Analyzing Margins in Boosting

Lev Reyzin
Princeton University Class of 2005, BSE
Senior Independent Work
Advised by Professor Robert Schapire

Abstract

While the success of boosting or voting methods has been evident from experimental data [11], questions about why boosting does not overfit on training data remain. One idea about the effectiveness of boosting was given by Schapire et al. in which they presented an explanation of why the test error of a boosting classifier does not increase with its size by looking at margins. They showed that boosting increases the margins of training examples effectively and that this increase in margins is related to the auspicious performance of AdaBoost [12]. Breiman invented ArcGv, an algorithm that increases margins even more aggressively than AdaBoost, yet does not perform better. He claimed this contradicts Schapire's margins explanation, since all else being equal a higher margins distribution did not in this case indicate better results [1]. In this paper, we experimentally examine Breiman's results by comparing the margins and performance of AdaBoost and ArcGv. We also analyze our approaches and show that our results partially contradict Breiman's findings. We also explore some other voting algorithms' performance and present some studies only indirectly related to this problem. We also present possibilities for further research and experimentation in this area, which it is our intention to continue pursuing.

1 Introduction and Previous Work

The problem of machine learning with regard to data classification can usually be expressed in the following way. Given already-classified data on which an algorithm can be trained, predict the classifications of future data. The data that the algorithm trains on is called training data. The data that needs to be classified is called test data. The algorithm that classifies the test data is called a classifier. The test error is the percent of test data misclassified.

If many classifiers are consistent with the training data, *Occam's razor* states that we should choose the simplest one. This intuitively makes sense because it is the role of a classifier to find patterns in training data instead of being so complex as to be adjusted to fit the training data [9]. When too complicated

a classifier is used and thus does not predict well on test examples, it *overfits* the training data. This is generally true for most algorithms, yet in the case of boosting this intuition fails.

1.1 Boosting

Freund and Schapire's AdaBoost [4] is a type of boosting algorithm that uses the combined vote of weak learners. Weak learners are themselves classifiers, but they are usually very simple classifiers that are not made to reduce the training error to zero. The idea being that while each individual weak learner is innacurate, their weighted vote will be. As the number of boosting rounds increases, the overall classifier becomes more complicated, since it becomes the combined vote of more weak classifiers, and the training error falls. Even after the training error reaches zero AdaBoost's test error continues to fall [11]. This violates the expectation that AdaBoost would overfit on the training data.

Boosting works in the following way (borrowed from the formulation by Schapire)[11]:

Given a binary dataset, let D_t be the vector of weights d_t that represent the weights of the N training examples, where $d_t(i)$ is the weight at time t of example i. For all i from 1 to N, initialize $D_1(i) = 1/N$. Let α_t represent the weight of h_t, the hypothesis generated at time t. Loop t from 1 to T, the number of rounds of boosting. On each iteration, do the following:
find h_t with minimum error with respect to D_t, then set α_t, the weight of the hypothesis by the rules of the boosting algorithm being used.

$$\alpha_t = f(h_{1,2,\ldots,t}, \alpha_{1,2,\ldots,t-1}, x_{1,2,\ldots,N}, y_{1,2,\ldots,N}, D_t) \tag{1}$$

Then, for each i from 1 to N update the distribution D for the next time step by

$$D_{t+1}(i) = \frac{D_t(i)exp(-\alpha_t y_i h_t(x_i))}{Z_t} \tag{2}$$

where Z_t is the normalizing vector. So, now that h_t and its corresponding α_t are calculated, continue with the loop.

When the training stage ends, the final combined classifier H predicts according to the following rule on datapoint x:

$$H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x)) \tag{3}$$

1.2 Margins

Notice that the vote of the combined classifier is binary. If the result of the summation from the equation above is positive, the datapoint is classified into one category. If it is negative, the datapoint is classified into another. The

amount of confidence in the final classification decision is lost in the binary output of the classifier.

However, the measure of confidence can be extracted by looking at the summation itself. If all of the classifiers voted in the same way, the sum would be positive or negative $\sum_{t=1}^{T} \alpha_t$ hence, we can define the "confidence" as

$$\text{confidence}_f(x) = |\frac{\sum_{t=1}^{T} \alpha_t h_t(x)}{\sum_{t=1}^{T} \alpha_t}| \tag{4}$$

The margin can now be defined as that value multiplied by 1 if the classification was correct and by -1 otherwise. This is equivalent to

$$\text{margin}_f(x, y) = y\frac{\sum_{t=1}^{T} \alpha_t h_t(x)}{\sum_{t=1}^{T} \alpha_t} \tag{5}$$

where y is the sign of the actual label.

A margin can be described in the following way. A high (positive) margin means that the classifier predicted correctly and with high confidence. A large negative margin means that the classifier mispredicted, yet was confident in its prediction. Margins express the confidence of the predictor in their size and its correctness with their signs. Margins are usually measured over training data, but we measured them for test data as well. The training margins can be measured once a final classifier has been determined, and it is run once again on the training data. If the dataset is seperable, as in if the training error reaches zero, then the training margins will all be positive. Whether this is possible or not relies on the dataset, the weak learner, and the boosting algorithm used.

In a paper by Schapire, Freund, Bartlett, and Lee, it was shown that a bound on the test error can be derived from [12] the margins distribution. While this bound is loose, as in algorithms perform with error well under this cap, it was a relation of the margins to the test error. In their paper, they said that given this connection, higher margins will be responsible for better classification. Breiman challenged this claim by finding a boosting algorithm he claims does not hold to this rule. His algorithm is described later in the paper.

In this paper we will present and evaluate these claims. This work is important because such a comparative experimental of margins has not yet been done, and its results will shed light on better understanding the algorithms' performances. We will study both AdaBoost and Breiman's algorithm, ArcGv, but using different weak learners. We will show that his findings do not necessarily contradict Schapire's claim and that further study yields results consistent with his mathematical claims, but not supporting his experimental data.

2 The Algorithms

In this section, we will present the boosting algorithms that have been studied. These are not the only boosting algorithms in existence, but they are the ones at the focus of our work.

2.1 AdaBoost

AdaBoost [4] is the boosting algorithm most widely used, and the term "boosting" is often used to mean "AdaBoost." In AdaBoost, the boosting framework is used, and α_t is set in the following way [11]

$$\alpha_t = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t} \tag{6}$$

where ϵ_t is the sum of all of the weights at time t of the misclassified examples by hypothesis h_t

$$\epsilon_t = \sum_{n=1}^{N} d_n^{(t)} I(y_n \neq h_t(x_n)) \tag{7}$$

2.2 ArcGv

ArcGv was first introduced by Breiman as an algorithm that increases margins more aggressively than AdaBoost. Breiman showed that ArcGv will have a greater smallest margin than AdaBoost [1]. This algorithm was reformulated by Meir et al. [7]
In ArcGv, the α_t is set in the following manner

$$\alpha_t = \frac{1}{2} \log \frac{1 + \gamma_t}{1 - \gamma_t} - \frac{1}{2} \log \frac{1 + \varrho}{1 - \varrho} \tag{8}$$

where we define

$$\gamma_t = 1 - 2\epsilon_t \text{[1]} \tag{9}$$

now let ϱ be the minimum margin of the hypotheses [2]

$$\varrho = \min_{n=1,\ldots,N} y_n f_{t-1}(x_n) \tag{10}$$

where

$$f_i(x) = \frac{\sum_{t=1}^{i} \alpha_t h_t(x)}{\sum_{r=1}^{i} \alpha_r} \tag{11}$$

now α_t can be expressed in a manner that would make it easy to compute

$$\alpha_t = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t} - \frac{1}{2} \log \frac{1 + \varrho}{1 - \varrho} \tag{12}$$

This algorithm is **ArcGv**.

In Meir's reformulation of ArcGv, Meir used a slightly different setting of ϱ [7], mainly

$$\varrho = \max\left(\varrho_{t-1}, \min_{n=1,\ldots,N} y_n f_{t-1}(x_n)\right) \tag{13}$$

[1] γ_t is also called the *edge* [7]

[2] In his original formulation of the algorithm, Breiman [1] used the notion of *top* instead of margins, where *top*, $\bar{t} = \frac{1}{2} - \frac{1}{2}\varrho$. He originally set α to $\frac{1}{2} log(\frac{1-\epsilon_t}{\epsilon_t} \frac{\bar{t}_t}{1-\bar{t}_t})$, which is mathematically equivalent to our expression

we will call this modified version of ArcGv, **ArcGvMax** to emphasize the "max" term.

The ArcGv algorithm was designed to maximize the minimum margin on training data.

2.3 Boosting on a Smooth Margin: Coordinate Assent

In their paper, Rudin et al. defined an algorithm called Coordinate Assent Boosting that aimed to maximize the margin explicitly. [8]

Their algorithm defined a Smooth Margin function, G, reformulated to be computed iteratively as

$$G = \frac{\sum_{i=1}^{N} e^{-y_i \sum_{t'=1}^{t} \alpha_{t'} h_{t'}(x_i)}}{\sum_{j=1}^{j=t} \alpha_j} \tag{14}$$

using the boosting framework described in 1.1, setting α by equation (12) with ϱ set to G gives the Coordinate Assent algorithm below

$$\alpha_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t} - \frac{1}{2} \log \frac{1+G}{1-G} \tag{15}$$

This algorithm, as presented in [8] uses AdaBoost's update rule until the value of G becomes positive. We use this rule from the beginning, though an exprimental comparison of the two may be interesting.

3 Approach

Our approach was to implement the algorithms above and to look at their margins distributions and performance on test data. Our goal was to test Breiman's theory, so we first focused on implementing AdaBoost and ArcGv, but we used decision stumps as a weak learner.

3.1 Choosing a Weak Learner: Decision Stumps and Decision Trees

In this section, we will present the various weak learners considered or used in our approach.

Decision stumps work in the following way: given all labeled weighted training data, find the optimal attribute to decide on. This is the task of finding the single attribute of the datapoints which best correlates, directly or inversely, with the training labels. Since this process is weighted, as the weights of the datapoints change, different attributes will give the best weighted results.

Decision trees work with the same idea, only they split on an attribute given some criterion. The decision tree algorihtm Breiman employed is called CART (Classification and Regression Trees) and was presented in his book by that name [2]. The splitting criterion used by this algorithm is called the Gini

index. Then, the tree is not just grown to one node as in decision stumps, but to more levels, given the specification. Then, usually the tree is pruned so that it is not complicated enough to overfit the training data.

The Gini Index is a measure of the *impurity* of a given split. The impurity of a split, t, is

$$i(t) = 2p(1|t)p(2|t) \tag{16}$$

So, in binary classification, the impurity of a split is the proportion of labels of class 1 of a split times the proportion of labels of class 2 of a split. The imurity of a split is the average of the impurities of both sides of a split. The object is to choose a split of least impurity at each level. The CART algorithm uses the Gini index splitting critereon and prunes the tree to find the best tree of k nodes [2].

Our work focused on using decision stumps instead of decision trees. we felt it was more important to look at the behavior of boosting algorithms using stumps for several reasons. Primarily, decision stumps are a simpler algorithm and thereby have certain properties. For example, decision stumps evaluate the optimal case in the sense that it is always possible to find a decision stump that splits the training data optimally because there are one as many possible splits as there are attributes, times 2, for direct and inverse correlation of the label with the attribute. So, if a is the number of attributes, at the formation of each hypothesis, only $2a$ attributes need to be considered. However, it is harder to find the optimal decision tree. This is an exponential computation, and there are important mathematical properties that are not present in the non-optimal case.

3.2 Measuring Test Error

The first and perhaps the most obvious approach to measuring the difference between two algorithms would involve looking at how well they predict on the various datasets. We wanted to vary both the number of training examples and the number of boosting rounds to see how well each algorithm predicts.

Obviously, since both Schapire's and Breiman's claims say something about the test error's relationship to the margins distributions, it was important to measure the test errors to see any correlations.

3.3 Looking at Margins

It was important to measure the margins distributions with respect to some of the same experiments for which the test errors were measured. Given that we had access to labels of test examples, we also looked at the margins distributions over both training and test data to see the differences in the distributions depending on whether the algorithm was trained on that data or not. (Training margins are the ones normally examined) In particular, we wanted to test the following:

Whether ArcGv has the same test error and margins distribution as ArcGvMax. When breiman suggested the ArcGv algorithm, it was written in a form equivalent to ArcGv presented in this paper. Meir could prove the same results, but using ArcGvMax, which includes the extra term. We suspected that the distributions would look almost identical since in order for the minimum ϱ to decrease, its

$$\min_{n=1,\ldots,N} y_n f_{t-1}(x_n)$$

term must decrease with the formation of a new hypothesis. However, the argmin of that term would be the example most often misclassified, and the chances of the next hypothesis misclassifying it were low since new hypotheses focus on classifying the most misclassified data correctly. There is still a possibility of another term becoming smaller than the previous min, but that is also unlikely since the object of the boosting algorithms is to continue driving training data misclassifications down. Yet, we wanted to find out experimentally (and confirm or disprove our belief) if the max term had any impact on the predictions, their corectness, or the confidence in them. Hence, we examined the margins distributions of the two.

Seeing how the test and training margins differ for various learning algorithms. Since it was included in our experiments to look at both training and test data margins, we wanted to see if and how the training and test margins were correlated for the various algorithms. We expected that the overall margins distributions to be shifted rightward for training data compared to test data since algorithms are more likely to predict better on the data they are trained on. In fact, we know that if the data is *seperable*, boosting can drive training error to 0, but that does not mean it will predict perfectly on test data. However, we did not know if the two margins curves for a given algorithm were similar. We plotted the two against each other to observe any correlation.

Comparing AdaBoost and ArcGv (and Coordinate Assent). We wanted to see whether the correlations Breiman noted between the error and margins of ArcGv and AdaBoost hold over decision stumps. This is the main stud presented in this paper. We also compared the margins distributions with those of Coordinate Assent. The studies involved both test and training margins and were analyzed to shed light on Breiman's claim.

4 Methodology

The implementations of the algorithms relied on that they were all boosting algorithms. Hence, we first implemented the boosting framework as described in Section 1.1. Then, each algorithm needed to be reformulated in a format such that it would fit the framework and all that needed to be changed was the setting of the alpha. This was presented in section 2 in equation (1). Finally, they were tested on various datasets. Implementation detrails and descriptions of the datasets are presented below.

4.1 DataSets

All of the datasets [10] were converted to a format using binary attributes and binary labels. They are listed from easiest to hardest to predict on, as evidenced in our experiments.
OCR 17 - Given pixel locations of images, classifying them as ones or sevens. This is called an OCR task (Optical Character Recognition).
OCR 49 - A similar OCR task, differentiating fours from nines.
DNA - Given DNA sequences, predict whether they are "splice" or "non-splice."
Census - Given binary data against individuals' location, marital status, age, etc., classify them into income categories of making > 50k or ≤ 50k per year.

4.2 Algorithm Implementations

The implementation of AdaBoost involved setting α as in equation (6). The implementation of Coordinate Assent involved setting α by equation (15). AcrGv can be inplemented by using equation (12), but a more efficient solution is possible.

4.2.1 ArcGv - An Efficient Implementation

If the algorithm using the general boosting framework is followed, with α_t set by (12), the time required to do the task for T boosting steps and N examples takes looking at the misclassification of each hypothesis on each example to find ϵ by (7) is $O(NT)$. The time to determine ϱ by (10) is $O(NT^2)$. This is because for each new hypothesis, one must look at how all of the previous hypotheses performed on each example. So, the running time of the brute-force implementation is $O(NT^2)$.

Yet, given $O(N)$ space we can reduce the time complexity. By allocating a vector of size N, R, where in equation (11) instead of doing the full summation, we can store the following

$$R_t(n) = \sum_{t'=1}^{t} \alpha_{t'} h_{t'}(x_n) \tag{17}$$

and keep track of the sum of the alphas in α_{sum}. We can then find the solution to (10), below

$$\varrho = \min_{n=1,\ldots,N} y_n f_{t-1}(x_n)$$

recursively by the following algorithm in computing

$$f_i(x_n) = \frac{R_{t-1}(n) + \alpha_i h_i}{\alpha_{sum} + \alpha_i} \tag{18}$$

and updating α_{sum} and the vector, R for each example, n

$$R_t(n) = R_{t-1}(n) + a_i h_i$$

This way, ArcGv can be implemented by doing a constant number of computations for N examples for T rounds, reducing the running time to $O(NT)$.

4.2.2 Decision Tree Pruning - An Dynamic Programming Solution

There also exists a known efficient solution to the CART decision tree weak learner. This can be used to reduce the computation of which k nodes of a grown tree to select on to polynomial. This still does not give the optimal solution because the tree being pruned is not an optimally generated tree to begin with, but instead uses the Ginny index to select attributes.

Given a tree, to find an optimal subtree of k nodes, define $f(n, k)$ to be the error rate for the split on node n of the best subtree rooted at n of size $\leq$ k. Define $e(n)$ to be the error if n is the node, n, is a leaf. Let

$$f(n, k) = \min\left(e(n), f(n_0, k') + f(n, k - k' - 1)\right) \tag{19}$$

This recursive procedure finds the minimum error (defined by some function) and considers leaving each node encountered as a leaf or splitting on it. This seems like an exponential computation, but it has a dynamic programming solution [6]. One can define a k by n matrix and since for any call to f(x,y), x $\leq$ n and y $\leq$ n, the functions called recursively need only be computed at most nk times. This reduces the brute force exponential algorithm to a polynomial time one [3].

The implementation of decision stubs involves generating the tree at depth 1, and for that, it is efficient to try all possibilities. In our experimentss, we only used decision stubs, but Breiman's work is based on finding best decision tree subtrees of size k [2] [1].

5 Results

In this section, we will present the results of all of the experiments set up in the Approach section. We will also present tangental findings not included in our initial approach. *All algorithms herein used Decision Stumps as weak learners.*

5.1 Test Errors and Training Set Size

The following test errors were measured for the following algorithms, varying the size of the training set, but keeping the number of boosting rounds constant across all algorithms.

	ArcGv	ArcGvMax	AdaBoost	C-Assent
50	4.47%	4.70%	4.40%	4.00%
100	3.93%	3.93%	2.37%	2.67%
200	2.53%	2.57%	1.47%	1.57%
500	1.23%	1.23%	1.07%	1.10%
1000	1.10%	1.10%	0.87%	0.80%

Figure 1: Test errors for the **OCR 17** dataset with 100 rounds of boosting

	ArcGv	ArcGvMax	AdaBoost	C-Assent
50	14.17%	14.33%	14.70%	14.03%
100	12.90%	12.53%	11.00%	12.17%
200	9.00%	9.07%	8.73%	9.40%
500	7.10%	7.27%	7.77%	8.07%
1000	6.87%	6.93%	6.77%	7.43%

Figure 2: Test errors for the **OCR 49** dataset with 100 rounds of boosting

	ArcGv	ArcGvMax	AdaBoost	C-Assent
50	16.76%	17.33%	18.53%	19.53%
100	13.23%	13.99%	13.55%	13.11%
200	9.89%	9.89%	9.33%	11.09%
500	8.38%	9.07%	8.38%	9.96%
1000	6.68%	6.68%	6.43%	8.89%

Figure 3: Test errors for the **DNA** dataset with 100 rounds of boosting

	ArcGv	ArcGvMax	AdaBoost	C-Assent
50	24.33%	24.53%	23.33%	26.73%
100	25.53%	25.17%	22.87%	24.00%
200	21.00%	21.43%	20.67%	22.33%
500	19.23%	19.33%	19.03%	19.70%
1000	18.60%	18.23%	18.03%	19.10%

Figure 4: Test errors for the **Census** dataset with 100 rounds of boosting

This experiment was designed to test whether there was a significant difference between the algorithms' test errors and to see if there is any difference the rate at which the test errors fall for the algorithms. As expected, the test errors fell for all four algorithms as the training set and the error rates fell dramatically with initial increases but then continued falling at slower rates.

As for a comparison of the test errors, AdaBoost seemed to slightly outperform the other algorithms on most tasks, but the difference was not strong and may not be significant. ArcGv sometimes outperformed Coordinate Assent, and Corrdinate Assent sometimes outperformed ArcGv depending on the dataset.

5.2 ArgGv and ArcGvMax

Our comparison of the margins distributions yields the following results. This graph is representative of the similarity in the margins distributions of the two.

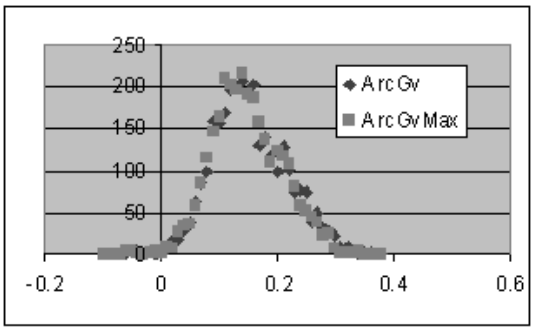

Figure 5: Test error margins v frequency plots for 100 rounds of boosting on the OCR 17 task

There was no noticable difference in the prediction rates of ArcGv and ArcGvMax, nor was there a noticable difference in their test margins distributions, meaning they were equally confident and correct in their predictions. Given this evidence, we concluded that we could not find that the *max* term had any affect on ArcGv and from this point forward, we only used ArcGv as Breiman formulated it. This result was expected, as discussed in Section 3.3.

5.3 Test Errors and Boosting Rounds

To perform the main part of our experiment, we needed to obtain the test errors for AdaBoost and ArcGv as a function of the number of rounds of boosting of the various datasets. For all of these experiments, 1500 training examples were used. Hence, this data differs from the data in **Section 5.1** because of the different number of training examples used.

	Algorithm	
rounds	AdaBoost	ArcGv
100	0.77%	0.67%
200	0.70%	0.70%
500	0.73%	0.73%
1000	0.77%	0.80%
5000	0.73%	0.83%

Figure 6: Test errors for the **OCR 17** dataset

	Algorithm	
rounds	AdaBoost	ArcGv
100	6.23%	5.80%
200	6.00%	6.17%
500	6.10%	6.23%
1000	6.47%	6.43%
5000	7.07%	7.00%

Figure 7: Test errors for the **OCR 49** dataset

	Algorithm	
rounds	AdaBoost	ArcGv
100	5.55%	6.18%
200	5.73%	6.24%
500	6.62%	7.50%
1000	6.93%	7.31%
5000	8.89%	8.82%

Figure 8: Test errors for the **DNA** dataset

	Algorithm	
rounds	AdaBoost	ArcGv
100	18.60%	18.57%
200	18.47%	18.53%
500	18.43%	18.47%
1000	18.63%	18.57%
5000	19.00%	18.47%

Figure 9: Test errors for the **Census** dataset

Here, we notice that AdaBoost and ArcGv outperform each other, depending on the dataset and the slight advantage AdaBoost seemed to have is not present when the number of boosting rounds was increased. AdaBoost, however, still performed better on the OCR 17 task, especially with great numbers of boosting rounds. Hence, we cannot contradict Breiman's result that AdaBoost outperforms ArcGv [1], especially since we used a different weak learner.

One interesting result to note is that both algorithms seemed to overfit on the DNA dataset as the number of boosting rounds increased. Such a result was unexpected since AdaBoost has been observed to usually overfit after a greater number of boosting rounds [5]. After studying this phenomenon, we discovered that for the DNA dataset, the algorithms quickly begin to produce weak learners (Decision Stumps) whose error rates on the weighted training data set become closer and closer to .5 (yet remained below). So, while each new weak learner

continued to try to classify the datapoints the previous ones misclassified, the overall predictor became worse, which is exactly the case of overfitting. We, however, do not know what properties of the DNA dataset caused this to occur.

5.4 The Margins of AdaBoost and ArcGv

The margins distributions are at the heart of this study. We computed the margins distributions for all four datasets with 1500 training examples and for various rounds of boosting. We computed margins for both training and text predictions.

5.4.1 Margins on Training Data

The following were margins taken on training examples after the final classifier was determined. We will only show the data for one of the datasets for space considerations and describe any relevant differences between these results and the results for the other datasets later on.

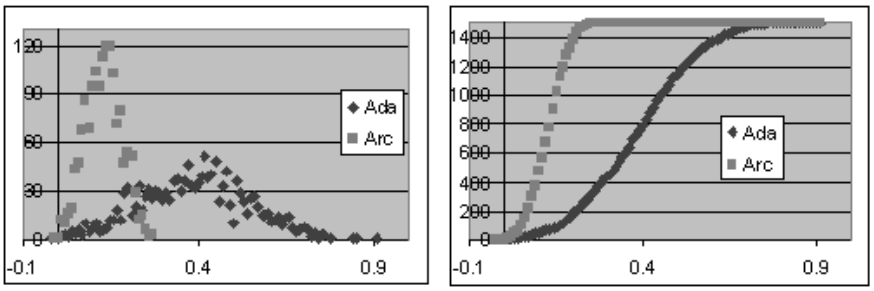

Figure 10: Training margins frequency disribution, left, and cumulative distribution, right, for **50** rounds of boosting

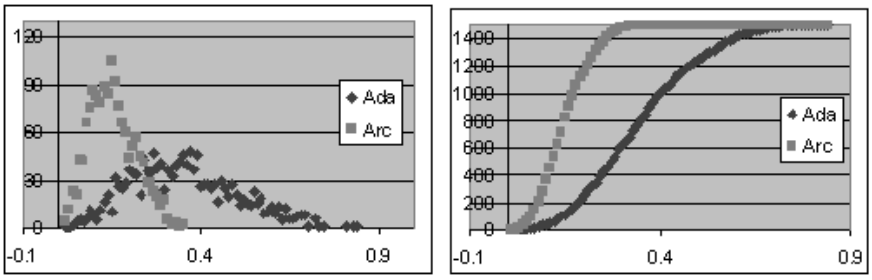

Figure 11: Training margins frequency disribution, left, and cumulative distribution, right, for **100** rounds of boosting

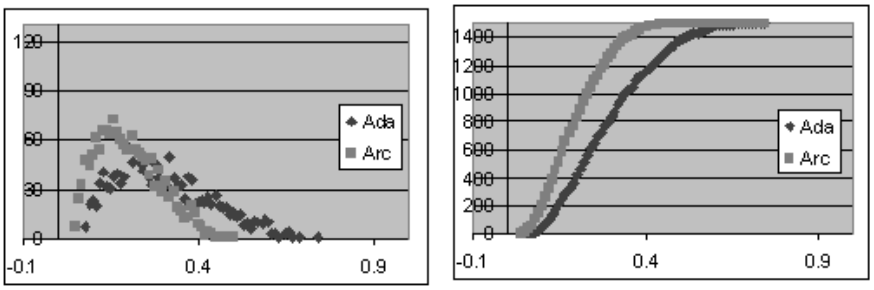

Figure 12: Training margins frequency disribution, left, and cumulative distribution, right, for **500** rounds of boosting

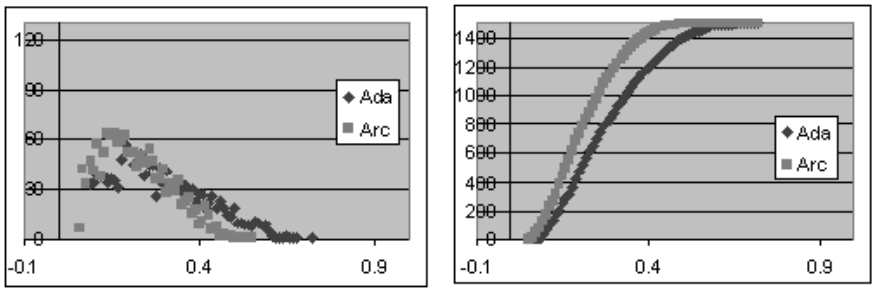

Figure 13: Training margins frequency disribution, left, and cumulative distribution, right, for **1000** rounds of boosting

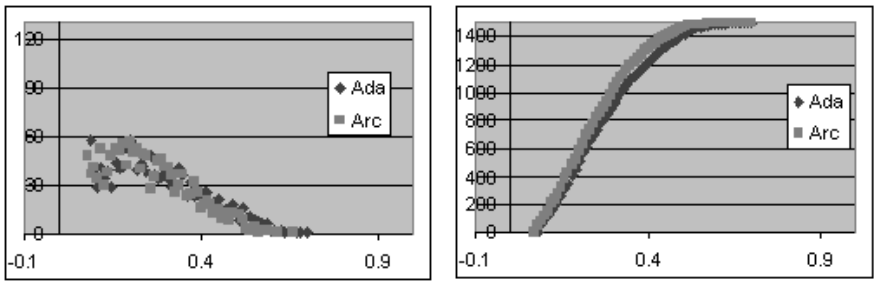

Figure 14: Training margins frequency disribution, left, and cumulative distribution, right, for **5000** rounds of boosting

These results make the cornerstone of our conclusions. As we can see, several interesting patterns can be observed in the comparison of AdaBoost and ArcGv. Even though it is hard to see on these plots, ArcGv always has a higher

minimum margin than AdaBoost, as Breiman showed [1], but the cumulative margins distributions for AdaBoost are still to the right of ArcGv's, which is the opposite result of Breiman's [1]. Even though the minimum margin is higher for ArcGv, it does not mean that its margins need also be higher for the entire distribution, which they were not. In fact, while ArcGv experimentally had the higher minimum margin, it is AdaBoost that consistently had the higher maximum margin. This seems consistent with Schapire's margins explanation [12] assuming that AbaBoost outperforms ArcGv. However, AdaBoost had higher training margins even in the datasets where it performed slightly worse.

The second and perhaps more interesting result is that the margins distributions for AdaBoost and ArcGv converged to be the same as the number of boosting rounds increased.[3] We are not sure why this is the case, yet it appears to hold true over many datasets.

One analysis shows the following. Since in equation (10), ϱ is set to be the minimum margin of the hypotheses, as the number of rounds of boosting increases, the smaller the chance that the a new hypothesis will make a difference in the mininum margin. Hence, as N increases, α_t tends toward

$$\alpha_t = \frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t} - \frac{1}{2}\log\frac{1+C_1}{1-C_1}$$

$$\alpha_t = \frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t} - C_2 \tag{20}$$

where C_1 and C_2 are constants. This may seem promising since it may seem that in the long run ArcGv is a constant term different from AdaBoost, only the constant term does not remain so in the computation of the distribution (2). The computation becomes the following when substituting for α in ArcGv

$$D_{t+1}(i) = \frac{D_t(i)exp(-(\frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t} - C)y_i h_t(x_i))}{Z_t}$$

$$D_{t+1}(i) = \frac{D_t(i)exp(-(\frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t})y_i h_t(x_i) - C(y_i h_t(x_i)))}{Z_t} \tag{21}$$

While in AdaBoost, the α setting is simply

$$D_{t+1}(i) = \frac{D_t(i)exp(-(\frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t})y_i h_t(x_i))}{Z_t} \tag{22}$$

This means that ArcGv, in the limit of increasing the number of boosting rounds, contains the extra $C(y_i h_t(x_i))$ term, which while in the exponentiation, is no longer a constant away from Adaboost, and disproves this easy explanation. However, this intuition may still have something to do with why the two algorithms converge in their margins distributions.

Finally, despite the overfitting that occurred with the DNA dataset, the margins distributions for the DNA dataset still followed the same pattern as they did for the OCR 17 data in the figures above.

[3]This was only not clear to be the case for the Census dataset, though there is experimental evidence that the two distributions may converge. This may be tied to the Census dataset not being seperable for this algorithm.

5.5 Margins on Test Data

The following data are the margins on the predictions on the test data. We will again use the OCR 17 task as the representative dataset so that we can see correlations between the training and test margins. We have used the same set of numbers of boosting rounds as for the Margins on Training Data section, and we tested all results on a set of 3000 test examples. Any differences between this representative dataset and others will be described later.

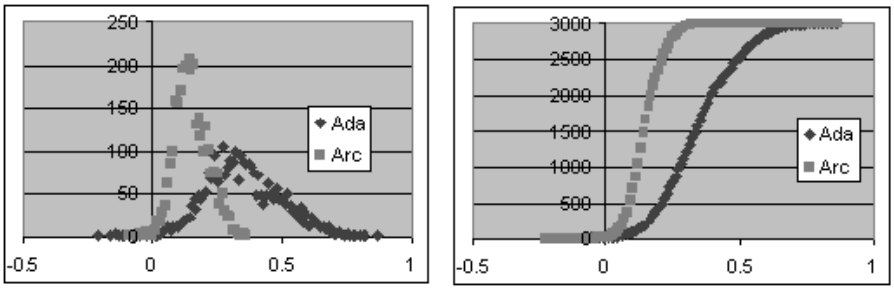

Figure 15: Test margins frequency disribution, left, and cumulative distribution, right, for **50** rounds of boosting

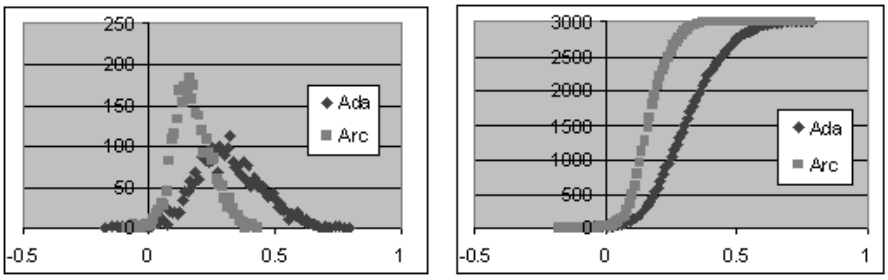

Figure 16: Test margins frequency disribution, left, and cumulative distribution, right, for **100** rounds of boosting

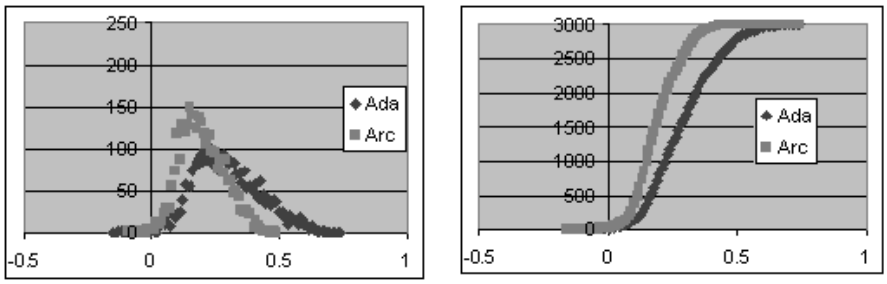

Figure 17: Test margins frequency disribution, left, and cumulative distribution, right, for **500** rounds of boosting

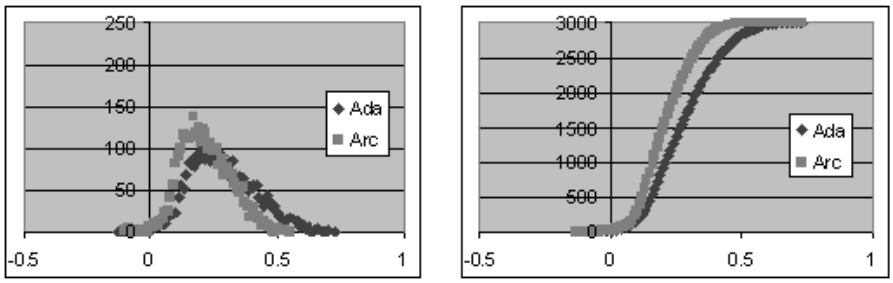

Figure 18: Test margins frequency disribution, left, and cumulative distribution, right, for **1000** rounds of boosting

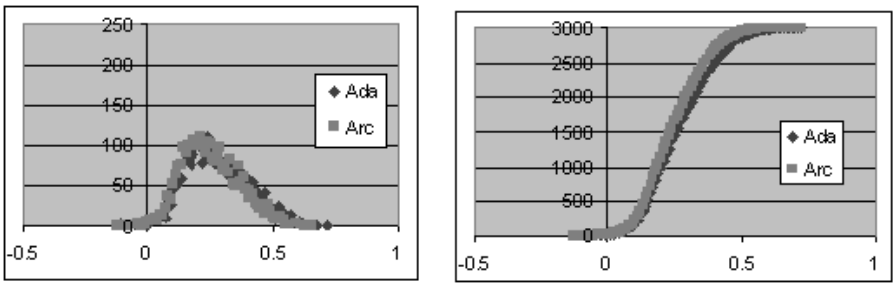

Figure 19: Test margins frequency disribution, left, and cumulative distribution, right, for **5000** rounds of boosting

These results mirrored the results of the experiments looking at training margins, only with a couple differences. The margins distributions again converged. However, the margins for test data are smaller because this was not

the data trained on, and hence there are negative margins as well since some classifications of test data would obviously be incorrect.

Furthermore, while the frequency distributions for training margins were missing their left tail, these looked more like a gaussian curve. This held for other datasets as well.

5.6 Margins of Coordinate Assent

Again, some representative data plots are used for the coordinate assent algorithm, labeled as "Smooth" because it uses the smooth margin function.

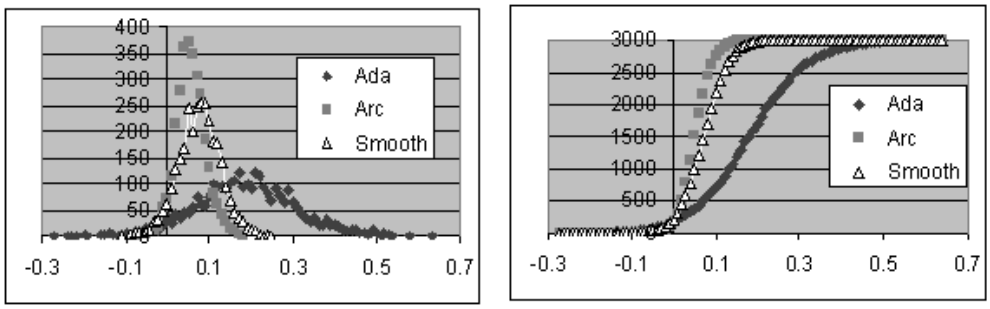

Figure 20: Test margins frequency disribution, left, and cumulative distribution, right, for **100** rounds of boosting on **OCR 49**

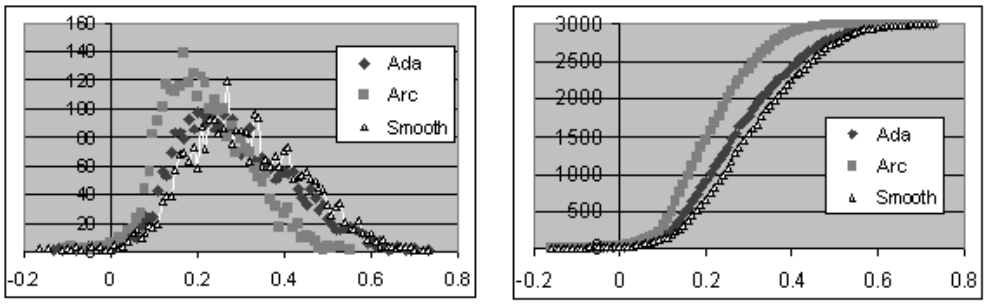

Figure 21: Test margins frequency disribution, left, and cumulative distribution, right, for **1000** rounds of boosting on **OCR 17**

While it would have been nice to observe difinitive rules for Coordinate Assent, unfortunately this algorithm seemed to behave less predictably than the other two. In the dataplots, one can see that the Coordinate Assent algorithm sometimes acted like ArcGv and sometimes like AdaBoost in its margins distributions. In fact, for OCR 17 data, its cumulative margins distribution was to

the right of AdaBoost's, something we have not seen for ArcGv. However, our studies of Coordinate Assent margins were limited, so we could not make many conclusions based on our experiments about the behavior of this algorithm. One thing no note is that its margins distribution also seemed to converge with AdaBoost and ArcGv's margins distributions as the number of boosting rounds increased.

6 Discussion

This research has shown that looking at margins distributions to analyze the behavior of boosting algorithms can yield interesting results that are sometimes not intuitive. We have also reason to believe that this method is promising in seeing the correlation between the margins distribution and the test error of an algorithm.

We have tested select boosting algorithms under different conditions, including variations in the training set size, number of boosting rounds, and datasets. We also looked at training and test margins distributions. We came upon some results, including the convergence of the distributions with an increase in boosting rounds, the equivalence of ArcGv and ArcGvMax, an unexpected case where AbaBoost overfit quickly, and most importantly, a contradiction to the claim that margins distributions are higher for ArcGv.

While we have been able to cast doubt on Breiman's claim [1], we also see there is more work to be done in this area. Mainly, this experiment should be repeated using CART Decision Trees [2], as described in the approach section. Further, these experiments should also be repeated on the same datasets that were used in Breiman's experiments to evaluate his claim that he found an inverse correlation between algorithm performance and its margins distributions. It is also probably worthwhile to explore the differences in the margins distributions between the seperable and non-separable cases since our results suggest such an approach may be fruitful.

Finally, the results in this paper should be replicated and tested with more datasets and different implementations. Any intuitive explanations we may come up with to explain the behavior of these algorithms may also lead to new ideas about how to describe their behavior more rigorously.

In doing this research, we were able to learn about the difficulty of approaching such a problem and choosing which features were relevant to look at. Given the degrees of freedom in various parameters, there are countless experiments that can be performed, and it is always hard to evaluate which would be the most promising. In this case, any results we would come up with became interesting because they shed light on this problem by looking at a case not analyzed before, but it is not always true in research. It was also interesting to see how much relevant information there existed and how much previous work had been done in this area and how difficult and rewarding it is to work on research problems.

7 Conclusion

We have seen that margins play a role in the analysis of boosting algorithms' behavior, yet we also know that doing such an approach can leave us with many unanswered questions about why an algorithm performs as well as it does. We have seen that looking at margins distributions can lead us to both make claims and cast doubts about how AdaBoost and ArcGv perform. It is also clear, however, that an experimental approach to this problem, while vital, will not solve all open questions about these algorithms and that more mathematical analysis of these results is also needed.

Acknowledgements

I would like to thank my advisor for this project, Robert Schapire for not only suggesting this topic, but also for his extremely helpful advice and input into this project. He has generated or informed me of many of the ideas I have used in this paper, including the reformulation of ArcGv and Coordinate Assent, the dynamic programming solution to CART, and an explanation of the overfitting on the DNA dataset, among others. This paper could not have materialized without his help.

References

[1] Leo Breiman. Prediction games and arcing algorithms. *Neural Computation*, 11(7):1493–1517, 1999.

[2] Leo Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Wadsworth, 1984.

[3] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001.

[4] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, 1997.

[5] W. Jiang. Does boosting overfit: Views from an exact solution. Technical Report 00-04, Department of Statistics, Northwestern University, September 2000.

[6] Xiao-Bai Li, James Sweigart, James Teng, Joan Donohue, and Lori Thombs. A dynamic programming based pruning method for decision trees. *INFORMS J. on Computing*, 13(4):332–344, 2001.

[7] Ron Meir and Gunnar Ratsch. An introduction to boosting and leveraging. pages 118–183, 2003.

[8] Cynthia Rudin, Robert E. Schapire, and Ingrid Daubechies. Boosting based on a smooth margin. In *COLT*, pages 502–517, 2004.

[9] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.

[10] Robert Schapire. Princeton class notes for computer science 402. machine learning, 2004.

[11] Robert E. Schapire. The boosting approach to machine learning: An overview, December 19 2001.

[12] Robert E. Schapire, Yoav Freund, Peter Barlett, and Wee Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. In *ICML '97: Proceedings of the Fourteenth International Conference on Machine Learning*, pages 322–330. Morgan Kaufmann Publishers Inc., 1997.