

# Boosting a Simple Weak Learner For Classifying Handwritten Digits

Matthew P. Carter  
Senior Honors Thesis (Advisor: Javed A. Aslam)

Department of Computer Science  
Dartmouth College  
Hanover, NH 03755

Computer Science Technical Report PCS-TR98-341

`matthew.p.carter.98@alum.dartmouth.org`, `jaa@cs.dartmouth.edu`

June 5, 1998

## Abstract

A weak PAC learner is one which takes labeled training examples and produces a classifier which can label test examples more accurately than random guessing. A strong learner (also known as a PAC learner), on the other hand, is one which takes labeled training examples and produces a classifier which can label test examples arbitrarily accurately.

Schapire has constructively proved that a strong PAC learner can be derived from a weak PAC learner. A performance boosting algorithm takes a set of training examples and a weak PAC learning algorithm and generates a strong PAC learner.

Our research attempts to solve the problem of learning a multi-valued function and then boosting the performance of this learner.

We implemented the AdaBoost.M2 boosting algorithm. We developed a problem-general weak learning algorithm, capable of running under AdaBoost.M2, for learning a multi-valued function of uniform length bit sequences.

We applied our learning algorithms to the problem of classifying handwritten digits. For training and testing data, we used the MNIST dataset.

Our experiments demonstrate the underlying weak learner's ability to achieve a fairly low error rate on the testing data, as well as the boosting algorithm's ability to reduce the error rate of the weak learner.

## 1 Introduction

### 1.1 Background

A PAC learning algorithm, also known as a strong learning algorithm, draws a training sample from an oracle and outputs a hypothesis  $h$  such that, with probability at least  $1 - \delta$ , the error of the hypothesis is at most  $\epsilon$ . Moreover, the training time and number of training examples required

must be polynomial in  $1/\epsilon$ ,  $1/\delta$ , and the size of the training sample. Moreover, the learner must perform this well on *any* sample distribution that the oracle provides.

A learning algorithm conforming to the PAC specification (i.e., a strong learner) seems difficult to come by. Nevertheless, Schapire demonstrated that a *boosting* algorithm can generate a PAC learner from a *weak learner*, the specification for which has less demanding requirements [Sch90].

A *weak learner* is just like a normal PAC learner, except that the error rate of the hypothesis it generates need only be slightly better than that of random guessing.

A *boosting algorithm* takes a weak learner and returns a strong learner. Virtually all boosting algorithms operate by modifying the distribution of the training sample and re-training with another weak learner to concentrate on the errors of the previous weak learners. AdaBoost.M1 and AdaBoost.M2 are two recently-developed boosting algorithms which both operate under this principle. [FS97] AdaBoost.M2 requires the weak learner to output, rather than simply a label, an array of confidences associated with each possible labeling of an example. Additionally, the weak learner of AdaBoost.M2 must have a training function which takes an array of penalties  $p$ , where  $p_i$  corresponds to the penalty that the weak learner will suffer by misclassifying the given example as an  $i$ . (The greater the weak learner's total penalties, the less it contributes to the final boosted learner.)

## 1.2 Practical Constraints

In spite of the aforementioned equivalence between weak and strong learning, it is usually very difficult to obtain a strong learner in practice. This is a result of the fact that it is usually very difficult to find a true weak learner. The requirement that a weak learner perform slightly better than random guessing *on any sample distribution* is crippling. Many learning algorithms perform well on the original training distribution, but when the distribution is modified by the boosting algorithm to concentrate on problematic training examples, the weak learner eventually breaks down and cannot classify with error less than  $1/2$ .

Even if a weak learner can be obtained in practice, other practical problems can arise.

First, information theory dictates that the size of the training sample must be on the order of  $\frac{1}{\epsilon} \log \frac{1}{\delta}$  in order to obtain a strong (PAC) learner. This many training examples can be hard to come by in practice.

Also, if the input sample contains noise (i.e., mislabeled examples), only a very specialized

learner/booster pair will not fail.<sup>1</sup> In most cases, the boosting algorithm will modify the training sample distribution to force the learner to concentrate on its errors, and will thereby force the learner to concentrate on learning mislabeled examples!

### 1.3 A Problem-General Weak Learner

We developed a weak learner which can be applied to any problem where the function to be learned is multi-valued, and the input examples are uniform length bit sequences. We designed the weak learner to work with AdaBoost.M2, as AdaBoost.M2 seemed to perform better than the other boosting algorithms that we tested.

#### 1.3.1 High Level Description

The weak learner keeps track of the differences between all target concepts. When asked to compute how confident it is that an example  $x$  has the label  $i$ , it cycles through all labels  $j$  (for all  $j$  not equal to  $i$ ) and computes its confidence that  $x$  is labeled  $i$  and not  $j$ , returning the minimum confidence across all possible values of  $j$ . Thus, if an incoming example is a handwriting bitmap whose correct label is 8 and the learner is asked how confident it is that the correct label is 3, the learner will have meaningless confidences that the label is 3 and not  $j$  (for all  $j$  not equal to 8) and a very low confidence that the label is 3 and not 8, thus giving it a low confidence that the label is 3. Naturally, the greatest (for all  $i$ ) minimum (for all  $j$ ) confidence will occur when  $i$  is 8, because the confidences that the label is 8 and not  $j$  are large for all  $j$ .

The confidence that the label of an example  $x$  is  $i$  and not  $j$  is calculated from the training examples whose labels are  $i$  and the penalties for classifying each of those as  $j$  as well as the training examples whose labels are  $j$  and the penalties for classifying each of those as  $i$ . For each bit in  $x$ , the probability  $p$  that the label of  $x$  is  $i$  rather than  $j$  given the value of the bit is calculated using Bayes Theorem. Let  $H(x)$  be the entropy function. If  $p > \frac{1}{2}$ , a vote of  $1 - H(p)$  is cast for  $i$ , otherwise, a vote of  $1 - H(1 - p)$  is cast for  $j$ . After casting votes for each bit of  $x$ , the total votes for  $i$  divided by the total votes cast is the confidence that the example has label  $i$  and not  $j$ .

#### 1.3.2 Pseudocode

$nb$  = number of bits in an example  
 $nl$  = number of possible labels

---

<sup>1</sup>Aslam showed that a PAC learning algorithm can be boosted in the presence of noise if it can learn via querying an oracle for statistics about the training data[Asl95]. However, the boosting algorithm in such a case cannot, itself, modify the distribution weights of individual training examples as AdaBoost.M1 and AdaBoost.M2 do.

```

Learn(example  $x_{1..nb}$ , label  $l$ , penalties  $p_{1..nl}$ ) {
  for  $i \in 1 \dots nl, i \neq l$ 
    average  $x$  into  $map_{l,i}$  with weight  $p_i$ 
}

```

```

Test(example  $x_{1..nb}$ ) {
  for  $i \in 1 \dots nl$ 
     $MinScoreOfI = 1$ 
    for  $j \in 1 \dots nl, j \neq i$ 
       $votes_i = 0$ 
       $votes_j = 0$ 
      for  $b \in 1 \dots nb$ 
        if ( $b == 1$ )
           $BitBeliefINotJ = \frac{map_{i,j}}{map_{i,j} + map_{j,i}}$ 
        else
           $BitBeliefINotJ = \frac{1 - map_{i,j}}{(1 - map_{i,j}) + (1 - map_{j,i})}$ 
        if ( $BitBeliefINotJ > 0.5$ )
           $votes_i = votes_i + 1 - H(BitBeliefINotJ)$ 
        else
           $votes_j = votes_j + 1 - H(1 - BitBeliefINotJ)$ 
       $BeliefINotJ = \frac{votes_i}{votes_i + votes_j}$ 
      if ( $BeliefINotJ < MinScoreOfI$ )
         $MinScoreOfI = BeliefINotJ$ 
       $confidence_i = MinScoreOfI$ 
  return  $confidence_{1..nl}$ 
}

```

## 2 Experiments

We tested the ability of our weak learner, in combination with AdaBoost.M2, to classify handwritten digits from the MNIST Database.<sup>2</sup> This dataset contains handwritten digit images which are

---

<sup>2</sup><http://www.research.att.com/~yann/ocr/mnist/>

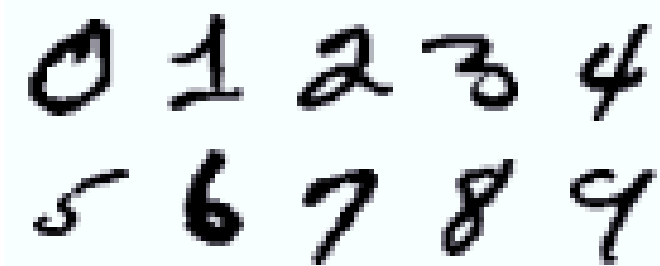


Figure 1: Example training patterns from the MNIST database of handwritten digits.

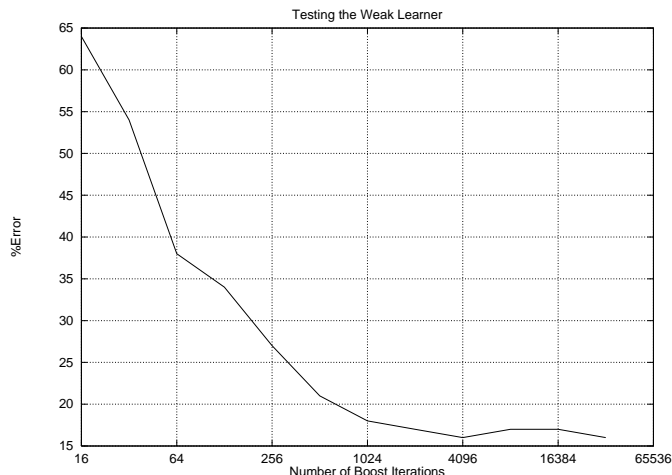


Figure 2: The performance of a weak learner versus the size of the training sample.

size-normalized and centered; therefore, we did not pre-process the images.<sup>3</sup> Some representative example patterns from the training dataset are shown in Figure 1.

We began by testing the effectiveness of the weak learner. Figure 2 shows the relationship between the size of the training sample and the error rate of the weak learner on the test data. Initially, the weak learner does not have enough training data with which to generalize, so the error rate is near 90% (which corresponds to random guessing over 10 possible labels).

Next, we demonstrated the phenomenon known as *overfitting*. When the boosting algorithm repeatedly trains the weak learners on the same data, the weak learners eventually “memorize” the training patterns, so that their performance on the (different) testing patterns reaches a minimum, then deteriorates with further boosting. (Please see Figure 3.) The smaller the size of the training sample, the sooner this deterioration occurs.

To evaluate the overall performance of our weak learner boosted by AdaBoost.M2, we trained on 60000 examples and tested on 10000 examples. (Please see Figure 4.) It is apparent that,

---

<sup>3</sup>De-skewing the images has, however, been found to be beneficial by some other researchers who have used this dataset.

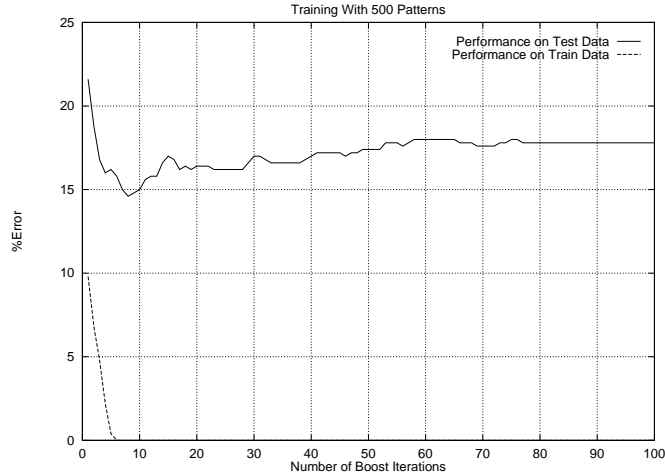


Figure 3: The performance of an under-trained weak learner across rounds of boosting.

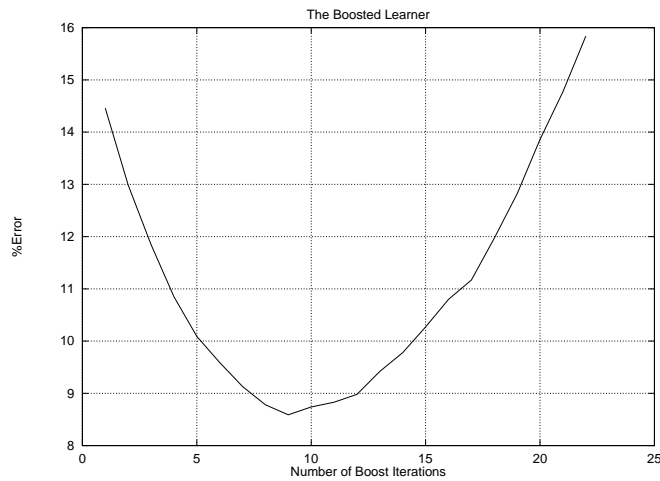


Figure 4: The performance of the boosted learner with 60000 training examples.

even with 60000 training patterns, the weak learner’s performance deteriorates after 10 rounds of boosting as a result of overfitting. This demonstrates a combination of two possible problems: first, a lack of sufficient data to achieve truly low error rates; and second, a weak learning algorithm perhaps somewhat deficient for this particular task.

### 3 Conclusions

We have demonstrated the use of AdaBoost.M2 to boost a weak learner in order to solve the problem of classifying handwritten digits. The underlying weak learner begins with an error rate of 14.5% and eventually reaches 8.59% after 9 rounds of boosting. These error rates, however, are rather high, even for the boosted learner which trained on 60000 patterns (Figure 4).

The reason for this, though, is that our underlying weak learner is not designed specifically to solve the problem of handwritten digit recognition. If that were the case, we would certainly have used a very different weak learner, and we would probably have pre-processed the data as well.

## 4 Future Research

We hope to modify the weak learner to use priors (which take into account the distribution of examples  $i$  and  $j$ ) in the calculation of the belief that an example is labeled  $i$  and not  $j$ .

We would also like to experiment with different underlying weak learners, such as C4.5 and neural networks.

## 5 Acknowledgments

We are very grateful to Yann LeCun for referring us to the MNIST digit database. We would also like to thank Holger Schwenk, Robert Schapire, and Harris Drucker for information on obtaining handwriting sample data. We would like to give a special thanks to Robert Schapire for keeping in touch with us throughout our research and sharing ideas about AdaBoost. Finally, we would like to thank David Kotz for his helpful suggestions and his seemingly endless supply of both knowledge and patience.

## 6 Availability

Our source code may be downloaded at “<ftp://ftp.cs.dartmouth.edu/TR/TR98-341.code.tar.Z>”.

## References

- [Asl95] Javed A. Aslam. *Noise Tolerant Algorithms for Learning and Searching*. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, February 1995.
- [FS97] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer Science and System Sciences*, 55(1):119–139, 1997.
- [Sch90] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, June 1990.