$+1.943163$
-1	-0.376886	$+0.043267$	-0.352270	$\cdots$	-1.711700
Hits	17	17	18	$\cdots$	25

We experimented using all 1000 of the available classifiers c_ℓ, and also considered the case where we only use the first 500 classifiers, and the case where we use the first 250 of the c_ℓ. More details on these classifier subsets are provided in Table 4.

Table 4: Test data for Figure 1 ($n = 100$)

L	Hits	
	51	52
1000	482	518
500	246	254
250	124	126

The results of our AdaBoost experiments on this data are summarized in Figure 1. The red line is the case where all 1000 classifiers are available, the blue line represents the case where 500 of the classifiers are used, and the green line is for the case where only 250 of the classifiers are used. In each case, we have graphed the classification accuracy of the classifiers C_m, for $m = 1, 2, \ldots, 200$, which were constructed using AdaBoost, as given in Algorithm 3.1.

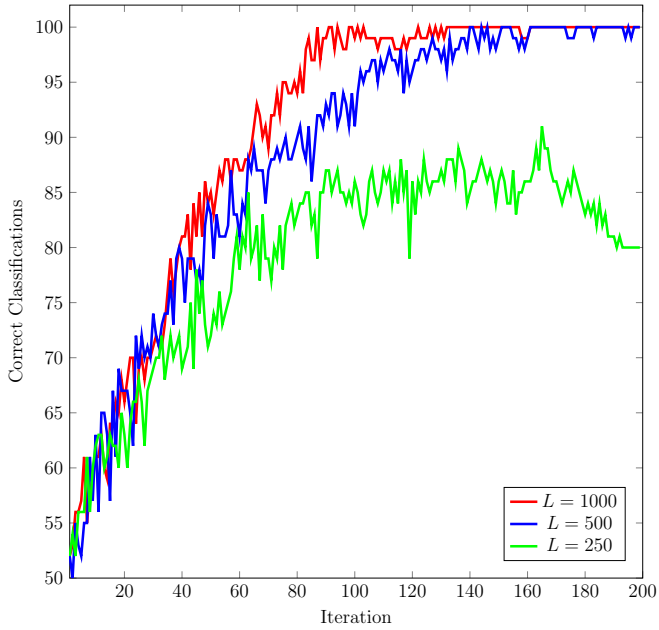

Figure 1: Correct classifications vs iteration ($n = 100$)

From Figure 1, we see that with $L = 1000$ classifiers—each of which is just marginally better than flipping a coin—we can obtain ideal accuracy using

about 100 classifiers. On the other hand, with $L = 500$ classifiers available, we need about 140 iterations before we achieve ideal classification, and with "only" $L = 250$ weak classifiers, we never reach more than about 90% accuracy.

Why do we obtain better results in Figure 1 when more classifiers are available? In terms of the football analogy, the more players that we have to choose from, the more likely it is that we'll find a player that better addresses our specific weaknesses at each iteration. The same principle holds true for the classifiers used in AdaBoost.

Given the weakness of the individual classifiers, each of the cases in Figure 1 is impressive. Nevertheless, these results do illustrate that AdaBoost is likely to require a large number of weak classifiers.

5 Exercises

1. Consider the example in Section 4.1.

   a) Use AdaBoost, as specified in Algorithm 3.1, to construct the next classifier, C_4, and give its classification results in the same form as for the classifiers in Table 3.

   b) Determine the minimum iteration m for which perfect classification is achieved and verify the results in Table 3 for classifier C_{30}.

2. For the $L = 250$ case in Section 4.2, determine the classifier C_{250} and give its classification accuracy. For this same case, determine all iterations m for which an accuracy of 90% or greater is achieved.

References

[1] R. Rojas. AdaBoost and the Super Bowl of classifiers: A tutorial introduction to adaptive boosting. http://www.inf.fu-berlin.de/inst/ag-ki/adaboost4.pdf, 2009.

[2] M. Stamp. Data for examples in "Boost your knowledge of AdaBoost". https://www.cs.sjsu.edu/~stamp/ML/files/adaData.zip, 2017.

[3] M. Stamp. *Introduction to Machine Learning with Applications in Information Security*. Chapman & Hall/CRC Press, 2017.