Evaluating Boosting Algorithms to Classify Rare Classes: Comparison and Improvements

Mahesh V. Joshi
IBM T. J. Watson Research Center
P.O.Box 704
Yorktown Heights, NY 10598, USA.
joshim@us.ibm.com

Vipin Kumar
Department of Computer Science
University of Minnesota
Minneapolis, MN 55455, USA.
kumar@cs.umn.edu

Ramesh C. Agarwal
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120, USA.
ragarwal@us.ibm.com

Abstract

Classification of rare events has many important data mining applications. Boosting is a promising meta-technique that improves the classification performance of any weak classifier. So far, no systematic study has been conducted to evaluate how boosting performs for the task of mining rare classes. In this paper, we evaluate three existing categories of boosting algorithms from the single viewpoint of how they update the example weights in each iteration, and discuss their possible effect on recall and precision of the rare class. We propose enhanced algorithms in two of the categories, and justify their choice of weight updating parameters theoretically. Using some specially designed synthetic datasets, we compare the capability of all the algorithms from the rare class perspective. The results support our qualitative analysis, and also indicate that our enhancements bring an extra capability for achieving better balance between recall and precision in mining rare classes.

1 Introduction and Motivation

Recent surge in volumes of data and relatively much smaller increase in the events of interest have brought critical importance to the problem of effectively mining rarely occurring events. One example of this is the click-stream data on the web. A popular e-commerce web site can receive millions of hits in a day, but very small proportion among these hits are of actual interest from the revenue generation point of view. Some work has started to emerge in building descriptive models for the rare events [1, 5]. Classification shows promise in achieving this task. In past few years, boosting has emerged as a competitive meta-technique that has a theoretically justified ability to improve the performance of any *weak* classification algorithm. Various different boosting algorithms have been proposed in the literature [7, 2, 10, 4]. They have been analyzed for their effectiveness [8], and they have been adapted to special tasks [9]. Despite of this abundant work on boosting, no work has dealt directly with evaluating boosting algorithms in the context of mining rare events.

Boosting algorithms work in iterations, each time learning a weak classifier model[1] on a different weighted distribution of training records. After each iteration, the weights of the examples are updated. The intention is to increase the weights of the incorrectly classified examples and decrease the weights of the correctly classified examples. This forces the classifier to focus more on the incorrectly classified examples in the next iteration. The algorithm stops after a pre-specified number of iterations, or based on some measured quality. While classifying a new example, all the models from all the iterations vote in proportion to their accuracy; the class with most votes wins.

The crucial step that we focus on in this paper is the weight update mechanism in each iteration. In the binary classification scenario, there are four kinds of examples after every iteration. From the perspective of rare class C (versus all the other classes clubbed into one class called NC), these can be categorized as follows:

	Predicted as C	Predicted as NC
Actually C	True Positives (TP)	False Negatives (FN)
Actually NC	False Positives (FP)	True Negatives (TN)

We evaluate boosting algorithms from three different categories: those making true or false decision for every example, those choosing to abstain from making any decision on some examples, and those that take misclassification costs into account. One of the contributions of this paper is that,

[1]A weak classifier is the algorithm that, given $\epsilon, \delta > 0$, can achieve at least slightly better error rate, ϵ, than random guessing ($\epsilon \geq 1/2 - \gamma$, where $\gamma > 0$), with a probability $(1 - \delta)$.

we bring out the differences in these algorithms with respect to how they modify the weights on four types of examples: TP, FP, TN, and FN. This insight allows us to qualitatively discuss the effects of their weight modifications on the recall and precision of the target class[2].

A classifier obtains high recall by learning better models for distinguishing FN from TN, and high precision by better distinguishing FP from TP. Recall and precision goals are often conflicting; hence, attacking them simultaneously may not work well, especially when one of the classes is rare. Thus, it is desirable to give different treatment to FPs and FNs. Based on this theme, we propose enhancements to two state-of-the-art algorithms, AdaBoost [7] and SLIPPER [2]. We justify the weight update formulae theoretically.

The theme of applying different update factors to different types of examples is present in the boosting algorithms CSB1, CSB2 [10] and AdaCost [4]. These algorithms take into account different costs of making a false positive prediction versus a false negative prediction. Our study shows that AdaCost has the capability of controlling its emphasis on recall, while trying to focus on precision as well. This makes it a better algorithm in many datasets with rare classes. However, we also show that its over-emphasis on recall may sometimes lead to a much poorer precision.

We validate our qualitative study of weight update effects using some synthetic datasets specially designed for studying rare classes. We show that our proposed enhancements outperform their respective predecessors in achieving better recall-precision balance, and that they have the ability to even outperform the most competitive cost-sensitive algorithm, AdaCost, in some situations.

2 Boosting Algorithms that Do Not Abstain

In this section, we describe algorithms that make a true or false decision on *every* example.

The popular AdaBoost [7] learning algorithm is described in Figure 1. The weak model, h_t, must have an *accuracy* $> 50\%$, in order to increase weights of FP and FN decisions and decrease weights of TP and TN decisions, after each iteration. Note that the same update factor is used for all four types. The α_t value is chosen to minimize the sum of all the weights, Z_t, before the beginning of $(t+1)^{st}$ iteration, which is shown to minimize an upper bound on the training error. Also note that, α_t, voting power of h_t, is monotonic with the accuracy of h_t.

[2]Overall accuracy is not a good metric for evaluating rare class performance. Instead a balance between recall and precision, to be defined later, is desired.

```
Given: Training Set, $\mathcal{T} : \{(x_i, y_i)\}, i = 1 \cdots N$;
$x_i \in \mathcal{X}, y_i \in \{-1, 1\}$; and number of trials, $M$.
Initialize weights $D_1(i) = 1/N$.
for $t = 1 \cdots M$
  i. Learn weak model, $h_t$, using $D_t$.
  ii. Compute importance weight, $\alpha_t$:
      $r_t = \sum_{i=1}^{N} D_t(i) h_t(x_i) y_i;\ \alpha_t = \frac{1}{2} ln \left(\frac{1+r_t}{1-r_t}\right)$   (1)
  iii. Update Weights:
      $D_{t+1}(i) = (D_t(i) exp(-\alpha_t y_i h_t(x_i))/Z_t,$   (2)
      where $Z_t$ is chosen such that $\sum D_{t+1}(i) = 1$.
endfor
Final Model:
  $H(x) = sign\left(\sum_{t=1}^{M} \alpha_t h_t(x)\right)$   (3)
```

Figure 1. *AdaBoost Algorithm [7]*

2.1 Our Proposed Enhancement (RareBoost-1)

We propose an enhancement to AdaBoost, as described in Figure 2. The key observation to make is that we are giving a different treatment to positive and negative predictions. If each model makes every prediction (C or NC) with an accuracy of greater than 50%; i.e., if $TP_t > FP_t$ and $TN_t > FN_t$, then the algorithm will decrease (resp. increase) the weights of correct (resp. incorrect) predictions.

Proof for the choice of α_t^p and α_t^n:
We essentially modify the proof given in [7]. By recursively expanding the weight update rules 8 and 9, and using equation 10, the weights at the end of iteration M are

$$\begin{aligned} D_{M+1}(i) &= \frac{1}{N \prod_t Z_t}(exp(-(\sum_{t:h_t(x_i)\geq 0} \alpha_t^p y_i h_t(x_i) + \\ &\quad \sum_{t:h_t(x_i)<0} \alpha_t^n y_i h_t(x_i)))) \\ &= \frac{exp(-y_i g(x_i))}{N \prod_t Z_t} \end{aligned}$$

Following [7], training error can be minimized by greedily minimizing Z_t, which is the sum of weights after iteration t. $Z_t = Z_t^p + Z_t^n$ can in turn be minimized by minimizing each of its constituent terms. Using notation $u_i = y_i h_t(x_i)$, we have $Z_t^p = \sum_{i:h_t(x_i)\geq 0} D_t(i)exp(-\alpha_t^p u_i)$, and $Z_t^n = \sum_{i:h_t(x_i)<0} D_t(i)exp(-\alpha_t^n u_i)$. We now derive expressions for α_t^p by minimizing Z_t^p. Derivation for α_t^n is symmetric. Using the same linear upper bound as given in [7],

$$Z_t^p \leq \sum_{i:h_t(x_i)\geq 0} D_t(i)\left(\frac{1+u_i}{2}\right) exp(-\alpha_t^p) + D_t(i)\left(\frac{1-u_i}{2}\right) exp(\alpha_t^p)$$

and minimizing it by differentiating it w.r.t. α_t^p, yields

$$\begin{aligned} \alpha_t^{p1} &= \tfrac{1}{2}\, ln\left(\frac{1+\sum_{i:h_t(x_i)\geq 0} D_t(i)y_i h_t(x_i)}{1-\sum_{i:h_t(x_i)\geq 0} D_t(i)y_i h_t(x_i)}\right) \\ &= \tfrac{1}{2}\, ln\left(\frac{1+TP_t-FP_t}{1-TP_t+FP_t}\right) \end{aligned} \quad (4)$$

However, this is not a unique solution. It is just one possible solution for α_t^p obtained by minimizing the tightest *linear* upper bound. Here is another possible solution for α_t^p:

$$\alpha_t^{p2} = \tfrac{1}{2}\, ln\left(\frac{TP_t}{FP_t}\right). \quad (5)$$

This choice of α_t^{p2} simplifies the qualitative analysis that we present later. In fact, for the range of $h_t = \{-1, +1\}$; i.e., by ignoring confidence-rating of a decision, we can show that use of α_t^{p2} achieves smaller value of Z_t^p than use of α_t^{p1}. Using α_t^{p2}, we obtain $Z_t^{p2} = 2\sqrt{TP_t\, FP_t}$, since $u_i = -1$ or $+1$. Similarly, using α_t^{p1}, we obtain

$$Z_t^{p1} = TP_t \frac{\sqrt{\delta + 2\,FP_t}}{\sqrt{\delta + 2\,TP_t}} + FP_t \frac{\sqrt{\delta + 2\,TP_t}}{\sqrt{\delta + 2\,FP_t}};$$

where $\delta = (1 - TP_t - FP_t) \geq 0$. The following expression can be easily derived using formulae for Z_t^{p1} and Z_t^{p2}.

$$Z_t^{p1} - Z_t^{p2} = \frac{\beta^2}{\sqrt{(\delta + 2\,TP_t)(\delta + 2\,FP_t)}} \geq 0$$

where

$$\beta = (\sqrt{\delta\, TP_t + 2\,TP_t\, FP_t} - \sqrt{\delta\, FP_t + 2\,TP_t\, FP_t})$$

Thus, $Z_t^{p2} \leq Z_t^{p1}$. Figure 2 uses $\alpha_t^p = \alpha_t^{p2}$. ♠

3 Boosting Algorithms that Abstain

In this section, we describe boosting algorithms using base classifiers that abstain from making any decision on some training examples. SLIPPER [2] is one such algorithm. It focuses on building single-rule model for only one of the classes in all the iterations. If a good model cannot be built, it uses default model for that iteration, which predicts everything to be of that class. In case of rare classes, its

```
Given: $\mathcal{T}$, M.
Initialize weights $D_1(i) = 1/N$.
for $t = 1 \cdots M$
  i. Learn weak model, $h_t$, using $D_t$.
  ii. Compute importance weight for positive predictions, $\alpha_t^p$:
        $TP_t = \sum_{i:h_t(x_i)\geq 0, y_i>0} D_t(i)h_t(x_i)$;
        $FP_t = \sum_{i:h_t(x_i)\geq 0, y_i<0} D_t(i)h_t(x_i)$
        $\alpha_t^p = 1/2\, ln\,(TP_t/FP_t)$      (6)
  iii. Compute importance weight for negative predictions, $\alpha_t^n$:
        $TN_t = \sum_{i:h_t(x_i)<0, y_i<0} D_t(i)h_t(x_i)$;
        $FN_t = \sum_{i:h_t(x_i)<0, y_i>0} D_t(i)h_t(x_i)$
        $\alpha_t^n = 1/2\, ln\,(TN_t/FN_t)$      (7)
  iv. Update weights: For positive predictions ($h_t(x_i) \geq 0$),
        $D_{t+1}(i) = D_t(i)exp(-\alpha_t^p y_i h_t(x_i))/Z_t$,      (8)
      For negative predictions ($h_t(x_i) < 0$),
        $D_{t+1}(i) = D_t(i)exp(-\alpha_t^n y_i h_t(x_i))/Z_t$,      (9)
      where $Z_t$ is chosen such that $\sum D_{t+1}(i) = 1$.
endfor
Final Model:
        $H(x) = sgn(g(x))$,
where
        $g(x) = \left(\sum_{t:h_t(x)\geq 0} \alpha_t^p h_t(x) + \sum_{t:h_t(x)<0} \alpha_t^n h_t(x)\right)$      (10)
```

Figure 2. *RareBoost-1 Algorithm: The difference from AdaBoost is the use of different importance weights for positive and negative predictions*

primary ability to achieve better recall stems from its use of default model. We explain later in section 5 as to why this is not a good strategy.

Our enhancement of SLIPPER, called RareBoost-2, is described in Figure 3. The primary difference is that we build models for both the classes in every iteration and do not use default model[3].

Due to space constraints, we refer reader to [6] for the derivation of the choice of α_t. It follows the guideline in SLIPPER's paper [2]. The key idea is again to choose α_t to minimize Z_t in every iteration, to minimize the training error. The choice between C's and NC's model is made based on whichever minimizes the corresponding Z_t value.

4 Cost-Sensitive Boosting Algorithms

The algorithms described so far use the same weight update factor for true and false predictions of a given kind. RareBoost-1 and RareBoost-2 enhance AdaBoost and SLIPPER to use different factors across positive and negative predictions. However, most generally, one can use different factors for each type: TP, FP, TN, and FN. Cost-sensitive algorithms take a step towards this. The AdaCost algorithm [4] modifies AdaBoost's weight update equation 2 to

$$D_{t+1}(i) = (D_t(i) exp(-\alpha_t y_i h_t(x_i) \beta_{sgn(h_t(x_i) y_i)}))/Z_t.$$

where $\alpha_t = 1/2\ ln((1+r_t)/(1-r_t))$ and $r_t = \sum_i D_t(i) exp(-y_i h_t(x_i) \beta_{sgn(h_t(x_i) y_i)})$. The AdaCost paper proves a general guideline for choosing the multiplying factors β_+ and β_-, that $0 \leq \beta_+ \leq \beta_- \leq 1$. We have chosen their recommended setting of $\beta_{TP} = 0.5 - 0.5\ cost(TP)/f = 0.5$, $\beta_{TN} = 0.5 - 0.5\ cost(TN)/f = 0.5$, $\beta_{FP} = 0.5 + 0.5\ cost(FP)/f = 0.5(f+1)/f$, $\beta_{FN} = 0.5 + 0.5\ cost(FN)/f = 1.0$. f is an input parameter[4]. β_+ and β_- satisfy the required constraints, for $f \geq 1$. Using these values, the expanded weight update formulae look like

$$\begin{aligned} D_{t+1}(i) &= D_t(i) exp(-0.5\ \alpha_t h_t(x_i)), \text{for TP, TN} \\ &= D_t(i) exp(0.5\ \alpha_t h_t(x_i)(f+1)/f), \text{for FP} \\ &= D_t(i) exp(\alpha_t h_t(x_i)), \text{for FN} \end{aligned}$$

Two other variations [10] of cost-sensitive algorithms are CSB1, that does not use any α_t factor (or

```
Given: $\mathcal{T}$, M.
Initialize weights $D_1(i) = 1/N$.
for $t = 1 \cdots M$
  i. Learn weak model for $y_i = +1$ (class C) examples,
     $h_t^{+1} : x_i \to \{+1, 0\}$, using $D_t$.
  ii. Learn weak model for $y_i = -1$ (class NC) examples,
     $h_t^{-1} : x_i \to \{-1, 0\}$, using $D_t$.
  iii. Evaluate C's model and NC's model:
     $$TP_t = \sum_{i:y_i>0, h_t^{+1}(x_i)>0} D_t(i);$$
     $$FP_t = \sum_{i:y_i<0, h_t^{+1}(x_i)>0} D_t(i)$$
     $$TN_t = \sum_{i:y_i<0, h_t^{-1}(x_i)<0} D_t(i);$$
     $$FN_t = \sum_{i:y_i>0, h_t^{-1}(x_i)<0} D_t(i)$$
  iv. Choose Model and Compute importance weight, $\alpha_t$:
     if( $(1 - (TP_t - FP_t)^2) < (1 - (TN_t - FN_t)^2)$ ) then,
     Choose C's Model, by setting $h_t = h_t^{+1}$:
        $$\alpha_t = 0.5\ ln\,(TP_t/FP_t) \qquad (11)$$
     else Choose NC's Model by setting $h_t = h_t^{-1}$:
        $$\alpha_t = -0.5\ ln\,(TN_t/FN_t) \qquad (12)$$
     endif
  v. Update Weights:
     $$D_{t+1}(i) = D_t(i) exp(-\alpha_t y_i)/Z_t; h_t(x_i) \neq 0, \qquad (13)$$
     where $Z_t$ is chosen such that $\sum D_{t+1}(i) = 1$.
endfor
Final Model:
     $$H(x) = sign\left(\sum_{t:x \text{ satisfies } h_t} \alpha_t\right)$$
```

Figure 3. *RareBoost-2 Algorithm: The difference from SLIPPER algorithm is the choice between C's model and NC's model, and absence of Default Model*

[3]Algorithm of Fig. 3 can be made equivalent to SLIPPER by replacing model for NC (resp. C) with a default model predicting all examples as C (resp. NC), when the focus is on class C (resp. NC). The values TN_t and FN_t (resp. TP_t and FP_t) will be replaced by $\sum_{i:y_i>0} D_t(i)$ and $\sum_{i:y_i<0} D_t(i)$ (resp. $\sum_{i:y_i<0} D_t(i)$ and $\sum_{i:y_i>0} D_t(i)$)

[4]The cost of true predictions (positive and negative) is assumed to be $cost(TP) = cost(TN) = 0$, cost of false positives is fixed at $cost(FP) = 1$, and cost of false negative is $cost(FN) = f$.

$\alpha_t = 1$), CSB2, that uses same α_t as computed by AdaBoost. The weight update formula for CSB1 is $D_{t+1}(i) = (D_t(i)C_{sgn(h_t(x_i)y_i)}exp(-y_ih_t(x_i))/Z_t$, and that for CSB2 is $D_{t+1}(i) = (D_t(i)C_{sgn(h_t(x_i)y_i)}exp(-\alpha_ty_ih_t(x_i))/Z_t$. The parameters C_+ and C_- are defined as $C_+ = 1$, and $C_- = cost(y_i, h_t(x_i))$. Using the same cost matrix as that for AdaCost, this is equivalent to $C_{TP} = 1$, $C_{TN} = 1$, $C_{FP} = 1$, and $C_{FN} = f$.

5 Comparing All Algorithms

In this section, we analyze all the described algorithms from the perspective of how they update the weights on four types of examples: TP, FP, FN, and FN. To simplify analysis, we assume, without loss of generality, that each model generates binary decision; i.e., we ignore confidence-rating of a prediction. It is in general difficult to assess how the weight of a given training example will change over all the iterations, because of the cumulative effect, and one example may switch its role among TP and FN or FP and TN from iteration to iteration. Using the fact that all the algorithms treat all examples of one kind equally (i.e. one TP is treated same as other TP) in an iteration, we decide to infer the effect by observing how aggregate weights of all example types change from iteration t to $t + 1$. Table 1 summarizes the effect for all algorithms.

Here is a brief reasoning of why the weight update factors are important. All algorithms try to concentrate on FP and FN examples by boosting their weights, and suppressing weights of TP and TN. In the next iteration, a model geared towards learning C will try to capture more FN and less FP examples. So the model tries to convert more FN's into TP's, thus increasing recall. However, if the weights of TN are reduced significantly (as compared to FN's), C's model may capture some of TN's, thus losing precision. Similarly, a model geared for NC will try to capture more FP and as little of FN as possible, thus trying to convert more FP's into TN's, thus improving precision. However, it might capture more TP's if weight on the TP's is reduced to low levels, thus losing recall.

AdaBoost vs. RareBoost-1:
AdaBoost gives equal importance to both types of false predictions. RareBoost-1 scales FP examples in proportion to how well they are distinguished in an iteration from TP examples, and FN in the proportion of how well they are distinguished from TN. The essential effect of weight update in AdaBoost is to stratify the sum of weights on *all* true predictions against *all* false predictions. RareBoost-1 stratifies true positives against false positives and true negatives against false negatives. Traditional weak learners learn effective models when the class proportion is balanced. The separate stratification of positive and negative predictions gives the weak learner a better chance at distinguishing each type. Thus, we expect RareBoost-1 to achieve better recall and precision as compared to AdaBoost.

SLIPPER vs. RareBoost-2:
In rare class context, when it is difficult to build a good model for C, SLIPPER's primary ability to aim for recall comes due to its default model. From Table 1, with default model, effect of weight update is to equalize the weights on both classes. From our experience with rare classes in [5], such stratification usually improves recall at the cost of precision. If good models for C cannot be found to distinguish FP examples from TP examples, then the overall balance may suffer. Instead, RareBoost-2, at the cost of building models for both C and NC, specifically targets the weak learner to model either TP vs. FP, or TN vs. FN. This may help it achieve better performance than SLIPPER.

Cost-Sensitive vs. Cost-Insensitive Algorithms:
As shown in Table 1, unlike any of the cost-insensitive algorithms, AdaCost comes close to a generic strategy of updating weights of all four types of examples differently. For values of $f > 1$, it increases weights on FNs more than any other type of examples. The net effect is that the weak learner focuses on recall in the next iteration. The FPs also get boosted more than TPs are suppressed. This allows AdaCost to focus on precision as well. But, unlike other algorithms, TP and FP are not stratified (neither are TN and FN), so increasing precision might tend to lose recall by capturing more TP examples. Maybe for higher f values this loss in recall can be regained. Usually higher recall comes at a cost of lower precision. So, it will depend on the dataset whether the recall focus helps[5]. Like AdaCost, CSB1 and CSB2 focus on FN by varying f. However, this focus is not accompanied by focus on precision, which can result in high recall at a loss of precision.

6 Results on Synthetic Datasets

In this section, we validate our qualitative analysis using some specially designed synthetic datasets. Due to space constraints, we refer reader to [6] for more detailed experiment description. Briefly, we use RIPPER0 [3] as the weak learner. We evaluate the performance of each algorithm using $F1$-measure[6] after each iteration on the test data for 100 iterations, and report the best numbers[7]. We report best result of the two variants of RareBoost-1 formed by using α_t^{p1} and α_t^{p2}. For SLIPPER, we choose best result from two

[5]Note that unlike methods purely focusing on recall by an up-front oversampling of rare class, AdaCost can cater to precision also.

[6]F_1-measure is defined as 2*R*P)/(R+P), where recall R = TP/(TP+FN) and precision P = TP/(TP+FP).

[7]Test data is generated using the same model as training data. So, monitoring performance on it essentially is same as monitoring the generalization ability of the algorithm. For the purposes of our experiments, test data can be thought of as validation data.

AdaBoost	**RareBoost-1**
$TP_{t+1} = TP_t/\gamma$	$TP_{t+1} = TP_t/\gamma_1$
$FP_{t+1} = FP_t * \gamma$	$FP_{t+1} = FP_t * \gamma_1$
$TN_{t+1} = TN_t/\gamma$	$TN_{t+1} = TN_t/\gamma_2$
$FN_{t+1} = FN_t * \gamma$	$FN_{t+1} = FN_t * \gamma_2$
$\gamma = e^{\alpha_t} = \sqrt{(TP_t + TN_t)/(FP_t + FN_t)}$	$\gamma_1 = e^{\alpha_t^p} = \sqrt{TP_t/FP_t}, \gamma_2 = e^{\alpha_t^n} = \sqrt{TN_t/FN_t}$
Effect: $TP_{t+1} + TN_{t+1} = FP_{t+1} + FN_{t+1}$	Effect: $TP_{t+1} = FP_{t+1}$ & $TN_{t+1} = FN_{t+1}$

SLIPPER-C	**SLIPPER-NC**
If C's Model is chosen in t^{th} iteration,	If NC's Model is chosen in t^{th} iteration
$TP_{t+1} = TP_t/\gamma_1$	$TP_{t+1} = TP_t$
$FP_{t+1} = FP_t * \gamma_1$	$FP_{t+1} = FP_t$
$TN_{t+1} = TN_t$	$TN_{t+1} = TN_t/\gamma_2$
$FN_{t+1} = FN_t$	$FN_{t+1} = FN_t * \gamma_2$
$\gamma_1 = e^{\alpha_t} = \sqrt{TP_t/FP_t}$	$\gamma_2 = e^{\alpha_t} = \sqrt{TN_t/FN_t}$
Effect: $TP_{t+1} = FP_{t+1}$	Effect: $TN_{t+1} = FN_{t+1}$
OR	OR
If Default Model is chosen in t^{th} iteration,	If Default Model is chosen in t^{th} iteration
$TP_{t+1} = TP_t/\gamma_3$	$TP_{t+1} = TP_t/\gamma_3$
$FP_{t+1} = FP_t * \gamma_3$	$FP_{t+1} = FP_t * \gamma_3$
$TN_{t+1} = TN_t/\gamma_3$	$TN_{t+1} = TN_t/\gamma_3$
$FN_{t+1} = FN_t * \gamma_3$	$FN_{t+1} = FN_t * \gamma_3$
$\gamma_3 = e^{\alpha_t} = \sqrt{(TP_t + FP_t)/(TN_t + FN_t)}$	$\gamma_3 = e^{\alpha_t} = \sqrt{(TN_t + FN_t)/(TP_t + FP_t)}$
Effect: $TP_{t+1} + FN_{t+1} = TN_{t+1} + FP_{t+1}$	Effect: $TP_{t+1} + FN_{t+1} = TN_{t+1} + FP_{t+1}$

RareBoost-2	
If C's Model is chosen in t^{th} iteration OR	If NC's Model is chosen in t^{th} iteration
$TP_{t+1} = TP_t/\gamma_1$	$TP_{t+1} = TP_t$
$FP_{t+1} = FP_t * \gamma_1$	$FP_{t+1} = FP_t$
$TN_{t+1} = TN_t$	$TN_{t+1} = TN_t/\gamma_2$
$FN_{t+1} = FN_t$	$FN_{t+1} = FN_t * \gamma_2$
$\gamma_1 = e^{\alpha_t} = \sqrt{TP_t/FP_t}$	$\gamma_2 = e^{-\alpha_t} = \sqrt{TN_t/FN_t}$
Effect: $TP_{t+1} = FP_{t+1}$	Effect: $TN_{t+1} = FN_{t+1}$

AdaCost	**CSB1** and **CSB2**
$TP_{t+1} = TP_t/\gamma$	$TP_{t+1} = TP_t/\gamma$
$FP_{t+1} = FP_t * \gamma^{(f+1)/f}$	$FP_{t+1} = FP_t * \gamma$
$TN_{t+1} = TN_t/\gamma$	$TN_{t+1} = TN_t/\gamma$
$FN_{t+1} = FN_t * \gamma^2$	$FN_{t+1} = FN_t * f * \gamma$
$\gamma = e^{0.5\alpha_t}$	$\gamma = e^1$ for CSB1, and $\gamma = e^{\alpha_t}$ for CSB2
Effect1: Increase weights of FN more than FP	
Effect2: Decrease weights of TP or TN by a smaller factor than FN or FP or both	

Table 1. *Comparing the effect of weight updates in each of the algorithms. This analysis assumes the range of h_t's prediction to be binary. The entities TP_t, etc. are aggregate weights of examples of that type.*

variants, one with focus on C and other with focus on NC. For CSB1 and AdaCost, we use f=1, 2, and 5, and report the best number. For f=1, CSB2 is equivalent to AdaBoost, so we report its best result among f=2 and f=5. Last point to note is that, we used only the binary valued models (i.e. ignored confidence-rated predictions). This is done so as to be consistent with our qualitative analysis assumptions[8].

Datasets Without Any Attributes Correlations :
The model has three types of attributes. The records of classes C and NC are divided into multiple subclasses. For each attribute, Figure 4 shows the histogram distribution of subclasses over the range of its values. Each attribute distinguishes one subclass of C and/or one subclass of NC. For example, C_a can be distinguished by a rule capturing the range of values in $ACOM_a$ where C_a peaks.

In our experiments, the proportion of the target class C with respect to NC was fixed at 5% in all these datasets. We varied four peak-width parameters, defined in Figure 4. These parameters let us control the recall and precision obtainable for a dataset.

All the algorithms perform equally well when all the peaks are very sharp (snc1). Even when the C_a and C_b peaks widen (snc2), all algorithms, except SLIPPER, are competitive, because good models for the majority class NC can still be learned. The reason for SLIPPER's performance deterioration can be attributed to its default model selection

[8]This assumption affects all but SLIPPER and RareBoost-2, where the confidence rating is embedded in the α_t factor.

mechanism. As the NC_b peaks are made wider (snc3), all the algorithms suffer a dramatic loss in F_1-measure. This can be attributed to the fact that NC_b is the majority subclass of NC, and wider WNC_b makes it difficult to remove FPs due to it without removing TPs of C. And because of the rarity of C and the nature of the data model, it is difficult to learn a good model to regain true positives. RareBoost-1 is achieving higher precision while maintaining the recall at the level of AdaBoost. RareBoost-2 is significantly better than SLIPPER for both recall and precision. Although AdaCost is achieving the best F_1-measure on the basis of boosting false negatives significantly (best f is 5), in the process it suffers from a very poor precision (lowest precision). In fact, it can be claimed that recall and precision numbers of RareBoost-2 are better balanced. As the NC_a peaks are made wider (snc4), the loss in F_1-measure is not as dramatic as in snc3, because NC_a peaks are not captured when good signatures of C_a are learned. AdaCost again wins here on F_1-measure because of its emphasis on recall (again best f is 5). Its precision, however, is again much worse as compared to that of RareBoost-1. Here, RareBoost-1 can be said to achieve better balance. The last dataset (snc5) has a mixture of wide C_b peaks and wide NC_a peaks. This time AdaCost is able to capture good recall as well as good precision (for $f = 2$) among all. This dataset may be representing the scenario where the cost-sensitivity is required.

Datasets With Attribute Correlations:
We use the data generating model that was first used by us in [5]. It is described in detail therein. Briefly, it is similar to previous model, except for the correlations among signatures of attributes of type $ACOM_a$. It also has some categorical attributes of type ANC_b and AC_b. We believe that this model is fairly general and complex. The results on this generic model are given in Table 3. The dataset model is kept fixed (the parameters are $WC_a = WC_b = WNC_a = WNC_b = 2.0\%$), and only the target class proportion (Cfrc) in training (and test) data is varied. As Cfrc increases, it becomes easier to achieve better recall and precision. RareBoost-1 and RareBoost-2 outperform AdaBoost and SLIPPER in all cases. More importantly, the performance gap increases as Cfrc decreases. AdaCost is the most competitive cost-sensitive technique. For 2.9% and 5.7% datasets, it achieves its best numbers when $f = 2$. For these datasets, RareBoost variants are either closer in performance to AdaCost (2.9%) or they are better than AdaCost (5.7%). For higher Cfrc values, AdaCost achieves its best only for $f = 1$. RareBoost-2 beats it for 13.1% dataset. As f increases on these datasets, AdaCost's precision shows significant degradation [6]. These results indicate that overemphasizing recall can result in a significant loss in precision. In fact, precisely in these kind of situations, cost-insensitive algorithms and our proposed RareBoost variants can perform better than cost-sensitive algorithms.

7 Concluding Remarks

The outcome of our study is a critical qualitative and empirical comparison of three representative categories of most popular boosting variants in the context of rare classes, and two enhanced versions of boosting algorithms that are shown to be better especially in certain situations involving rare classes. The guideline that emerges from this is that weight update mechanisms that resemble to those of RareBoost variants or AdaCost are required for handling rare classes using boosting algorithms. Further generalization of the weight update mechanisms should aim for theoretically arriving at different optimal update factors for four types of examples: true positives, false positives, true negatives, and false negatives.

References

[1] P. Chan and S. Stolfo. Towards scalable learning with non-uniform class and cost distributions: A case study in credit card fraud detection. In *Proc. of Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98)*, pages 164–168, New York City, 1998.

[2] W. Cohen and Y. Singer. A simple, fast, and effective rule learner. In *Proc. of Annual Conference of American Association for Artificial Intelligence*, pages 335–342, 1999.

[3] W. W. Cohen. Fast effective rule induction. In *Proc. of Twelfth International Conference on Machine Learning*, Lake Tahoe, California, 1995.

[4] W. Fan, S. J. Stolfo, J. Zhang, and P. K. Chan. AdaCost: Misclassification cost-sensitive boosting. In *Proc. of Sixth International Conference on Machine Learning (ICML-99)*, Bled, Slovenia, 1999.

[5] M. V. Joshi, R. C. Agarwal, and V. Kumar. Mining needles in a haystack: Classifying rare classes via two-phase rule induction. In *Proc. of ACM SIGMOD Conference*, pages 91–102, Santa Barbara, CA, 2001.

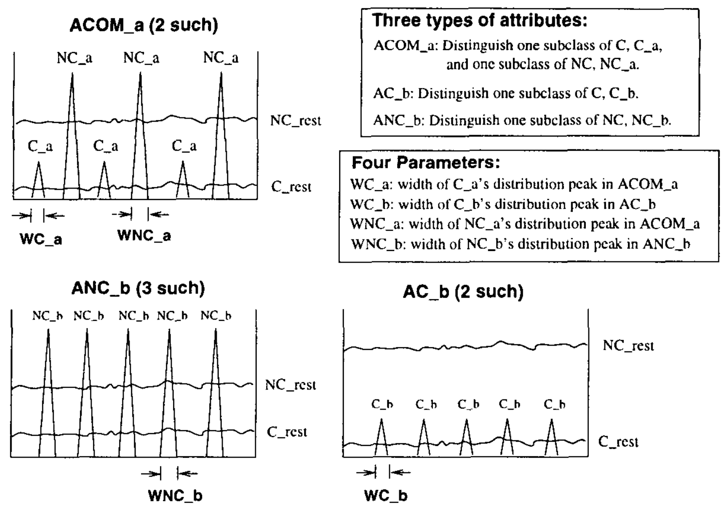
[6] M. V. Joshi, V. Kumar, and R. C. Agarwal. Evaluating boosting algorithms to classify rare classes: Comparison and improvements. Technical Report RC-22147, IBM Research Division, August 2001.

[7] R. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.

[8] R. E. Schapire. Theoretical views of boosting. In *Proc. 4th European Conference on Computational Learning Theory*, volume 1572, pages 1–10. Springer-Verlag, 1999.

[9] F. Sebastiani, A. Sperduti, and N. Valdambrini. An improved boosting algorithm and its application to text categorization. In *Proc. 9th International ACM Conf. on Information and Knowledge Management (CIKM-00)*, pages 78–85, New York, USA, 2000.

[10] K. M. Ting. A comparative study of cost-sensitive boosting algorithms. In *Proc. of 17th International Conf. on Machine Learning*, pages 983–990, Stanford University, CA, 2000.



Three types of attributes:

ACOM_a: Distinguish one subclass of C, C_a, and one subclass of NC, NC_a.

AC_b: Distinguish one subclass of C, C_b.

ANC_b: Distinguish one subclass of NC, NC_b.

Four Parameters:

WC_a: width of C_a's distribution peak in ACOM_a
WC_b: width of C_b's distribution peak in AC_b
WNC_a: width of NC_a's distribution peak in ACOM_a
WNC_b: width of NC_b's distribution peak in ANC_b

Figure 4. *Description of the model generating datasets with no correlations among attributes.*

Dataset	WC_a	WC_b	WNC_a	WNC_b
snc1	0.4	0.4	0.4	0.4
snc2	1.6	1.6	0.4	0.4
snc3	0.4	0.4	0.4	2.4
snc4	0.4	0.4	2.4	0.4
snc5	0.2	2.4	2.4	0.2

DSet		ABst	RB-1	SLIP	RB-2	CSB1	CSB2	ACst
snc1	R	77.04	77.04	81.13	74.93	77.04	77.04	77.04
	P	94.19	94.19	95.94	98.61	94.19	94.19	94.19
	F	84.76	84.76	87.92	85.16	84.76	84.76	84.76
snc2	R	76.78	76.78	69.39	76.78	76.78	76.78	77.31
	P	95.41	95.41	85.53	95.41	95.41	95.41	95.75
	F	85.09	85.09	76.62	85.09	85.09	85.09	85.55
snc3	R	41.03	41.16	35.09	49.21	43.14	43.14	79.02
	P	59.58	65.00	59.11	62.37	67.84	67.84	51.73
	F	48.59	51.10	44.04	55.01	52.74	52.74	62.53
snc4	R	54.75	58.71	47.63	56.99	52.64	38.13	69.53
	P	79.35	90.45	94.75	82.60	55.65	51.89	76.38
	F	64.79	71.20	63.39	67.45	54.10	43.95	72.79
snc5	R	47.23	51.58	43.14	53.96	56.46	55.67	63.19
	P	76.50	87.87	88.14	89.50	50.53	77.86	92.47
	F	58.40	65.00	57.93	67.33	53.33	64.92	75.08

Table 2. *Table on the Left: Specific Datasets Generated with model of Figure 4. Peak widths are given as a percentage of the total range of the attribute. Table on the Right: Results on datasets with no attribute correlations. Notation: R:recall for C (in %), P: precision for C (in %), F: F_1-measure (in %), ABst: AdaBoost, RB-1: RareBoost-1, SLIP: SLIPPER, RB-2: RareBoost-2, ACst: AdaCost.*

Cfrc		ABst	RB-1	SLIP	RB-2	CSB1	CSB2	ACst
2.9%	R	50.93	57.20	64.80	56.27	57.07	84.00	67.20
	P	71.27	81.25	57.51	82.10	60.03	44.74	68.39
	F	59.41	67.14	60.94	66.77	58.51	58.39	67.79
5.7%	R	63.20	68.67	63.60	73.20	67.07	84.13	76.00
	P	80.61	83.74	71.19	81.70	72.90	46.43	72.15
	F	70.85	75.46	67.18	77.22	69.86	59.84	74.03
13.1%	R	76.53	77.47	70.00	79.73	77.07	88.67	74.80
	P	80.62	83.36	78.59	86.04	72.70	58.18	85.65
	F	78.52	80.30	74.05	82.77	74.82	70.26	79.86
23.1%	R	78.27	84.40	80.80	80.80	86.13	96.13	85.07
	P	83.74	85.43	82.56	86.45	76.81	68.60	84.39
	F	80.91	84.91	81.67	83.53	81.21	80.07	84.73

Table 3. *Results on datasets correlations between attributes. Notation: Cfrc: Proportion of the target class C in the training dataset. R:recall for C (in %), P: precision for C (in %), F: F_1-measure (in %), ABst: AdaBoost, RB-1: RareBoost-1, SLIP: SLIPPER, RB-2: RareBoost-2, ACst: AdaCost.*