Journal of Machine Learning Research 8 (2007) 409-439 Submitted 7/04; Revised 8/06; Published 3/07

Boosted Classification Trees and Class Probability/Quantile Estimation

David Mease MEASE_D@COB.SJSU.EDU
Department of Marketing and Decision Sciences
San Jose State University
San Jose, CA 95192-0069, USA

Abraham J. Wyner AJW@WHARTON.UPENN.EDU
Andreas Buja BUJA@WHARTON.UPENN.EDU
Department of Statistics
Wharton School, University of Pennsylvania
Philadelphia, PA 19104-6340, USA

Editor: Robert Schapire

Abstract

The standard by which binary classifiers are usually judged, misclassification error, assumes equal costs of misclassifying the two classes or, equivalently, classifying at the 1/2 quantile of the conditional class probability function $P[y = 1|x]$. Boosted classification trees are known to perform quite well for such problems. In this article we consider the use of standard, off-the-shelf boosting for two more general problems: 1) classification with unequal costs or, equivalently, classification at quantiles other than 1/2, and 2) estimation of the conditional class probability function $P[y = 1|x]$. We first examine whether the latter problem, estimation of $P[y = 1|x]$, can be solved with LogitBoost, and with AdaBoost when combined with a natural link function. The answer is negative: both approaches are often ineffective because they overfit $P[y = 1|x]$ even though they perform well as classifiers. A major negative point of the present article is the disconnect between class probability estimation and classification.

Next we consider the practice of over/under-sampling of the two classes. We present an algorithm that uses AdaBoost in conjunction with **O**ver/**U**nder-**S**ampling and **J**ittering of the data ("JOUS-Boost"). This algorithm is simple, yet successful, and it preserves the advantage of relative protection against overfitting, but for arbitrary misclassification costs and, equivalently, arbitrary quantile boundaries. We then use collections of classifiers obtained from a grid of quantiles to form estimators of class probabilities. The estimates of the class probabilities compare favorably to those obtained by a variety of methods across both simulated and real data sets.

Keywords: boosting algorithms, LogitBoost, AdaBoost, class probability estimation, over-sampling, under-sampling, stratification, data jittering

1. Introduction

Misclassification error is the standard by which binary classifiers are usually judged. This implicitly assumes equal cost of misclassifying the two classes or, equivalently, classifying at the 1/2 quantile of the *conditional class probability function*, or **CCPF** for short. By this standard, AdaBoost and other boosting algorithms have demonstrated some of the lowest misclassification errors across a wide range of applications. Furthermore, boosting algorithms are successful at minimizing mis-

©2007 David Mease, Abraham J. Wyner and Andreas Buja.

classification error in ways that are remarkably insensitive to the algorithm's number of iterations. However, for many problems, this is insufficient. Since misclassification error implicitly assumes equal costs for each of the two classes, it is not always an appropriate standard. One is often interested in classification with unequal costs, as in medical problems where a false negative is often much more serious than a false positive. When costs are unequal a desirable classifier selects, for a given feature vector x, the class label that minimizes the expected cost. Alternatively, classification problems may be formulated not in terms of the differential costs of misclassification, but rather in terms of a classifier's ability to predict a quantile threshold, as in marketing when households with an estimated take-rate higher than, say, 0.80 are slated for a campaign. In short, the problem of binary classification with unequal costs is equivalent to the problem of thresholding conditional class probabilities at arbitrary quantiles. As a special case, one obtains the equivalence of an equal-cost classifier and a median classifier that thresholds the conditional probability of both classes at 1/2.

A related problem is that of mapping a classifier to a base rate for the two classes other than that for which the classifier was trained. For example, the training sample may have contained 10% positives, but one is interested in a classifier that performs well when there are 50% positives. A simple calculation shows that a change of base rate is equivalent to a change in the cost ratio and also to a change of the quantile on the class probabilities. These connections are lucidly explained by Elkan (2001), and we will recount them in Section 3.1 in a form suitable for our purposes.

The problem of classifying with arbitrary cost ratios and/or imbalanced base rates has been addressed by a sizable literature. The problem of imbalance has been of particular interest in areas such as text classification (Liu et al., 2002) and bankruptcy prediction (Foster and Stine, 2004) where there often exist very few positives but a vast supply of negatives. One will then try to correct for this imbalance. With respect to boosting, the literature dealing with unequal costs of misclassification is small relative to the huge literature on boosting with respect to (equal) misclassification error. Modifications to boosting and similar algorithms that have been suggested to handle this more general problem include Slipper (Cohen and Singer, 1999), AdaCost (Fan et al., 1999), CSB1 and CSB2 (Ting, 2000) and RareBoost (Joshi et al., 2001). These algorithms have demonstrated some success over straightforward AdaBoost, but no single such algorithm stands out as the favorite.

1.1 Over/Under-Sampling

One general class of approaches (popular in the Computer Science community) for calibrating any equal-loss classifier to handle the equivalent problems of unequal costs, arbitrary quantile thresholds, and imbalanced base rates is the idea of over- and under-sampling (Chan and Stolfo, 1998; Elkan, 2001; Estabrooks et al., 2004). In these schemes one typically over-samples the minority class by sampling with replacement, and/or one under-samples the majority class by sampling without replacement. Sampling with replacement is necessary when the class size is increased, whereas sampling without replacement seems more natural when the class size is decreased. Note that sampling with replacement is bound to produce ties in the sample, especially as the sampling rate increases. An interesting variation of the idea is the Synthetic Minority Over-Sampling TEchnique ("SMOTE") by Chawla et al. (2002, 2003) which avoids ties in the over-sampled minority class by moving the sampled predictor points toward neighbors in the minority class.

The necessity to break ties is an issue that is addressed in detail in Section 3 and also summarized here. Ties can be a problem because classification methods may be driven more by the set of distinct data points than the multiplicities of the tied data points. Changing multiplicities, which is what sampling with replacement does, will therefore have a smaller than expected effect. Thus changing multiplicities should be accompanied by breaking the ties, as in SMOTE. There is evidence that tie-breaking does not need to be sophisticated; random perturbations or "jittering" of the data points in predictor space work surprisingly well. The effectiveness of this technique for AdaBoost is demonstrated in this article.

1.2 Direct Estimation of the CCPF

A second general solution to the problem of classification at arbitrary quantile thresholds is by direct estimation of conditional class probability functions. This is the traditional approach in statistics, as exemplified by logistic regression. Given an estimate of the CCPF one can trivially achieve classification at any arbitrary quantile by thresholding the estimated CCPF at that quantile.

If estimation of a CCPF solves the problem of classification for arbitrary costs, quantiles and imbalances, it is natural to wonder why one bothers to solve the problem by any other means. Why would there still be an interest in, for example, classifiers trained on over/under-sampled training data? The reason for this has to do with the fact that finding a good classifier is a robust endeavor, whereas estimation of a CCPF is a subtle business. The latter has to simultaneously contain the information for good classification at *all* quantiles. By comparison, a classifier is usually described by thresholding some score function at zero, but this function is quite arbitrary except for the requirement to be on the proper side of zero often enough to produce few errors on test samples. This robustness of the classification problem (observed in the literature) is often, but not always, due to the crude nature of the criterion, namely, misclassification error.

Friedman et al. (2000) show that a stagewise approach to the minimization of an exponential loss function by a linear combination of classification trees is algorithmically similar to AdaBoost. In fact, these same authors (in Hastie et al., 2001, Section 10.2) attribute boosting's success to this similarity. This stagewise logistic regression view immediately led to two major developments. First, by connecting AdaBoost to logistic regression, Friedman et al. (2000) and Hastie et al. (2001) implied that one can apply a link function to the scores produced by AdaBoost to provide estimates of the CCPF. Second, by minimizing different loss functions, these authors led the way to the development of new boosting algorithms. In particular, in Friedman et al. (2000) we first encounter *LogitBoost*, an algorithm which computes the CCPF directly by a stagewise minimization of *log* loss on the training data. For the purposes of this paper we will focus on Logitboost; however, it should be noted that other researchers have also developed modifications to AdaBoost for minimization of log loss. One such algorithm is given by Duffy and Helmbold (1999). A similar algorithm is given by Collins et al. (2000) which is motivated by viewing boosting as an optimization of Bregman divergences.

We will give evidence that neither LogitBoost nor the application of the prescribed link to AdaBoost are effective at estimating the CCPF. We will show that when AdaBoost has been run for the large number of iterations required to produce a successful classifier (with respect to misclassification error), its scores have typically diverged to such a degree that putting them through a link function produces values for the CCPF that cluster near 0 and 1. Hence, they are quite erratic as CCPF estimates. The prevailing solution to this problem is regularization of the CCPF estimator. Such regularization methods include the imposition of limits on the class of base learners (i.e., decision stumps instead of trees), the refinement of the learning rate, or simply early stopping. This is not the approach we choose here, though. Instead, our aim is to use off-the-shelf boosting and explore how its manifest success in classification can be carried over to CCPF estimation.

In order to illustrate boosting's inability to estimate CCPF's, we present simulations for which AdaBoost is able to successfully lower the misclassification error for classification trees, but is unable to find the CCPF since its estimates get worse with each iteration of the algorithm. We conclude that AdaBoost is successful at estimating classification *boundaries* even though its scores do not estimate conditional class probabilities. Further, we show that *the same is true of LogitBoost*, despite the fact that estimation of conditional class probabilities is the motivation behind this algorithm. In short, both of these approaches are often ineffective for CCPF estimation.

Because of the relative difficulty of CCPF estimation compared to classification, it will always remain of interest to approach the latter problem directly. In fact, we take this argument as a license for travelling the opposite direction: we construct estimators of CCPF's from collections of classifiers computed from grids of quantiles. The precision of such estimators depends on the denseness of the quantile grids, but in view of the natural sampling variability of CCPF estimators, the granularity of the grid may need only compare with the noise level. Furthermore, if the primary use of the CCPF estimator is thresholding for classification, then the proposed estimator will be satisfactory for practical purposes if the grid contains all quantiles of actual interest. We show experimentally that over- and under-sampled estimates of the CCPF using AdaBoost, constructed in this way, compare favorably with competing methods.

The evidence presented in this paper may seem to be at odds with the logistic regression view of boosting. Certainly, from a practical point of view, this is true to some extent. Our experiments suggest that boosted classification trees produce excellent estimates of class membership and are surprisingly resistant to overfitting, while the same is not true of the values that the logistic regression view of Friedman et al. (2000) would suggest are the estimates of the class probabilities. However, a couple of other factors should be considered here. First, the Friedman et al. (2000) paper is not the only research connecting logistic regression and boosting. For instance, Lebanon and Lafferty (2001) present a very different view of this relationship. Our focus is only on probability estimation using the Friedman et al. (2000) link functions. Secondly, good estimation of the CCPF is not always necessary even for logistic regression itself. Logistic regression is just one of many examples where the underlying model may not represent the true distribution of data well, but can still be quite useful for prediction. Freund and Schapire (2004) discuss this point. They draw a parallel to HMM's for speech analysis and write, "nobody using HMM's for speech analysis really thinks that speech can be synthesized by these HMM's."

In the next section we present the argument that boosted classification trees combined with the Friedman et al. (2000) link functions do not estimate conditional class probabilities. The boosting algorithms we consider are standard, off-the-shelf techniques. It may be argued that some variations of boosting employing extensive regularization (for example, Zhang and Yu 2005, Rosset et al. 2004 and Blanchard et al. 2003) or direct cost-optimization techniques can overcome this problem. We do not consider these techniques in this paper for a number of reasons. A practical reason is that these techniques are often considered not worth the trouble by many practitioners. This is especially true when a simple effective alternative exists, as we will demonstrate. Further, the focus of the paper is on how the off-the-shelf utility of boosting algorithms which greatly contributes to their importance can be carried over to CCPF estimation in a straightforward manner. Regularization and cost-optimization boosting algorithms are outside the scope of this paper.

2. Boosted Classification Trees Do Not Estimate Conditional Class Probabilities

We now demonstrate, by means of simulations, how boosting with classification trees fails to estimate the CCPF while at the same time performing well in terms of the classification criterion, namely, misclassification error.

We introduce customary notation. We are given training data $x_1, ..., x_n$ and $y_1, ..., y_n$ where each x_i is a $d-$dimensional vector of predictors and $y_i \in \{-1, +1\}$ is the associated observed class label. To justify generalization, it is usually assumed that training data as well as any test data are *iid* samples from some population of (x, y) pairs.

We will consider the two boosting algorithms *LogitBoost* (Friedman et al., 2000) and (discrete) *AdaBoost*. AdaBoost (Freund and Schapire, 1996) is the most common implementation of boosting. The algorithm is as follows. First let $F_0(x_i) = 0$ for all x_i and initialize weights $w_i = 1/n$ for $i = 1, ..., n$. Then repeat the following for m from 1 to M:

- Fit the classifier g_m to the training data using weights w_i where g_m maps each x_i to -1 or 1.
- Compute the weighted error rate $\varepsilon_m \equiv \sum_{i=1}^{n} w_i \mathrm{I}\{y_i \neq g_m(x_i)\}$ and half its log-odds, $\alpha_m \equiv \frac{1}{2} \log \frac{1-\varepsilon_m}{\varepsilon_m}$.
- Let $F_m = F_{m-1} + \alpha_m g_m$.
- Replace the weights w_i with $w_i \equiv w_i e^{-\alpha_m g_m(x_i) y_i}$ and then renormalize by replacing each w_i by $w_i / (\sum w_i)$.

The final classifier is 1 if $F_M > 0$ and -1 otherwise. It has been observed repeatedly that the performance of the procedure, with respect to misclassification error, is quite insensitive to the choice of M and tends to result in small error rates (relative to competing methods) across a wide range of applications, especially in high dimensions. In many real examples, large values of M work very well.

Motivated by analogies between AdaBoost and additive logistic regression, Friedman et al. (2000) proposed a connection between the score function $F_m(x)$ and a logistic regression model. They suggest that an estimate $p_m(x)$ of the CCPF $p(x)$ can be obtained from F_m through an (essentially) logistic link function:

$$p_m(x) = p_m(y = 1|x) = \frac{1}{1 + e^{-2F_m(x)}}. \tag{1}$$

In fact, using this link function it can be shown that the "exponential loss" of AdaBoost can be mapped to a scoring rule on the probabilities similar to a maximum likelihood criterion (Buja et al., 2006). From this perspective, each iteration of boosting uses the current estimates of the probabilities to minimize this criterion in a stagewise manner.

The view of boosting as a form of logistic regression and thus as an estimator of the CCPF has given rise to much research since its publication. It has been the basis for recent work on choosing stopping rules so as to not overfit probabilities (Dettling and Buhlmann, 2003) as well as efforts to establish consistency using regularization (Bickel et al., 2006; Jiang, 2004; Lugosi and Vayatis, 2004). Furthermore, in their seminal article, Friedman et al. (2000) introduced LogitBoost, which directly minimizes the negative Bernoulli log-likelihood in a stagewise manner, instead of the exponential loss of AdaBoost. This is the prevailing reason why LogitBoost is widely thought to be a better estimator of the CCPF. LogitBoost is the second algorithm we will examine in our simulations.

The statistical view that casts the outputs from boosting classification trees as estimates of the CCPF poses no problem for the purpose of median classification. The symmetry of the chosen link functions implies that thresholding p_m at 1/2 amounts to thresholding F_m at 0. Used in this way, boosting provides excellent classification rules that are quite insensitive to the number of iterations M. However, as we will see in the following simulations, the idea that boosting classification trees can be used to estimate conditional probabilities is questionable. While in most instances there is little to no overfitting with respect to misclassification error (i.e., median estimation), with respect to probability estimation overfitting is likely and often very pronounced. This sheds doubt on the idea that the success of boosting is due to its similarity to logistic regression, and research motivated by this connection, while insightful in its own right, is most likely mistaken if it claims to throw light on boosting.

In all our experiments the base learners will be 8-node trees, as in Friedman et al. (2000), unless otherwise noted. The two exceptions are one simulated data set for which we will consider stumps in addition to 8-node trees for sake of comparison, and one large data set for which we use 2^{10}-node trees. The AdaBoost algorithm we use is described above, and the classification trees are constructed using the function “Rpart” in the R software package (http://www.r-project.org/). The probabilities are obtained by mapping the resulting score $F_m(x)$ through the link (1) to produce the CCPF estimate $p_m(y=1|x)$. We will refer to this procedure as *P-AdaBoost*. We will use Dettling and Buhlmann’s (2003) implementation of LogitBoost with two minor modifications to the source code to allow the function Rpart to fit 8-node decision trees instead of stumps.

2.1 A 2-dimensional Circle Model

For our first simulation, we consider the domain of X to be the square $[0,50]^2$ and construct level curves of $p(x)$ to be concentric circles with center $(25,25)$. We let

$$p(x) = p(y=1|x) = \begin{cases} 1 & r(x) < 8 \\ \frac{28-r(x)}{20} & 8 \le r(x) \le 28 \\ 0 & r(x) > 28 \end{cases}$$

where $r(x)$ is the distance from x to the point $(25,25)$ in R^2. We will call this the *2-Dimensional Circle* model.

The right panel of Figure 1 shows these probabilities for a hold-out sample that is a grid of 2500 evenly spaced points on $[0,50]^2$. The points are color-coded such that the white region corresponds to the subregion of the square where $p(x) \le 0.1$. The pink subregion is where $0.1 < p(x) \le 0.5$, the red subregion is where $0.5 < p(x) \le 0.9$ and the black subregion is where $p(x) > 0.9$. For the training data, we simulated $n = 1000$ *iid* observations with x uniform on $[0,50]^2$ and the class label y chosen according to $p(y=1|x)$ above. The training data is shown in the left panel of Figure 1 colored such that points are green plus signs when $y=-1$ and black plus signs when $y=1$.

If the algorithms are stopped early enough, both P-AdaBoost and LogitBoost provide reasonably accurate estimates of the CCPF. For example, we present in Figure 2 the resulting estimates of $p_m(x)$ at $m=5$ iterations. We use the same color coding as Figure 1. It is clear that both P-AdaBoost (left panel of Figure 2) and LogitBoost (right panel of Figure 2) estimate conditional probabilities that match the true probabilities (the right panel of Figure 1) fairly well. However, as the number of iterations increases, both begin to overfit the probabilities. As m increases the resulting conditional probabilities converge to zero and one for all x, and consequently the estimates of all quantiles

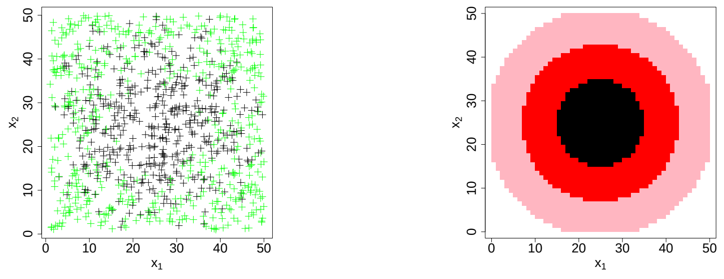

Figure 1: Training data (left panel) and true probabilities (right panel) for the 2-Dimensional Circle model with $n = 1000$.

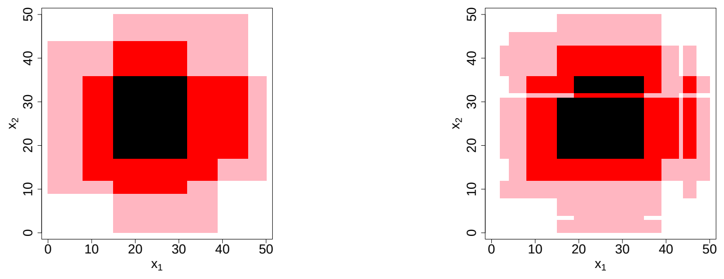

Figure 2: Estimated probabilities for P-AdaBoost (left panel) and LogitBoost (right panel) at 5 iterations.

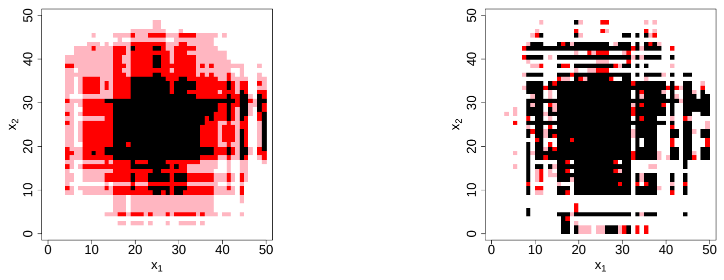

Figure 3: Estimated probabilities for P-AdaBoost (left panel) and LogitBoost (right panel) at 800 iterations.

converge to the estimate for the median. Figure 3 reveals the conditional probability estimates for the hold-out sample at $m = 800$ iterations. At $m = 800$, the P-AdaBoost algorithm exhibits some overfitting, as the white and black regions are larger than they should be. LogitBoost in contrast, has overfit the data more severely at $m = 800$. For almost all x in the hold-out sample, its estimate of the CCPF is either greater than 0.9 (black) or less than 0.1 (white). In fact, as m increases further both P-AdaBoost and LogitBoost eventually produce estimates of the CCPF in which almost all probability estimates are near zero or one. Thus, these algorithms eventually provide a complete overfit of the probabilities. In contrast, the estimate of the median remains relative stable as m increases and is quite accurate.

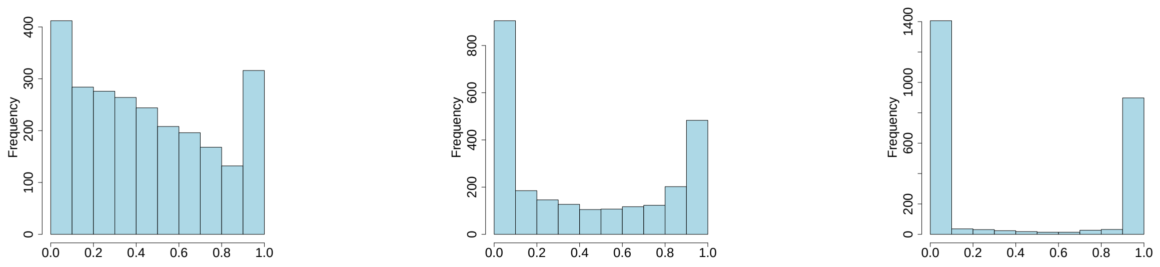

Figure 4: Histograms of true probabilities (first plot) and estimated probabilities for P-AdaBoost (second plot) and LogitBoost (third plot) at 800 iterations.

Overfitting the data, which is reflected in the divergence of the probability estimates to zero and one by both P-AdaBoost and LogitBoost, is particularly pronounced in this example since many of the true probabilities are actually quite far from zero and one. This is shown in Figure 4. The left most panel is a histogram of the true CCPF over the 2500 randomly chosen points x that form the hold-out sample. The middle and right most panels are the histograms of the estimated CCPF for P-AdaBoost and LogitBoost respectively, again at 800 iterations, on the hold-out sample. It is clear, in the case of LogitBoost, that for almost every x the estimated CCPF is greater than 0.9 or less than 0.1. In contrast, the true CCPF is much more spread out. The histogram of the estimated probabilities for P-AdaBoost exhibits the same extremal clumping, although not as severely as LogitBoost.

2.2 A 10-dimensional Model

Next we borrow a model first introduced in Friedman et al. (2000) that reveals the same pattern of overfitting and the same divergence of the CCPF. This example demonstrates how overfitting is possible in cases where the Bayes error rate is near or even equal to zero.

For the simulation, we sample n observations of the 10-dimensional vector x generated *iid* from $N^{10}(0,I)$ (we call this the *10-Norm* model). The CCPF is again the conditional probability that $y=1$ given x (which we denote $p(x)$) and is defined as follows:

$$\log\left(\frac{p(x)}{1-p(x)}\right) = \gamma[1-x^{(1)}+x^{(2)}-x^{(3)}+x^{(4)}-x^{(5)}+x^{(6)}]\sum_{j=1}^{6}x^{(j)} \tag{2}$$

where the superscripts denote the components of the vector. As γ increases the Bayes error rate decreases. We begin with $\gamma=10$, which mimics the simulation in Friedman et al. (2000) exactly. With $\gamma=10$ the Bayes error rate is only 3%. When $\gamma=0.5$ the noise level in the model is higher and the Bayes error rate climbs to 22%.

Since these simulations are in ten dimensions instead of two, it is difficult to display the overfitting of the probabilities graphically as in the previous example. Instead we consider some quantitative measures for the accuracy of probability estimates. There is no single, established method for measuring the performance of class probability estimators (see Zadrozny and Elkan (2001) for a discussion). When simulating from models with known parameters, there are many choices. We choose the following loss functions which all provide a penalty depending on the extent to which the estimate of the probability $\hat{p}$ diverges from the true probability $p = p(x)$ on a hold-out sample $x_1^*, \ldots, x_{n^*}^*$.

- Squared Loss:

$$\sum_{i=1}^{n^*} [p(x_i^*) - \hat{p}(x_i^*)]^2 \tag{3}$$

- Log Loss:

$$-\sum_{i=1}^{n^*} [p(x_i^*) \log(\hat{p}(x_i^*)) - (1 - p(x_i^*)) \log(1 - \hat{p}(x_i^*))] \tag{4}$$

- Exponential Loss:

$$\sum_{i=1}^{n^*} \left[p(x_i^*) \sqrt{\frac{1 - \hat{p}(x_i^*)}{\hat{p}(x_i^*)}} + (1 - p(x_i^*)) \sqrt{\frac{\hat{p}(x_i^*)}{1 - \hat{p}(x_i^*)}} \right] \tag{5}$$

This last scoring function is the population analogue of the exponential loss of AdaBoost when composed with the link (1) suggested by Friedman et al. (2000) as pointed out by Buja et al. (2006). The log loss is the population version of the negative log likelihood of the Bernoulli model. For the simulations we choose a hold-out sample with $n^* = 2500$ drawn from the same distribution as the training data. Note, however, that since both log loss and exponential loss are unbounded, we threshold $\hat{p}$ so that it always lies between 0.05 and 0.95 before computing these two loss functions. This prevents a few conditional probability estimates that are close to zero or one from dominating the loss. This truncation will become more important in Section 3.

The four plots in Figure 5 display the values of misclassification error, squared loss, log loss and exponential loss averaged over the hold-out sample for the 10 dimensional normal (10-Norm) model given by Equation (2) with $n = 500$ (and $\gamma = 10$). The large black dot at zero iterations is the loss occurred by estimating all probabilities by the marginal sample proportion in the training data or analogously by the majority class in the training data for misclassification error. P-AdaBoost is represented by the black line and LogitBoost by the blue line. The red and green lines represent the performance of JOUS-Boost, that is, AdaBoost combined with over/under-sampling and jittering, as will be described in Sections 3 and 4. The graph for misclassification error depicts the trademark features of boosting algorithms. There seems to be little or no overfitting and the error curve reflects a general decreasing trend as more and more iterations are performed. The classifiers continue to improve their performance on hold-out samples even as the number of iterations increase beyond the number required to fit the training data perfectly (that is, to achieve zero misclassification error on the training data). Friedman et al. (2000) also observed this trend; however, they did not examine any measures of performance with respect to the probability estimation. From the other three plots that graph the performance of the CCPF estimator, we can see that all three of the measures display a generally increasing pattern, implying that both of the boosting algorithms overfit the probabilities, even though they do not overfit with respect to out-of-sample misclassification error. Figure 6 shows that this pattern becomes even more pronounced when the value of γ is decreased from 10 to 0.5, which increases the Bayes error rate from 0.03 to 0.22.

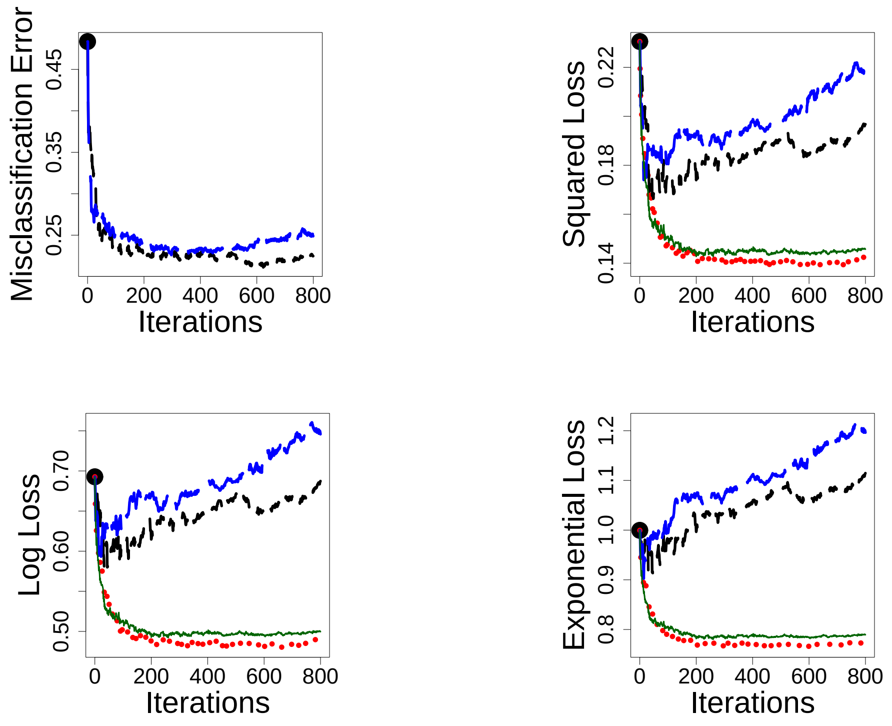

Figure 5: Misclassification error, squared loss, log loss and exponential loss for the 10-Norm model with $n = 500$ and $\gamma = 10$. **Black (Short Dashes)**: P-AdaBoost. **Blue (Long Dashes)**: LogitBoost. **Red (Dotted)**: JOUS-Boost, over-sampled. **Green (Solid)**: JOUS-Boost, under-sampled.

The observed divergence of the probability estimates to extreme values is a consequence of the fact that the span of the space of weak learners (8-node trees in this case) is rich enough to separate the two classes completely. This is always the case when the algorithms use all possible classification trees of a certain depth defined on the training data. As the training set grows, the space of weak learners grows (since they are data defined), allowing every point in the training set to be correctly classified. This effect holds even if the trees are restricted to a depth of 2 (so-called stumps), although the number of iterations required is typically a lot larger. More discussion of separation with boosting and a proof that boosted stumps will eventually separate all the data (for the case of a single predictor) can be found in Jiang (2000). The only condition needed for the separability is that at least one of the predictors assumes unique values for all points. For a continuous predictor this will occur with probably one, but for the case of all categorical predictors separation may not be possible.

As a consequence of this separability, as the number of boosting iterations increases, the scores $F(x_i)$ for which $y_i = +1$ tend towards $+\infty$, and scores for which $y_i = -1$ tend toward $-\infty$. This occurs because the exponential loss (or negative log likelihood in the case of LogitBoost) is minimized at these limiting values. On hold-out samples we observe that when $y = +1$, boosting usually returns a very large value for F, thus resulting in a correct classification when the threshold for F

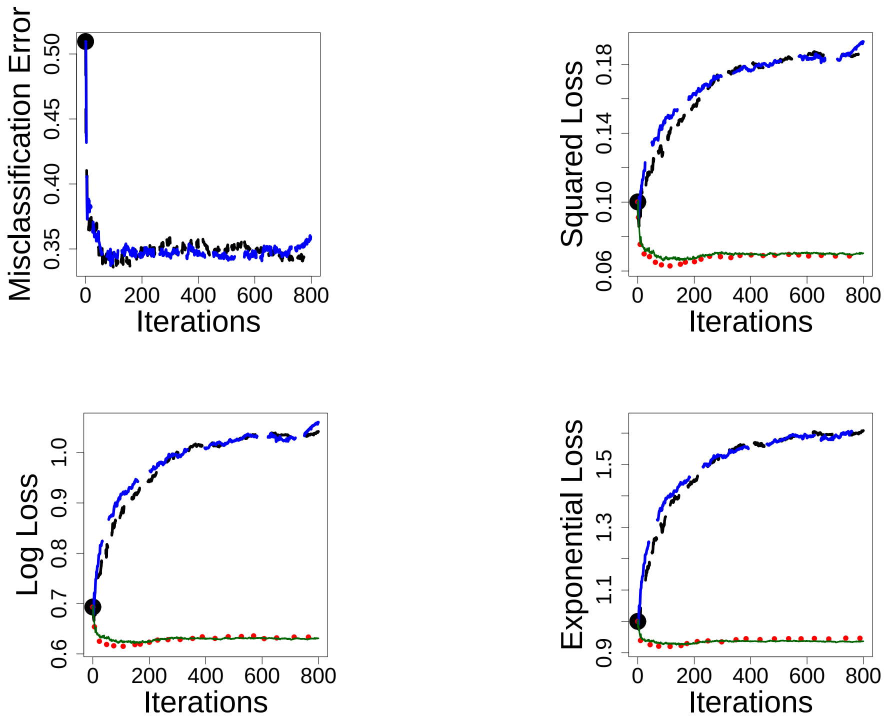

Figure 6: Misclassification error, squared loss, log loss and exponential loss for 10-norm model with $n = 500$ and $\gamma = 0.5$. **Black (Short Dashes)**: P-AdaBoost. **Blue (Long Dashes)**: LogitBoost. **Red (Dotted)**: JOUS-Boost, over-sampled. **Green (Solid)**: JOUS-Boost, under-sampled.

is zero. The analogue is true for observations for which $y = -1$. The effectiveness of boosting for classification problems is not in spite of this phenomenon, but rather due to it. The fact that the absolute values of F_m grow without bound as m increases does not necessarily lead to an over-fit with regard to misclassification error because the absolute value is irrelevant to a median-classifier; only the sign matters.

If boosting is to be used as a probability estimator or a quantile classifier by thresholding at a value other than zero (as prescribed by a link such as that of Equation (1)), then the absolute value of F_m *does* matter. Boosting will eventually estimate $p(x)$ by either 0 or 1. The specific form of the monotonic link function is irrelevant, so long as it ascends from zero to one, because the absolute value of F_m is largely a function of m rather than a reflection on the true class probability. We illustrate this graphically by tracing the values of F_m for AdaBoost in our last simulation for the points in the hold-out sample for which the true CCPF is between 0.7 and 0.8. Using the link function in Equation (1) we can compute that the fitted values of F_m for these points should ideally lie between 0.42 and 0.69. However, in Figure 7 we see that the median of F_m for the hold-out points with true p between 0.7 and 0.8 has a general linear trend, and by $m = 800$ iterations is greater than 6, corresponding to a $p(x)$ greater than 0.99999. Without the thresholding at 0.95 and 0.05 the overfitting caused by such extreme probabilities would be even more pronounced than displayed in the plots. This unbounded linear growth of the score function F_m suggests that we can obtain any estimate greater than 1/2 for the CCPF at these points by simply choosing the appropriate stopping time M. The link function is irrelevant.

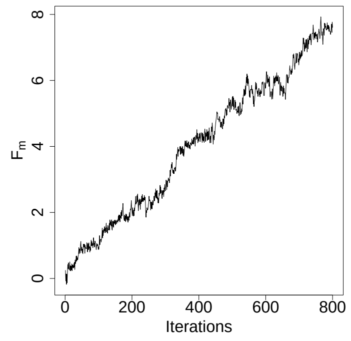

Figure 7: Median$(F_m(x) : 0.7 \leq p(x) \leq 0.8)$

As mentioned earlier, the realization that boosting often overfits the CCPF but not the median value of the CCPF has many important implications. It brings into question whether it is right to attribute the success of boosting to its similarity to logistic regression. Furthermore, the practical work that has arisen from this theoretical view of boosting may be misguided. For instance, stopping rules that are based on cross-validation using a maximum likelihood criterion are questionable. In fact, Dettling and Buhlmann's version of LogitBoost, which was modified for use in the previous simulations, has the option of employing such a stopping rule. Interestingly the authors themselves remark that for gene expression applications, the maximum likelihood criterion indicated that stopping should be implemented for some $M < 100$, but they observe that early stopping does not seem to provide any significant advantages for classification and "most often yields slightly worse results" (Dettling and Buhlmann, 2003).

Before leaving this section we should point out that there do exist cases in which overfitting of the CCPF does not appear to occur (at least not for $m \leq 800$). As discussed above, the probability estimates from boosting will necessarily eventually diverge to zero and one. This is true as long as the space of weak learners is allowed to increase in complexity with the sample size. However, the extent to which this divergence causes overfitting depends on the true probabilities and the size of the training data. In fact, the original simulation by Friedman et al. (2000) used a large data set ($n = 5000$) and $\gamma = 10$, implying a Bayes error rate of 0.03. With these values for the parameters, there does not seem to be substantial overfitting of the probabilities (by $m = 800$). To see this, consider the plots in Figure 8. The success in the problem can be attributed to the large sample size, the moderate number of dimensions (10) and the low Bayes error rate (0.03). Because the true probabilities are practically zero or one, the large number of training points in few dimensions allows for very accurate estimation of the true CCPF values that happen to clump near zero and one.

2.3 Simulations under Greater Uncertainty

It is quite possible that overfitting of the CCPF occurs even with a large sample of training data, namely, when either 1) the Bayes error is high, or 2) the number of dimensions is large, or 3) the decision boundaries are very complex. In the 10-Norm model we can create an example with higher

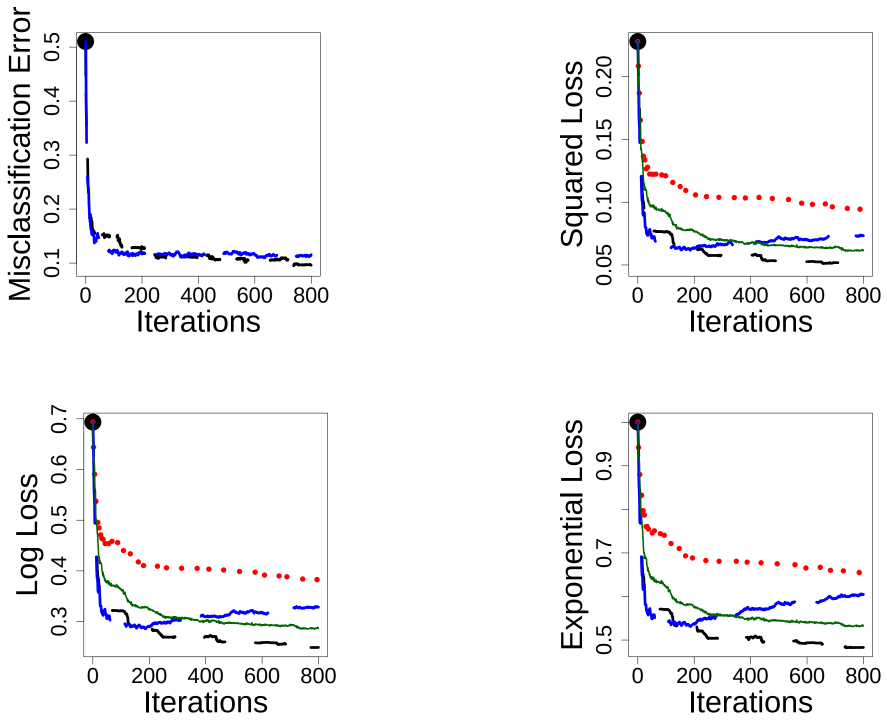

Figure 8: 10-Norm model with $\gamma = 10$ (Bayes error 0.03) and $n = 5000$. **Black (Short Dashes)**: P-AdaBoost. **Blue (Long Dashes)**: LogitBoost. **Red (Dotted)**: JOUS-Boost, over-sampled. **Green (Solid)**: JOUS-Boost, under-sampled.

Bayes error by lowering γ. Figure 9 displays the results for the 10-Norm simulation with $n = 5000$ and $\gamma = 0.5$, corresponding to a Bayes error of 0.22. Here we see that LogitBoost clearly overfits the probabilities, although P-AdaBoost does not, at least not by $m = 800$ iterations.

Figure 10 shows a simulation from the model

$$p(x) = 0.03 + 0.94\,\mathrm{I}\left\{\sum_{j=1}^{d} x^{(j)} > 0\right\} \tag{6}$$

in $d = 1000$ dimensions. We refer to this as the *1000-dim* model. The Bayes error rate is 3%. We chose a sample of size $n = 5000$. Note that both the sample size and the Bayes error rate match those used in original simulation by Friedman et al. (2000) with the 10-Norm model. Our experiments show that both P-AdaBoost and LogitBoost overfit the probabilities from the outset—there is no early stopping point which improves the fit. In contrast, the rate of misclassification drops continuously as the number of iterations increases, and there is no evidence of overfitting. Once again, the behavior we observe conflicts with the logistic regression view of boosting.

In cases such as these, a natural question to ask is whether the overfitting with respect to the probabilities might be avoided by choosing a weaker set of base learners, such as stumps instead of the 8-node trees. In fact, the logistic regression view of boosting suggests stumps are particularly attractive for a model in which the Bayes decision boundary is additive, as is the case for the

1000-dim model considered in the previous paragraph. The reasoning is that since stumps involve only single predictors, the boosted stumps in turn yield an additive model in the predictor space. Discussion of this can be found in Hastie et al. (2001) on pages 323-324 and in Friedman et al. (2000) on pages 360-361. Despite this, it can be observed in Figure 11 that using stumps does not eliminate the overfitting problem for the 1000-dim simulation model in the previous paragraph. Just as we observed in Figure 10 with the 8-node trees, the squared loss, log loss and exponential loss for the stumps all begin increasing with the very first iteration and at no point during the 800 iterations even match the majority vote classifier represented by the solid black dot at iteration zero. The use of stumps instead of 8-node trees in this case slows the overfitting, but by no means produces a useful algorithm for estimation of the CCPF. Further, the first plot in Figure 11 also reveals that the resulting classification rule after 800 iterations is not quite as accurate as with the 8-node trees used in Figure 10.

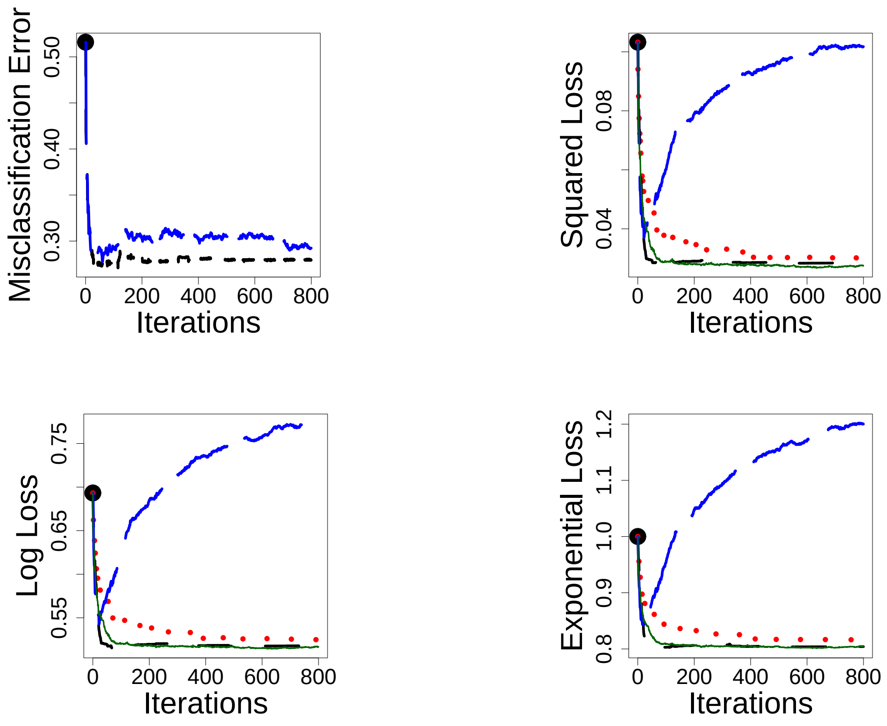

Figure 9: 10-Norm model with $\gamma = 0.5$ (Bayes error 0.22) and $n = 5000$. **Black (Short Dashes)**: P-AdaBoost. **Blue (Long Dashes)**: LogitBoost. **Red (Dotted)**: JOUS-Boost, over-sampled. **Green (Solid)**: JOUS-Boost, under-sampled.

Finally, from the realization that boosting does not provide reliable estimates of quantiles other than the median, we conclude that there is a problem when attempting to use boosting for classification if the misclassification costs of the two classes are unequal, or if the base rate of the training data differs from that of the population of interest as discussed previously. In what follows we address the question of how to use boosting to estimate quantiles other than the median in order to provide solutions to these problems.

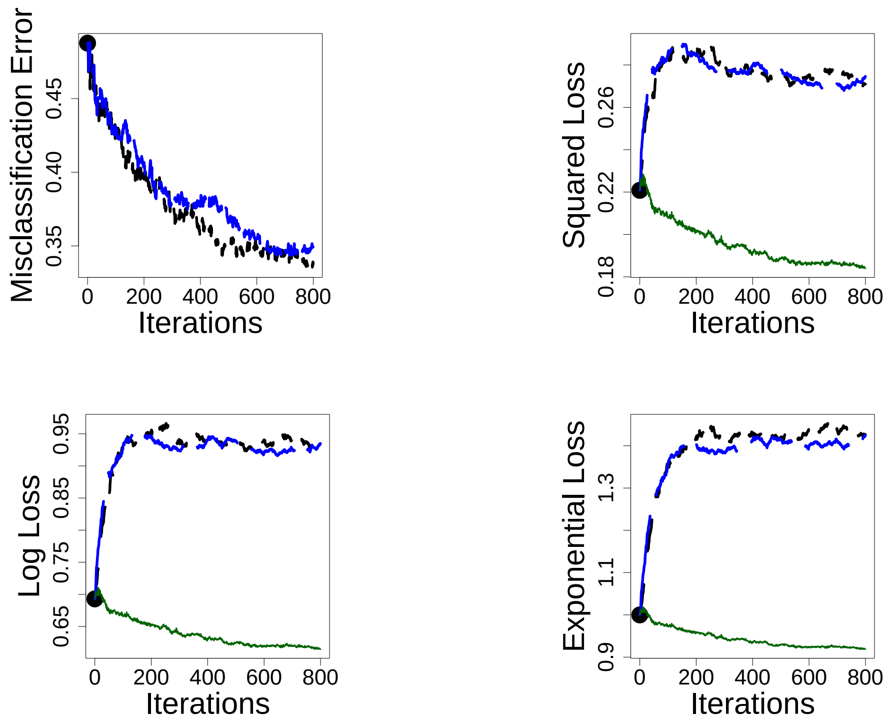

Figure 10: 1000-dim model (Bayes error 0.03) with $n = 5000$. **Black (Short Dashes)**: P-AdaBoost. **Blue (Long Dashes)**: LogitBoost. **Green (Solid)**: JOUS-Boost, under-sampled.

3. Classification With Unequal Costs

Symmetric 0-1 loss is historically motivated by a supposition (previously common in AI, not in statistics) that there always exists a “true” class label, that is, the class probability is either 0 or 1. Hence the goal is always perfect classification, and 0-1 loss is a reasonable measure. In practice, however, it is common to find situations where unequal loss is appropriate and cost-weighted Bayes risk should be the target.

It is an elementary fact that minimizing loss with unequal costs is equivalent to estimating the boundaries of conditional class-1 probabilities at thresholds other than 1/2. We outline the textbook derivation, drawing intuition from the Pima Indians Diabetes Data of the UCI ML Database: Let $p(x) = P[Y = +1|x]$ and $1 - p(x) = P[Y = -1|x]$ be the conditional probabilities of diabetes $(+1)$ and non-diabetes (-1), respectively, and assume the cost of misclassifying a diabetic is $1-c = 0.90$ and the cost of misclassifying a non-diabetic is $c = 0.10$. (The benefit of scaling the costs to sum to 1 will be obvious shortly.) The following consideration is conditional on an arbitrary but fixed predictor vector x, hence we abbreviate $p = p(x)$. The risk (=expected loss) of classifying as a diabetic is $(1 - p)c$ and the risk of classifying as a non-diabetic is $p(1 - c)$. The cost-weighted Bayes rule is then obtained by minimizing risk: Classify as diabetic when $(1 - p)c < p(1 - c)$ or, equivalently, $c < p$, here: $0.1 < p$, which is what we wanted to show. It is a handy convention to

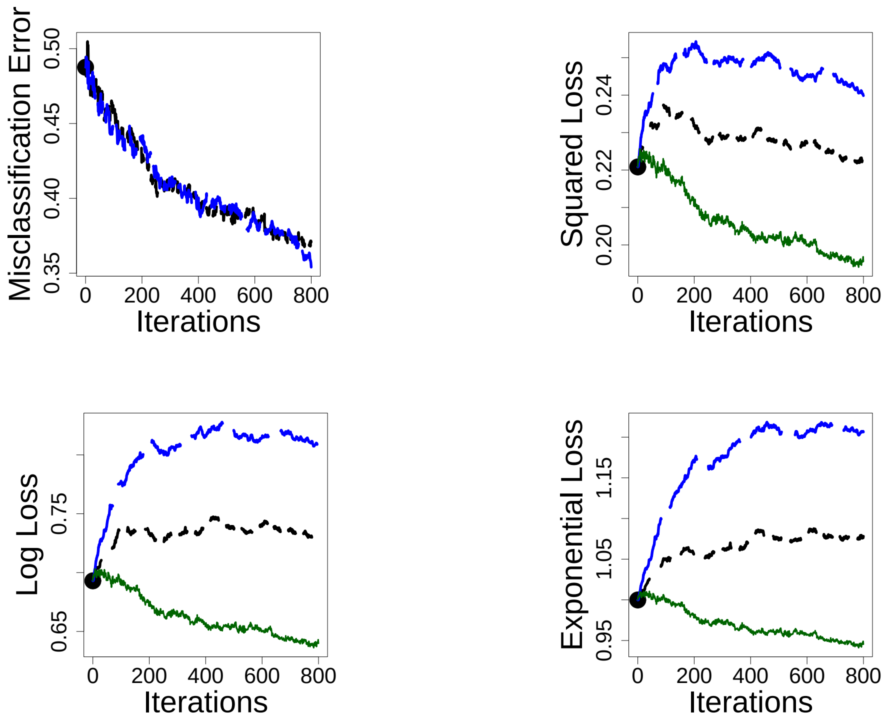

Figure 11: 1000-dim model (Bayes error 0.03) with $n = 5000$ using stumps instead of 8-node trees. **Black (Short Dashes)**: P-AdaBoost. **Blue (Long Dashes)**: LogitBoost. **Green (Solid)**: JOUS-Boost, under-sampled.

scale costs so they add up to 1, and hence the cost c of misclassifying a negative (-1) equals the optimal threshold q on the probability $p(x)$ of a positive $(+1)$.

The previous discussion implies the equivalence between the problem of *classification with unequal costs* and what we call the problem of *quantile classification*, that is, estimation of the region $p(x) > q$ in which observations are classified as $+1$. Most classifiers by default (AdaBoost included) assume equal costs, $c = 1-c = 0.5$, which implies that they are *median-classifiers*: $p(x) > q = 1/2$.

3.1 Over/Under-Sampling

Since boosting algorithms are very good median classifiers, one solution is to leave the boosting algorithm as is and instead tilt the data to force the q-quantile to become the median. Simple over/under-sampling, also called stratification, can convert a median classifier into a q-classifier as follows:

Over/under-sampled Classification:

1. Let N_{+1} and N_{-1} be the marginal counts of labels $+1$ and -1. Choose $k_{+1}, k_{-1} > 0$ for which $\frac{k_{+1}}{k_{-1}} = \frac{N_{+1}}{N_{-1}} / \frac{q}{1-q}$.

2. Select k_{+1} observations from the training set for which $y_i = 1$ such that each observation has the same chance of being selected.
3. Select k_{-1} observations for which $y_i = -1$ such that each observation has the same chance of being selected.
4. Obtain a median classifier from the combined sample of $k_{+1} + k_{-1}$ points. Assume its output is a score function $F_m(x)$ such that $F_m(x) > 0$ estimates $p(x) > 1/2$.
5. Estimate x as having $p(x) > q$ if $F_m(x) > 0$.

Note here that for $k_{\pm 1} < N_{\pm 1}$ (i.e., under-sampling) the selection can be done by random sampling with or without replacement, and for $k_{\pm 1} > N_{\pm 1}$ (i.e., over-sampling) the selection can be done either by sampling with replacement or by simply augmenting the data by adding replicate observations.

For the reader's convenience, we briefly give the justification for this re-weighting/re-sampling scheme. It can be inferred from Elkan (2001). We work conveniently in terms of populations/distributions as opposed to finite samples. Let X be the predictor random vector and Y the binary response random variable with values in $\{+1, -1\}$. Let further

$$\begin{aligned} p(x) &= P[Y = +1|x] \,, \\ \pi &= P[Y = +1] \,, \\ f_{+1}(x) &= P[x|Y = +1] \,, \\ f_{-1}(x) &= P[x|Y = -1] \,, \end{aligned}$$

which are, respectively, the conditional probability of $Y = +1$ given $X = x$, the unconditional probability of $Y = +1$, and the conditional densities of $X = x$ given $Y = +1$ and $Y = -1$. The densities $f_{\pm 1}(x)$ describe the distributions of the two classes in predictor space. Now the joint distribution of (X, Y) is given by

$$p(y = +1, x) = f_{+1}(x)\,\pi \,, \quad p(y = -1, x) = f_{-1}(x)\,(1 - \pi) \,,$$

and hence the conditional distribution of $Y = 1$ given $X = x$ is

$$p(x) = \frac{f_{+1}(x)\,\pi}{f_{+1}(x)\,\pi + f_{-1}(x)\,(1 - \pi)} \,,$$

which is just Bayes theorem. Equivalently we have

$$\frac{p(x)}{1 - p(x)} = \frac{f_{+1}(x)\,\pi}{f_{-1}(x)\,(1 - \pi)} \,,$$

or

$$\frac{p(x)}{1 - p(x)} \Big/ \frac{\pi}{1 - \pi} = \frac{f_{+1}(x)}{f_{-1}(x)} \,. \tag{7}$$

Now compare this situation with another one that differs only in the mix of class labels, call them π^* and $1 - \pi^*$, as opposed to π and $1 - \pi$. Denote by $p^*(x)$ the conditional probability of $Y = 1$ given $X = x$ under this new mix. From Equation (7) it is obvious that $p(x)$ and $p^*(x)$ are functions of each other, best expressed in terms of odds:

$$\frac{p^*(x)}{1 - p^*(x)} = \frac{p(x)}{1 - p(x)} \cdot \frac{1 - \pi}{\pi} \cdot \frac{\pi^*}{1 - \pi^*} \,.$$

Obviously thresholds q on $p(x)$ and q^* on $p^*(x)$ transform the same way. If $\pi/(1-\pi)$ is the original unconditional ratio of class labels, we ask what ratio $\pi^*/(1-\pi^*)$ we need for mapping the desired q to $q^* = 1/2$ or, equivalently, $q/(1-q)$ to $q^*/(1-q^*) = 1$. The answer is given by the condition

$$1 \;=\; \frac{q^*}{1-q^*} \;=\; \frac{q}{1-q} \cdot \frac{1-\pi}{\pi} \cdot \frac{\pi^*}{1-\pi^*} \;.$$

Solving for $\pi^*/(1-\pi^*)$, the desired marginal ratio of class labels is

$$\frac{\pi^*}{1-\pi^*} \;=\; \frac{\pi}{1-\pi} \Big/ \frac{q}{1-q} \;,$$

which justifies the algorithm above.

3.2 Difficulties With Over- and Under-Sampling

In principle, both the over- and under-sampled boosting algorithms should be effective q-classifiers for applications in which standard boosting is an effective classifier. However, there are some difficulties with each.

The problem with under-sampling is straightforward: a portion of data from the minority class is not used and consequently the effective sample size is smaller than the actual number of instances in the training data. For training data with a large number of observations, this may not be problematic. However, for small data sets or cases in which the desired quantile is close to zero or one, this can pose difficulties. In such cases, over-sampling may be preferable, although there have been attempts at more elaborate under-sampling techniques which use all the data (Chan and Stolfo, 1998).

Over-sampling has a less obvious problem that is particular to boosting and tends to show up after a large number of iterations. The algorithm is an effective q-classifier after a small number of iterations; however, just as with estimating probabilities using the link function, as the number of iterations increases it tends to revert to a median classifier ($p(x) > 1/2$). The reason for this limiting behavior has to do with the fact that the augmented training set includes many replicates of the same (x,y) pair. As boosting re-weights the training samples, the tied points all change their weights jointly, and their multiplicity is invisible in the iterations because only the sum of their weights is visible. In effect, boosting with over-sampled training data amounts to initializing the weights to values other than $1/n$, but nothing else. For increasing numbers of iterations, the effect of the initial weights diminishes and disappears in the limit.

For illustration we return to the two-dimensional circle example using a sample size of $n = 1000$. We will attempt to estimate the quantile $q = 0.9$ with over-sampled AdaBoost by augmenting the training data with 8 more replicates of each x_i for which $y_i = -1$ ($k_{+1} = N_{+1}$ and $k_{-1} = 9N_{-1}$). The left panel of Figure 12 shows the estimate of this quantile on the hold-out sample after 50 iterations. It appears to be a reasonable estimate of the true quantile which the reader should recall is given by the boundary of the red and black in the right panel of Figure 1. However, the middle panel of Figure 12 shows the estimate for the same simulation after 1000 iterations. In this plot it is clear that the circle has grown larger and has begun to revert back to a median classifier.

One solution to this problem is to add a small amount of random noise to the predictor x so that the replicates are not duplicated. While this *jittering* does add undesirable noise to the estimates of the level curves of the function $p(x)$, the amount of noise needed to alleviate this problem is not large and will only shift the boundary of the Bayes rule in the predictor space by a small amount.

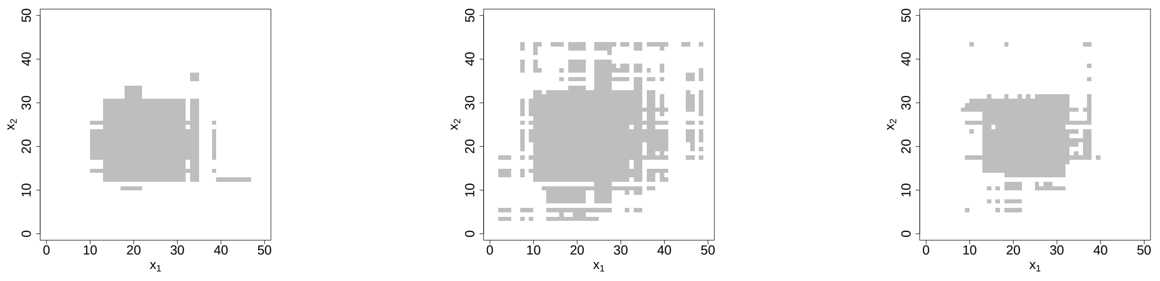

Figure 12: Estimates of the $q = 0.90$ quantile for the circle example. Left panel is $m = 50$, middle panel is $m = 1000$, and right panel is jittering applied for $m = 5000$.

We have achieved good results by adding *iid* uniform $(-\nu\sigma_j, \nu\sigma_j)$ noise to each of the d predictors where σ_j is the standard deviation for the jth predictor in the data set and ν is a tuning parameter that can be chosen to optimize performance. Good values will depend on the size of the training data, sample size, the number of dimensions, noise level, etc. Also, if desired one could consider using more complex procedures to break ties, such as the "SMOTE" algorithm (Chawla et al., 2002, 2003) described before.

As mentioned earlier, we use the acronym "JOUS-Boost" for the combination of over/under-sampling and jittering.

We note that for our procedure we only add noise to the values in the training data for which over-sampling creates replication. For instance, to estimate the $q = 0.9$ quantile in the circle example, the augmented data contains nine instances for every observation from the original training data for which $y = -1$. To eight of these nine we add the *iid* noise and leave the ninth unchanged. No noise is added to the observations for which $y = +1$, as none are replicated in the augmented data. In this way the augmented data contains exactly one copy of the original training data.

The third panel in Figure 12 shows the estimate of $q = 0.9$ for this same simulation after $m = 5000$ iterations with the noise added as described using $\nu = 1/4$. The algorithm still appears to be estimating the $q = 0.9$ quantile even with this large number of iterations and even with the training error on the augmented data set having been zero for the last 1000 of the 5000 total iterations. Note that while the training data does contain a number of $+1$'s outside this grey circle (left panel of Figure 1), the interpolation on these points is so extremely local that it affects only a very few points in our hold-out sample. This gives insight into why boosting classification trees can be an effective classifier in noisy environments despite fitting all the training data completely. From a probabilistic point of view, the training data has zero measure with respect to the population. More discussion of this phenomenon can be found in Mease and Wyner (2007).

4. From a *q*-Classifier to a Probability Estimator

We have discussed and illustrated how a median finder like AdaBoost can be converted to a q-classifier by under-sampling or over-sampling and jittering the data. However, if estimation of the CCPF is the end goal, it is still necessary to further convert the q-classifier to an estimator of the conditional class probabilities. We next present a simple general algorithm to do just that. That is, the following procedure can be used to produce an actual estimate $\hat{p}(x)$ for the CCPF $p(x)$ from a range of q-classifiers.

First a quantization level $\delta > 2$ is fixed. Next, q-classification is carried out on the data for a range of quantiles $q = 1/\delta, 2/\delta, ..., 1 - 1/\delta$. For each q and every x this provides an estimate of $\mathrm{I}\{p(x) \geq q\}$ which we will denote as $\hat{D}_q(x) \in \{0,1\}$. While it is clear that $\mathrm{I}\{p(x) \geq q\}$ is monotonically decreasing in q, the estimates $\hat{D}_q(x)$ will not generally be monotonic, especially when $p(x)$ is close to q. Thus, to produce an estimate of $p(x)$ from the sequence of quantile-based estimates $\hat{D}_q(x)$, monotonicity of the level sets must be achieved by brute force. One solution is to begin with the median $\hat{D}_{0.5}(x)$. The algorithm is as follows.

- If $\hat{D}_{0.5}(x) = 1$ then let $\hat{p}(x) = \min\{q > 0.5 : \hat{D}_q(x) = 0\} - \frac{1}{2\delta}$. If no such q is found take $\hat{p}(x) = 1 - \frac{1}{2\delta}$.
- If $\hat{D}_{0.5}(x) = 0$ then let $\hat{p}(x) = \max\{q < 0.5 : \hat{D}_q(x) = 1\} + \frac{1}{2\delta}$. If no such q is found take $\hat{p}(x) = \frac{1}{2\delta}$.

5. Performance Comparisons

In this section we seek to quantitatively compare the CCPF estimation performance of JOUS-Boost to some competing methods. The competing methods we will consider are LogitBoost, P-AdaBoost, nearest neighbor 10, Random Forests and classification trees from Rpart. We will compare performance on the simulated data sets presented earlier as well as four real data sets taken from the UCI ML Database.

We note that the purpose of these comparisons is not to argue that we have derived a superior method of estimating probabilities/quantiles using boosting. Rather, as discussed in Section 1.2, our purpose is to illustrate that the overfitting of probability/quantile estimates for regular, off-the-shelf boosting is not a problem inherent in boosting, but rather a consequence of believing that applying a link function to the scores should be used to recover probabilities. Most of the difficulty with probability/quantile estimation is solved by simply adopting the more modest view of boosting as a classifier and relinquishing the view of boosting as a probability estimator. Our experiments reveal that simple over- or under- sampling (which is the basis for our algorithm) solves the probability/quantile overfitting problems and produces a competitive algorithm. If we were instead to make an argument for the superiority of the algorithm, there are a large number of variations of boosting, some of which were mentioned in Section 1, which we would need to consider as competitors. These include direct cost-optimization boosting methods as well as regularizing boosting by slowing the learning rate or restricting the flexibility of the weak learners as in the case of decision stumps mentioned earlier.

5.1 Implementation Details

We will use the algorithm in Section 4 to produce an estimate of the CCPF from JOUS-Boost. Specifically, we will apply this algorithm with $\delta = 10$. Consequently our estimate $\hat{p}(x)$ for $p(x)$ at any x will be one of $\{0.05, 0.15, ..., 0.95\}$. This will of course involve performing AdaBoost on nine artificial training data sets. For under-sampling we will create these data sets for each value of q by sampling without replacement from the original training data using $k_{+1} = (1-q)N_{+1}$ and $k_{-1} = qN_{-1}$ (rounded appropriately). The one exception is $q = 0.5$ for which we simply leave the data unmodified, that is, $k_{+1} = N_{+1}$ and $k_{-1} = N_{-1}$. For over-sampling we will create these data sets by replicating observations in the training data using $k_{+1} = \delta(1-q)N_{+1}$ and $k_{-1} = \delta q N_{-1}$, except again in the case where $q = 0.5$ for which we use the original training data. For over-sampling we will add noise as described earlier to any artificially replicated values. Note that one attractive feature of this scheme is that for classification at the 1/2 quantile, the resulting probabilities by construction give rise to a classification rule equivalent to that of AdaBoost on the original data.

To minimize disagreement between the nine quantile estimates we will restrict the sampling such that any (artificial) data set with a smaller value of N_{+1} than that of a second (artificial) data set must not contain any observations for $y = +1$ which are not also in the second data set. The same is true for N_{-1} and $y = -1$. For example, the observations with $y = -1$ in the data set for estimating $q = 0.6$ are a proper subset of those in the data set for $q = 0.7$. Also for the purposes of minimizing the disagreement across the different values of q, when over-sampling we will "recycle" the noise across the different values of q; any observation that appears with noise added to it in more than one of these artificial data sets will have the same value for the noise added to it in all data sets.

The tuning parameter ν will be fixed at a value of 1 throughout the remainder of this article. We have found this to be a fairly "medium" noise level. For the data sets we considered for which the misclassification error and Bayes error are small, values of ν smaller than 1 work better. Likewise, for the data sets for which the misclassification error or Bayes error are larger, values of ν larger than 1 work better. We choose not to tune this parameter so as to provide for a more fair comparison with P-AdaBoost and LogitBoost; however, in practice tuning this parameter through cross-validation would be desirable.

5.2 Simulated Data Sets

We begin by examining the performance for the five simulated data sets presented earlier. These are the four simulations based on the 10-Norm model, first introduced by Friedman et al. (2000) and defined in (2), and the 1000-dimensional simulation defined in (6). For each of these the over-sampling and under-sampling algorithms using AdaBoost as described above are shown on the corresponding plots in Section 2 by the red and green curves respectively.

The original simulation by Friedman et al. (2000) used $n = 5000$ and $\gamma = 10$. The plots in Figure 8 display the misclassification error and our probability scoring functions averaged over the hold-out sample. The under-sampled AdaBoost algorithm performs better than LogitBoost, which exhibits overfitting around 200 iterations, but it does not perform as well as P-AdaBoost. The over-sampled AdaBoost algorithm has the worst performance of the four; however, if the noise parameter ν is set at a smaller value, its performance in this simulation is superior to that of the other three. Again, the amount of noise ν should be regarded as a tuning parameter, but we have kept it fixed throughout these experiments. Note that the fact that P-AdaBoost does not suffer from overfitting in this simulation can be attributed to the low Bayes error, large sample size and moderate number of dimensions as mentioned earlier.

In Figure 9 we saw that when γ was decreased to 0.5 the LogitBoost algorithm severely overfit the probabilities, but P-AdaBoost did not. The over- and under-sampled AdaBoost algorithms perform well in this simulation with the under-sampled AdaBoost algorithm being comparable to P-AdaBoost and over-sampling only slightly worse. Again, by using a smaller value for ν, the over-sampled AdaBoost can be tuned to outperform the others in this simulation.

When the sample size is decreased from 5000 to 500 for this model, Figures 5 and 6 show that both P-AdaBoost and LogitBoost overfit the probabilities more and more as the iterations are increased. However, the under- and over-sampled AdaBoost algorithms actually appear to asymptote to their minima. They provide stable probability estimation regardless of the number of iterations, mimicking the behavior of AdaBoost with respect to misclassification error as intended. Similar behavior can be observed in the plots in Figure 10 for the 1000-dimensional example with respect to under-sampling. Over-sampling was not performed for this simulation due to computational limitations.

Tables 1-4 display misclassification error, squared loss, log loss and exponential loss for P-AdaBoost and LogitBoost as well as the over- and under-sampled AdaBoost algorithms at $m = 800$ iterations. These tables also give the values for other competing methods for CCPF estimation: CART, Random Forests and nearest neighbor 10. The implementation of CART is that of the default classification tree grown by the function Rpart in R. Random Forests are also implemented in R using the default settings. The Bayes column for misclassification gives the error for the hold-out sample using the true (unknown) Bayes rule. All results are based on the same training and test sets in each case. With P-AdaBoost and LogitBoost we continue to threshold all probability estimates at 0.05 and 0.95 as before when computing log loss and exponential loss (but not squared loss). This prevents the over- and under-sampled AdaBoost algorithms from gaining an unfair advantage against overfitting due to their course quantization. We do not include JOUS-Boost in Table 1 because the classification rules for JOUS-Boost are equivalent to AdaBoost by definition.

From these tables it can be seen that the over-sampled and under-sampled AdaBoost algorithms provide a competitive method for probability estimation for these simulations. Their results are comparable to that of Random Forests and nearest neighbor 10 and superior to CART. They do not suffer from the severe overfitting of P-AdaBoost and LogitBoost. Further, even better performance may be achieved for over-sampling by allowing tuning of the parameter ν which controls the amount of noise added to the replicates.

5.3 Real Data Sets

We now consider the two real data sets “Sonar” and “Satimage”. Both of these are from the UCI ML database and are used in the article by Friedman et al. (2000).

When using real data sets we can no longer use our scoring rules on the probabilities since the true CCPF is unknown. Instead, we will substitute the value for the true class probabilities in

Simulation Name	AdaBoost	Logit-Boost	CART	Random Forests	Nearest Neighbor 10	Bayes
10-Norm(n=500, γ=10)	0.23	0.25	0.39	0.27	0.28	0.03
10-Norm(n=500, γ=0.5)	0.34	0.36	0.38	0.36	0.37	0.24
10-Norm(n=5000, γ=10)	0.10	0.12	0.31	0.16	0.19	0.03
10-Norm(n=5000, γ=0.5)	0.28	0.29	0.40	0.27	0.30	0.22
1000 Dimensions	0.34	0.35	0.50	0.37	0.44	0.03
Sonar	0.11	0.16	0.26	0.15	0.31	-
Sonar w/ Noise	0.22	0.27	0.35	0.21	0.36	-
Satimage	0.06	0.06	0.12	0.06	0.08	-
Small Satimage w/ Noise	0.21	0.20	0.21	0.18	0.18	-
German Credit	0.25	0.25	0.27	0.23	-	-
Connect 4	0.22	0.22	0.34	0.24	-	-

Table 1: Misclassification Error Comparison

Simulation Name	JOUS (Over-Sampling)	JOUS (Under-Sampling)	P-AdaBoost	Logit-Boost	CART	Random Forests	Nearest Neighbor 10
10-Norm(n=500, γ=10)	0.14	0.15	0.20	0.22	0.25	0.17	0.16
10-Norm(n=500, γ=0.5)	0.07	0.07	0.19	0.19	0.12	0.06	0.07
10-Norm(n=5000, γ=10)	0.09	0.06	0.05	0.07	0.19	0.11	0.12
10-Norm(n=5000, γ=0.5)	0.03	0.03	0.03	0.10	0.09	0.03	0.05
1000 Dimensions	-	0.18	0.27	0.28	0.23	0.21	0.22
Sonar	0.08	-	0.11	0.16	0.21	0.13	0.20
Sonar w/ Noise	0.15	-	0.22	0.26	0.28	0.17	0.21
Satimage	0.06	0.08	0.06	0.05	0.10	0.05	0.05
Small Satimage w/ Noise	0.15	0.16	0.19	0.20	0.17	0.14	0.15
German Credit	-	0.18	0.24	0.22	0.20	0.16	-
Connect 4	-	0.20	0.21	0.21	0.23	0.16	-

Table 2: Squared Loss Comparison

Simulation Name	JOUS (Over-Sampling)	JOUS (Under-Sampling)	P-AdaBoost	Logit-Boost	CART	Random Forests	Nearest Neighbor 10
10-Norm(n=500, γ=10)	0.49	0.50	0.69	0.75	0.76	0.55	0.54
10-Norm(n=500, γ=0.5)	0.63	0.63	1.04	1.06	0.77	0.61	0.63
10-Norm(n=5000, γ=10)	0.38	0.29	0.25	0.33	0.61	0.43	0.43
10-Norm(n=5000, γ=0.5)	0.53	0.52	0.52	0.77	0.66	0.54	0.57
1000 Dimensions	-	0.62	0.93	0.94	0.71	0.68	0.70
Sonar	0.26	-	0.36	0.51	0.65	0.41	0.56
Sonar w/ Noise	0.48	-	0.70	0.80	0.86	0.51	0.60
Satimage	0.21	0.29	0.22	0.21	0.35	0.19	0.19
Small Satimage w/ Noise	0.51	0.49	0.60	0.63	0.52	0.46	0.46
German Credit	-	0.55	0.75	0.68	0.59	0.49	-
Connect 4	-	0.59	0.68	0.65	0.65	0.49	-

Table 3: Log Loss Comparison

Simulation Name	JOUS (Over-Sampling)	JOUS (Under-Sampling)	P-AdaBoost	Logit-Boost	CART	Random Forests	Nearest Neighbor 10
10-Norm(n=500, γ=10)	0.78	0.79	1.12	1.20	1.10	0.85	0.84
10-Norm(n=500, γ=0.5)	0.95	0.94	1.61	1.63	1.12	0.92	0.93
10-Norm(n=5000, γ=10)	0.65	0.53	0.48	0.61	0.92	0.71	0.71
10-Norm(n=5000, γ=0.5)	0.82	0.80	0.80	1.20	0.97	0.83	0.86
1000 Dimensions	-	0.92	1.43	1.43	1.02	0.99	1.01
Sonar	0.50	-	0.67	0.88	1.00	0.69	0.85
Sonar w/ Noise	0.79	-	1.14	1.26	1.28	0.81	0.90
Satimage	0.43	0.53	0.46	0.45	0.62	0.41	0.40
Small Satimage w/ Noise	0.85	0.79	0.99	1.03	0.82	0.76	0.77
German Credit	-	0.86	1.19	1.08	0.90	0.78	-
Connect 4	-	0.89	1.11	1.06	0.96	0.78	-

Table 4: Exponential Loss Comparison

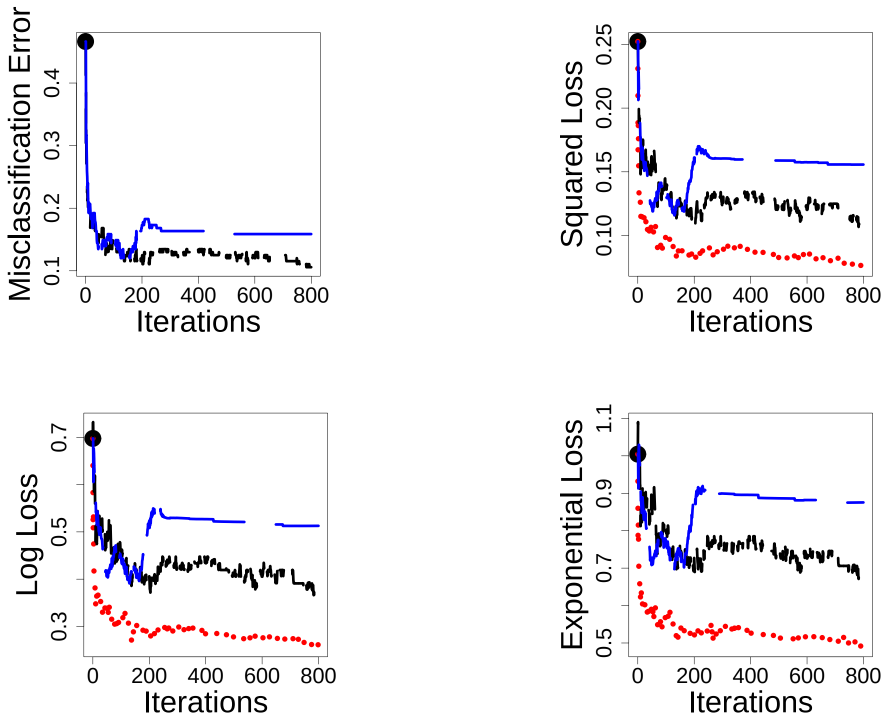

Figure 13: Misclassification error, squared loss, log loss and exponential loss for Sonar. **Black (Short Dashes)**: P-AdaBoost. **Blue (Long Dashes)**: LogitBoost. **Red (Dotted)**: JOUS-Boost, over-sampled.

the hold-out samples with the indicator of whether or not the observed class is one. That is, we substitute the true probability function $p(x) = \mathrm{P}[Y = 1|x]$, which is unknown by $1\{y = 1\}$, which is observed. With this substitution we can evaluate the loss functions (Equations (3), (4) and (5)) as before. Note that since all three of the scoring rules we are using are *proper scoring rules* (Savage, 1973) they are minimized in expectation as a function of $\hat{p}$ by the true probability p even with this substitution of the indicator for the probability.

The Sonar data set has 208 observations, $d = 60$ predictors and two classes. Since there is no designated test sample, 10-fold cross validation was used. While Figure 13 reveals both P-AdaBoost and LogitBoost have a general decreasing trend with respect to the probability scoring rules (excepting the strange jump by LogitBoost right before $m = 200$), both algorithms clearly could benefit from not estimating all probabilities with (essentially) zero and one as evidenced by the superior performance of the over-sampled AdaBoost algorithm. The fact that both LogitBoost and P-AdaBoost estimate all probabilities with values close to zero or one is reflected in the similarity of their respective values for squared loss and misclassification error. We did not apply under-sampling on this data set due to the small number of observations.

The negative effects of overfitting probabilities with P-AdaBoost and LogitBoost are amplified if we consider artificially adding "label noise." By label noise we mean that with some probability

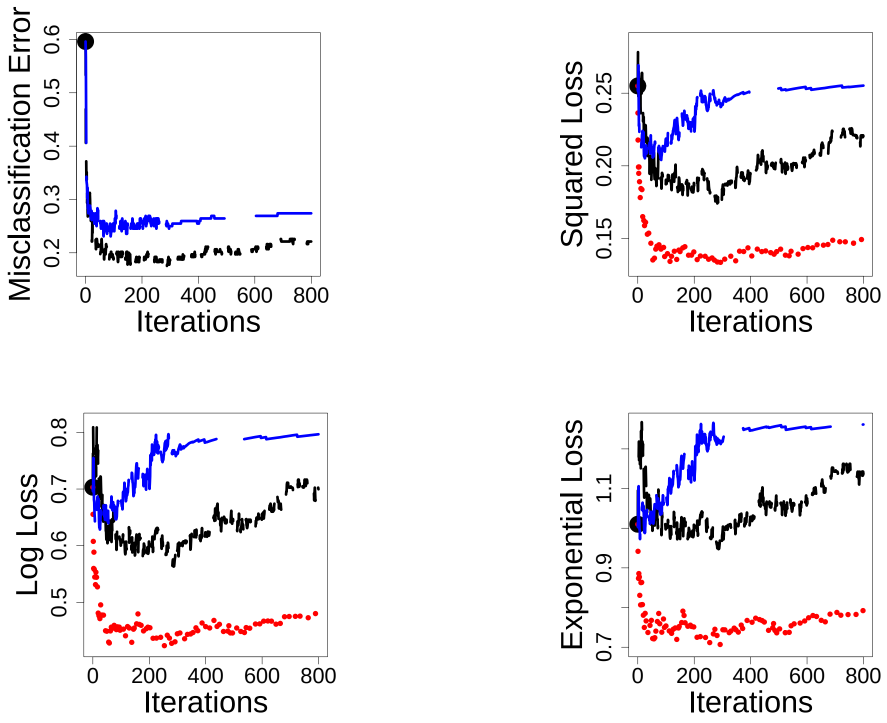

Figure 14: Misclassification error, squared loss, log loss and exponential loss for Sonar with noise. **Black (Short Dashes)**: P-AdaBoost. **Blue (Long Dashes)**: LogitBoost. **Red (Dotted)**: JOUS-Boost, over-sampled.

q we randomly change the signs of the class labels in the data set. The plots in Figure 14 show the results for label noise of $q = 0.1$ over the same ten fold resamples.

The Satimage data set has a designed training set with $n = 4435$ observations and a hold-out set of $n^* = 2000$ observations. There are $d = 36$ predictors. The data set has six classes labelled as 1, 2, 3, 4, 5 and 7. Since we are only dealing with two-class classification for the purposes of this article we will take one class to be 1 and 7 and the other class to be 2, 3, 4 and 5.

The four plots in Figure 15 show misclassification error, squared loss, log loss and exponential loss averaged over the designated hold-out sample of $n^* = 2000$ for over- and under- sampled AdaBoost, LogitBoost and P-AdaBoost. Neither P-AdaBoost nor LogitBoost seem to increase as a result of overfitting probabilities and perform similarly. Over-sampled AdaBoost performs only slightly better than both P-AdaBoost and LogitBoost, while under-sampled AdaBoost is the worst of the four, although it is still decreasing somewhat quickly at $m = 800$.

While P-AdaBoost and LogitBoost seem to perform well for probability estimation on the Satimage data set, the negative effects of the overfitting can be seen by again adding 10% label noise. Adding this noise severely diminishes the probability estimation of LogitBoost, although P-AdaBoost still performs well (not shown). To see the impact of overfitting for P-AdaBoost in this noisy environment it is sufficient to take a random sample of 1000 from the original 4435 ("Small Satimage"). The corresponding plots are given in Figure 16. Both under-sampling and

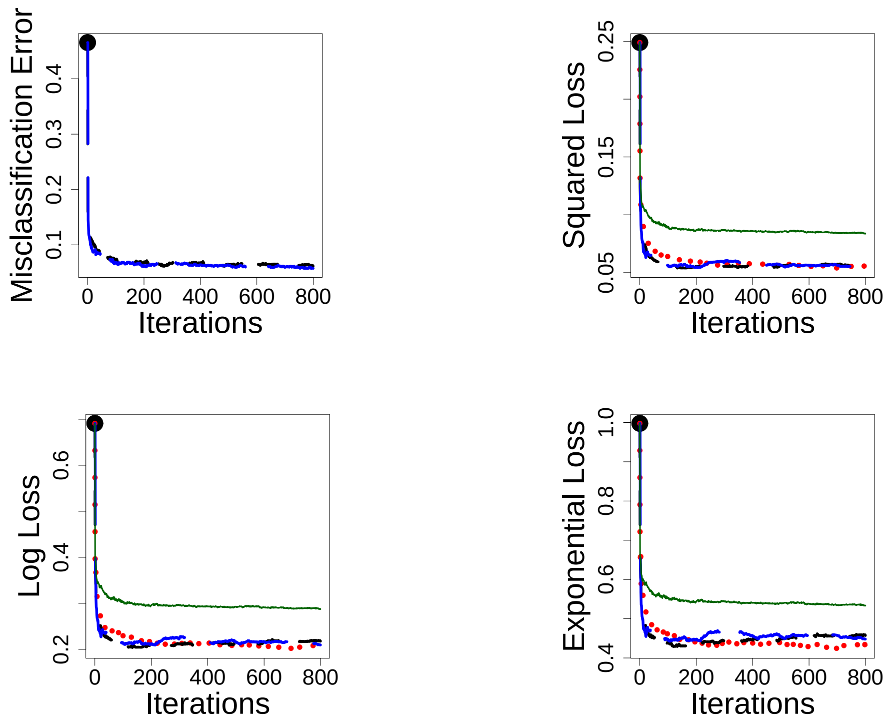

Figure 15: Misclassification error, squared loss, log loss and exponential loss for Satimage. **Black (Short Dashes)**: P-AdaBoost. **Blue (Long Dashes)**: LogitBoost. **Red (Dotted)**: JOUS-Boost, over-sampled. **Green (Solid)**: JOUS-Boost, under-sampled.

over-sampling continue to perform well on this smaller data set, with under-sampling now being superior to over-sampling with respect to log loss and exponential loss.

Tables 1-4 also give results at $m = 800$ iterations for the Sonar and Satimage data sets as well as the modified versions considered. The JOUS-Boost algorithm outperforms CART, Random Forests and nearest neighbor 10 on Sonar and is competitive for Satimage.

5.4 Data Sets with Categorical Predictors

Up until this point, we have considered data sets with only continuous predictors. To illustrate that the overfitting in terms of CCPF estimation with P-AdaBoost and LogitBoost can still occur with categorical predictors, we now turn our attention to two such data sets from the UCI ML database: “German Credit” and “Connect 4”. The German Credit data set contains 1000 observations with 20 predictors, 13 of which are categorical and 7 of which are continuous. The Connect 4 data set contains 67,557 observations with 42 predictors, all of which are categorical. This data set has three classes so we merged the two classes “loss” and “draw” into a single class.

The results for these two data sets are displayed in Figures 17 and 18. The final results at $m = 800$ iterations are also included in Tables 1-4. For German Credit, 10-fold cross validation was used, while for Connect 4, the data was randomly split into 40,000 training observations and 27,557

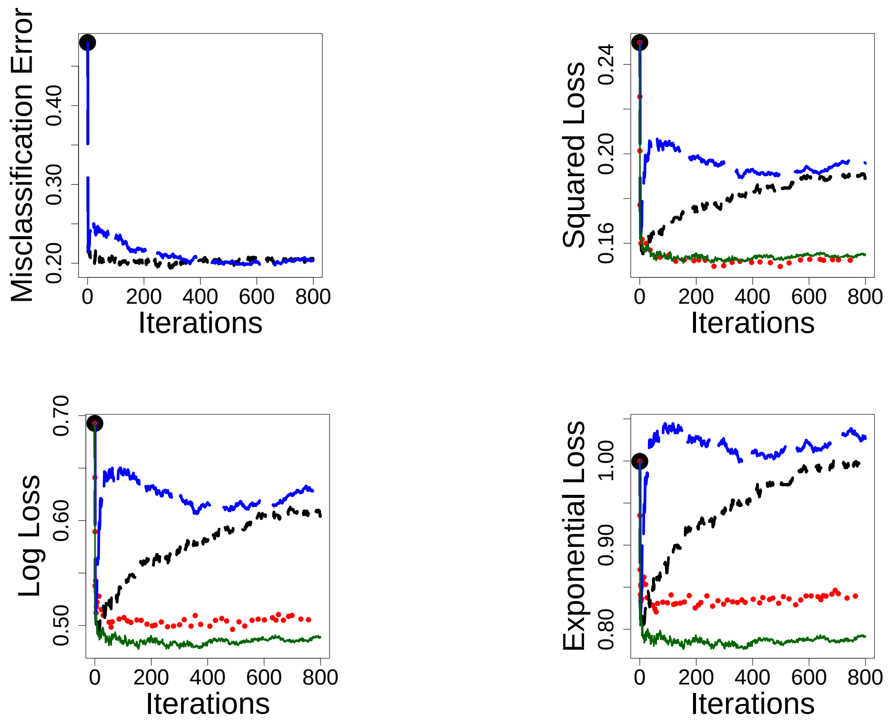

Figure 16: Misclassification error, squared loss, log loss and exponential loss for Small Satimage with noise. **Black (Short Dashes)**: P-AdaBoost. **Blue (Long Dashes)**: LogitBoost. **Red (Dotted)**: JOUS-Boost, over-sampled. **Green (Solid)**: JOUS-Boost, under-sampled.

test observations. Also, due to the large size of this data set, we used 2^{10}-node trees instead of 8-node trees, as this gave superior classification performance. We considered only under-sampling for these two data sets. Over-sampling was not used because it is not obvious how to add noise to categorical predictors.

As can be seen in Figures 17 and 18, both LogitBoost and P-AdaBoost exhibit substantial overfitting of the CCPF on these two data sets, while under-sampled AdaBoost shows only a relatively small amount of overfitting for German Credit and no overfitting for Connect 4. In terms of performance, Tables 1-4 show that under-sampled AdaBoost outperforms CART for both data sets, but that Random Forests is most effective overall.

6. Concluding Remarks

Boosting algorithms such as AdaBoost are known to perform well for classification and are very resistant to overfitting with respect to misclassification error, even though conditional class probability estimates eventually diverge to zero and one, implying complete overfit in terms of CCPF estimation but not classification. With Friedman et al.'s (2000) analysis of boosting in mind, we allow that one may still choose to interpret boosting classification trees as additive logistic regression,

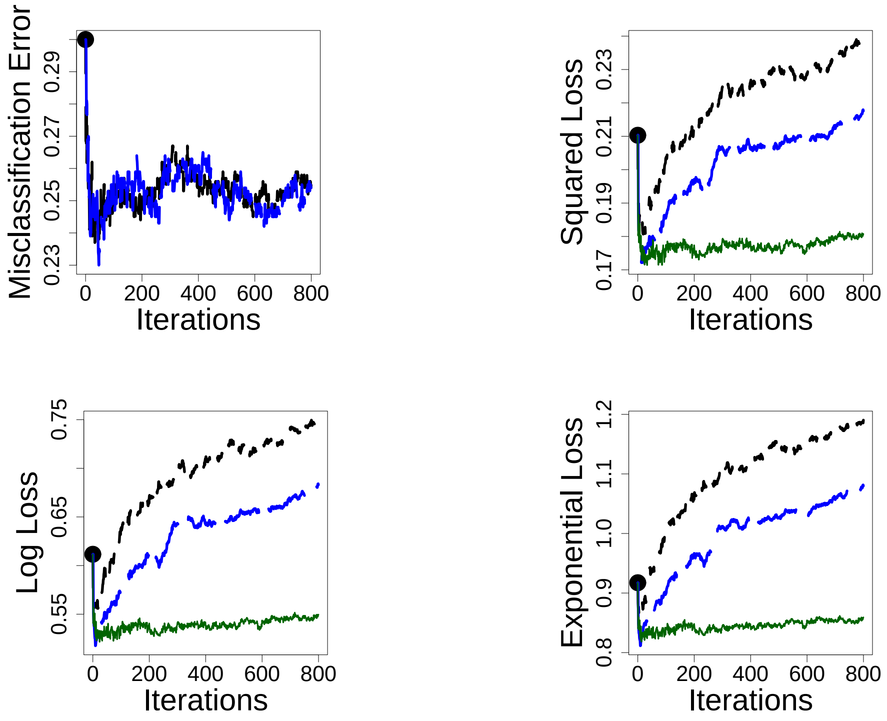

Figure 17: Misclassification error, squared loss, log loss and exponential loss for German Credit. **Black (Short Dashes)**: P-AdaBoost. **Blue (Long Dashes)**: LogitBoost. **Green (Solid)**: JOUS-Boost, under-sampled.

with the understanding, however, that the probabilities may need to be drastically overfit to obtain optimal classification. That is, when attempting classification, one may lose the connection with conditional class probability estimation.

There remains the practical problem of classification in the presence of unequal misclassification costs, or equivalently classification at quantiles other than 1/2, or yet equivalently classification for future base rates that are different from those in the training sample. Viewing AdaBoost or LogitBoost as a form of logistic regression may suggest a *practice* that often leads to poor performance. If one seeks a quantile other than the median and adopts this view of boosting algorithms, then one must hope to stop the boosting process early enough that the absolute value of the score function is not too large relative to the link function evaluated at the quantile of interest, but at the same time late enough that the boosting process has had ample time to learn the truth. Even if such a stopping time exists (we have shown examples where it does not), it may be difficult to find.

In contrast, the modest view that boosting produces only median classifiers suggests more effective modifications to the algorithm in order to obtain quantiles other than the median. One such modification is based on over/under-sampling with a fix (jittering) to eliminate problems with ties (here called JOUS-Boost). We have indicated that JOUS-Boost is often more successful through a number of examples, including some used by Friedman et al. (2000).

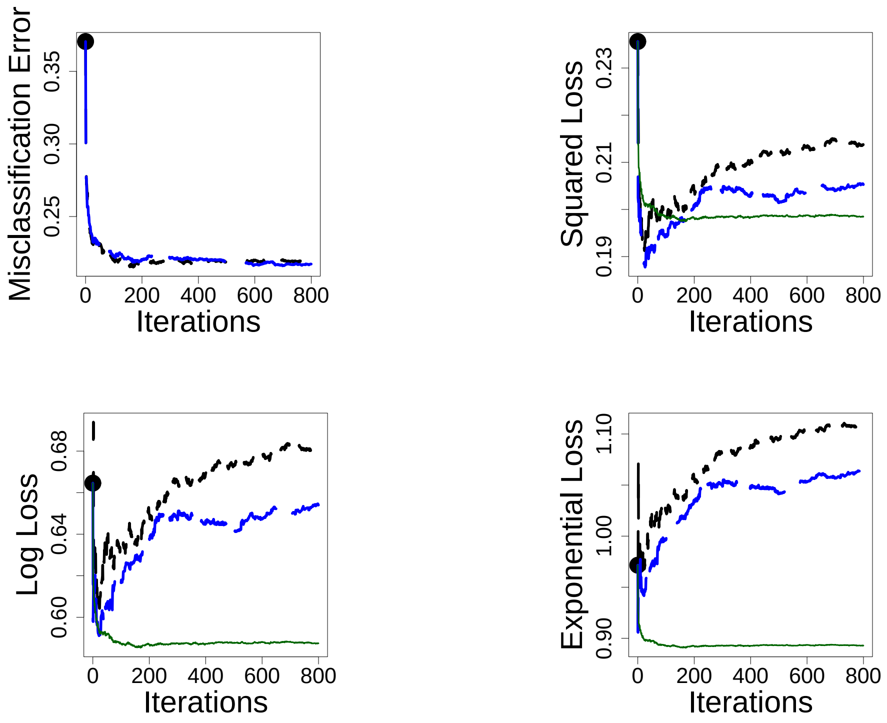

Figure 18: Misclassification error, squared loss, log loss and exponential loss for Connect 4. **Black (Short Dashes)**: P-AdaBoost. **Blue (Long Dashes)**: LogitBoost. **Green (Solid)**: JOUS-Boost, under-sampled.

Finally, we have demonstrated that JOUS-Boost classifiers obtained for a grid of quantiles can be combined to create class probability estimators that perform well with respect to probability estimation.

Acknowledgments

David Mease's work was supported by an NSF-DMS post-doctoral fellowship. We are grateful to Abba Krieger and Dean Foster for helpful comments and suggestions.

References

P. J. Bickel, Y. Ritov, and A. Zakai. Some theory for generalized boosting algorithms. *Journal of Machine Learning Research*, 7:705–732, 2006.

G. Blanchard, G. Lugosi, and N. Vayatis. On the rate of convergence of regularized boosting classifiers. *Journal of Machine Learning Research*, 4:861–894, 2003.

A. Buja, W. Stuetzle, and Y. Shen. Loss functions for binary class probability estimation and classification: Structure and applications. 2006.

P. Chan and S. Stolfo. Toward scalable learning with non-uniform class and cost distributions: A case study in credit card fraud detection. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, pages 164–168, 1998.

N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.

N. V. Chawla, A. Lazarevic, L. O. Hall, and K. W. Bowyer. Smoteboost: Improving prediction of the minority class in boosting. In *Proceedings of 7th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 107–119, 2003.

W. W. Cohen and Y. Singer. A simple, fast, and effective rule learner. In *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI)*, pages 335–342, 1999.

M. Collins, R. E. Schapire, and Y. Singer. Logistic regression, adaboost and bregman distances. In *Computational Learing Theory*, pages 158–169, 2000.

M. Dettling and P. Buhlmann. Boosting for tumor classification with gene expression data. *Bioinformatics*, 19:1061–1069, 2003.

N. Duffy and D. Helmbold. Potential boosters? In *Advances in Neural Information Processing Systems*, pages 258–264, 1999.

C. Elkan. The foundations of cost-sensitive learning. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 973–978, 2001.

A. Estabrooks, T. Jo, and N. Japkowicz. A multiple resampling method for learning from imbalanced data sets. *Computational Intelligence*, 20:18–36, 2004.

W. Fan, S. Stolfo, J. Zhang, and P. Chan. Adacost: Misclassification cost-sensitive boosting. In *Proceedings of the 16th International Conference on Machine Learning*, pages 97–105, 1999.

D. P. Foster and R. A. Stine. Variable selection in data mining: Building a predictive model for bankruptcy. *Journal of the American Statistical Association*, 99:303–313, 2004.

Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Machine Learning: Proceedings of the Thirteenth International Conference*, pages 148–156, 1996.

Y. Freund and R. E. Schapire. Discussion of three papers regarding the asymptotic consistency of boosting. *Annals of Statistics*, 32:113–117, 2004.

J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 28:337–374, 2000.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2001.

W. Jiang. Process consistency for adaboost. *Annals of Statistics*, 32:13–29, 2004.

W. Jiang. Does boosting overfit: Views from an exact solution. *Technical Report 00-03, Department of Statistics, Northwestern University*, 2000.

M. Joshi, V. Kumar, and R. Agarwal. Evaluating boosting algorithms to classify rare classes: Comparison and improvements. In *Proceedings of the First IEEE International Conference on Data Mining (ICDM)*, pages 257–264, 2001.

G. Lebanon and J. Lafferty. Boosting and maximum likelihood for exponential models. In *Advances in Neural Information Processing Systems*, 2001.

Y. Liu, Y. Yang, and J. Carbonell. Boosting to correct inductive bias in text classification. In *Proceedings of the Eleventh International Conference on Information and Knowledge Management*, pages 348–355, 2002.

G. Lugosi and N. Vayatis. On the bayes-risk consistency of regularized boosting methods. *Annals of Statistics*, 32:30–55, 2004.

D. Mease and A. Wyner. Evidence contrary to the statistical view of boosting. 2007.

S. Rosset, J. Zhu, and T. Hastie. Boosting as a regularized path to a maximum margin classifier. *Journal of Machine Learning Research*, 5:941–973, 2004.

L. J. Savage. Elicitation of personal probabilities and expectations. *Journal of the American Statistical Association*, 66:783–801, 1973.

K. M. Ting. A comparative study of cost-sensitive boosting algorithms. In *Proceedings of the 17th International Conference on Machine Learning*, pages 983–990, 2000.

B. Zadrozny and C. Elkan. Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 609–616, 2001.

T. Zhang and B. Yu. Boosting with early stopping: Convergence and consistency. *Annals of Statistics*, 33:1538–1579, 2005.