RC22147 (W0108-018) August 13, 2001
Computer Science

IBM Research Report

Evaluating Boosting Algorithms to Classify Rare Classes: Comparison and Improvements

Mahesh V. Joshi, Vipin Kumar, Ramesh C. Agarwal
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

LIMITED DISTRIBUTION NOTICE: This report has been submitted for publication outside of IBM and will probably be copyrighted if accepted for publication. Ithas been issued as a Research Report for early dissemination of its contents. In view of the transfer of copyright to the outside publisher, its distributionoutside of IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or legally obtained copies of the article (e.g, payment of royalties). Copies may be requested from IBM T. J. Watson Research Center , P. O. Box 218, Yorktown Heights, NY 10598 USA (email: reports@us.ibm.com). Some reports are available on the internet at http://domino.watson.ibm.com/library/CyberDig.nsf/home

Evaluating Boosting Algorithms to Classify Rare Classes: Comparison and Improvements

Mahesh V. Joshi* Vipin Kumar† Ramesh C. Agarwal‡

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 for their ability to achieve high recall and high precision for a given rare class in the context of binary classification. We explain all these algorithms from the single viewpoint of how their weight updating mechanisms work at each iteration, and discuss their possible effect on emphasizing recall or precision. 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 of the algorithms, and also indicate that our enhancements yield extra capability to their predecessor algorithms for achieving better balance between recall and precision.

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 clickstream 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. Another example can be cited from network intrusion detection domain, where the proportion of number of attack connections to a server as compared to normal connections is very low, but building models for attacks is very crucial. Some work has started to emerge in building descriptive models for the rare events [1, 7]. 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 [5, 9, 2, 13, 4]. They have been analyzed for their effectiveness [8, 10, 6], and they have been adapted to special tasks [11, 12]. Despite of this abundant work on boosting, no work has dealt directly with evaluating boosting algorithms in the context of mining rare events.

Most existing boosting algorithms are designed towards achieving better classification accuracy. However, from the context of rare classes, accuracy is not a sufficient evaluation metric. For example,

*IBM T.J. Watson Research Center and University of Minnesota, Minneapolis. Contact Author. Address: IBM T. J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598. Email: joshim@us.ibm.com. Phone: +1-914-784-6158. Fax: +1-914-784-7455

†Department of Computer Science, University of Minnesota, 200 Union St. SE, Minneapolis, MN 55455. (kumar@cs.umn.edu)

‡IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120. (ragarwal@us.ibm.com).

in a problem where rare class is represented in only 1% of the training data, a naive strategy of predicting the prevalent class label for every example, can achieve an acceptable accuracy of 99%. What matters for rare classes is both recall and precision[1]. Often, achieving very high recall and precision levels are conflicting goals, more so for the rare classes whose signatures are not very pure. In such cases, achieving a better balance[2] between recall and precision becomes crucial. In this paper, our goal is to critically compare some representative categories of boosting algorithms for their ability to achieve such balance.

Boosting algorithms work in iterations, each time learning a weak classifier model[3] on a different weighted distribution of training records. After each iteration, the weights of the examples are updated. The main intuitive idea behind a popular original boosting algorithm, AdaBoost[5], is to increase the weights of the incorrectly classified examples and decrease the weights of the correctly classified examples. This forces the classifier algorithm to focus more on the incorrectly classified examples in the next iteration. The algorithm usually stops after a few pre-specified iterations, or it can be stopped based on some measured quality of the algorithm (such as error rate of latest learned classifier getting close to random guessing). The set of classifiers obtained in these iterations are all used together for the final prediction on an unseen case. Each classifier gets to vote, which is usually monotonic with its accuracy. The class that gets the 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 first categorize the algorithms in three different categories: the algorithms that make true or false decision for every example during training, the algorithms that choose to abstain from making any decision on some examples during training, and finally, the algorithms that take misclassification costs into account. One of the contributions of this paper is that, we project some representative algorithms from these categories in the same light, in order to bring out their differences in 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 C.

The algorithms of the first category, to which AdaBoost [5] belongs, treat all correctly and incorrectly predicted examples equally, by increasing or decreasing the weights of false and true predictions using the same proportion. This has an effect of trying to achieve recall and precision objectives simultaneously for both the classes (C and NC). However, we believe that this simultaneous focus may not perform very well for achieving a better recall and precision for the rare class. Here is an intuitive reason why. In order to achieve better recall, one needs to learn better models for distinguishing FN from TN, and in order to achieve better precision, one needs to learn better models to distinguish FP from TP. In other words, we should give different treatment to FN and TN than

[1]Recall is the fraction of total examples of a class that are predicted to belong to that class by the classifier. Precision is the fraction of total examples predicted by the classifier to be of a class that indeed belong to that class.

[2]Balance between recall and precision is usually defined as a measure that achieves high value when both recall and precision are high, and is dominated by the smaller number. One such measure is F_1-measure [14].

[3]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 give to FP and TP, especially so when the one of the classes is under-represented. Based on this theme, we propose an enhanced algorithm which updates weights of TP and FP differently from weights of TN and FN. We derive the weight updating formulae theoretically.

One more point that is implicit in this theme is that it is important to learn models focused on reducing both types of false decisions. We use this point in proposing an enhancement for the second category of algorithms, that abstain from making any decisions on some examples. The weights are updated only on the examples where some decision is made, others are left untouched. A well-known algorithm of this category is SLIPPER [2]. However, the problem with SLIPPER is that it keeps its focus on one class in all the iterations. Thus, if we choose C, only weights of TP and FP get updated. However, a good model for C may not be obtainable. In such situations, SLIPPER chooses a default model that predicts everything to be of class in focus. Studying the weight update mechanism shows us that the effect of this default choice is to equalize the sum of weights on both the classes. However, from our experience with rare classes [7], such stratification by oversampling the minority class achieves recall at the cost of precision, and in case where it becomes difficult to regain this precision (i.e. very good C models can not be built), the recall-precision balance suffers. In order to alleviate this problem, we propose an enhanced algorithm, that dynamically switches the focus from C to NC, and does not require a default model. The decision to choose a C or NC model after every iteration is based on the same principle as used by SLIPPER. This principle is theoretically justified.

The third category of techniques belongs to cost-sensitive boosting algorithms such as CSB1, CSB2 [13] and AdaCost [4]. These algorithms cater to different misclassification costs for making a false positive prediction versus a false negative prediction. Usually, rare classes are handled by giving higher misclassification costs to FN examples. The primary difference between these algorithms from the previous two categories is that weights of TP (TN) are decreased using a factor different from the one used to increase the weights of FP (FN). Our study will show 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, its over-emphasis on recall may sometimes lead to a poorer precision.

We validate our qualitative study of the effect of weight update mechanisms using some synthetic datasets, that are specially designed for studying rare classes. We show that our proposed enhancements in first two categories outperform their respective predecessors in achieving better recall-precision balance. We also show that one of our proposed enhancements has the ability to even outperform the most competitive cost-sensitive algorithm, AdaCost, in some situations.

Our Contributions in this paper:

- A first comparative study of the boosting algorithms in the context of building classification models for rare classes. We consider all the representative variations of the boosting methods.
- A common perspective for analyzing the weight update mechanisms of the algorithms, and a qualitative and empirical analysis of their effect on recall and precision.
- Two new enhanced boosting algorithms for two cost-insensitive categories of algorithms. We show that these algorithms bring an extra capability that is useful in building better models for rare classes.

The rest of the paper is organized as follows. In the following sections 2 and 3, we describe two representative algorithms from two cost-insensitive categories, and propose our enhancement of each. Theoretical justification is given for the choice of weight update parameters in our proposed algorithms. In section 4, we describe cost-sensitive boosting algorithms. Later in section 5, we put

all the algorithms in same perspective to compare and analyze the effect of their weight updating strategies on recall and precision. Section 6 gives results on synthetic datasets. Section 7 concludes the paper with some remarks.

2 Boosting Algorithms that Do Not Abstain

In this section, we describe algorithms that make a true or false decision on every example. We start by giving the structure of the AdaBoost [9] algorithm. In all the algorithms that we describe, the goal is to build a classification model for the class attribute in terms of other attributes, using examples in the training data. Each boosting algorithm yields an ensemble of models that will be used together to make a prediction on any unseen case. The way to combine votes from each model is described in the last step of each algorithm.

Here is a quick reference to the common notation used in all the algorithms.

Notation:

$\mathcal{T}$	: Training Data.
x_i	: Attribute vector from a domain $\mathcal{X}$.
y_i	: Class attribute from a domain $\mathcal{Y} = \{-1, +1\}$, we use $+1$ to denote rare class of interest, C, and -1 to denote other class NC.
(x_i, y_i)	: i^{th} training example in $\mathcal{T}, i = 1 \cdots N$.
M	: maximum number of boosting iterations
t	: Boosting iteration number ($t \leq M$).
$D_t(i)$	: Weight on i^{th} example at the beginning of t^{th} iteration
h_t	: Model generated in the t^{th} iteration; $h_t : \mathcal{X} \rightarrow [-1, +1]$. This usually yields a signed number, where sign corresponds to the predicted class and number corresponds to the confidence in the prediction.
α_t	: Importance weight (vote) assigned to t^{th} model
H	: Final model formed by the ensemble of models; $H : \mathcal{X} \rightarrow \{-1, +1\}$.

2.1 AdaBoost Algorithm [9]

This algorithm is described in Figure 1. We briefly discuss its features here. The strength associated with each classifier, α_t is proportional to the accuracy of the classifier, in some sense (true when h_t's range is {-1,+1}). The weak classifier must yield an accuracy of greater than 50%, in order to satisfy a crucial condition $\alpha_t > 0$. When this condition is satisfied, the algorithm increases the weights of the false predictions (both true and false) and decreases the weights of the true predictions (true and false), at the end of every iteration. The choice of α_t is derived so as to minimize the sum of all the weights before the beginning of t^{th} iteration. Minimizing this sum greedily in each iteration minimizes an upper bound on the training error. Note that the algorithm is treating both classes equally.

2.2 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. The intuition is that, in order to achieve a good recall for the class C, we must steer the algorithm towards building better models to distinguish FN from TN. Similarly, in order to achieve better precision for C, we must allow for better distinction between TP and FP. We choose to treat positives differently from

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, α_t:

$$r_t = \sum_{i=1}^{N} D_t(i) h_t(x_i) y_i, \qquad \alpha_t = \frac{1}{2} ln \left(\frac{1 + r_t}{1 - r_t} \right) \tag{1}$$

iii. Update Weights:

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

where Z_t is chosen such that $\sum D_{t+1}(i) = 1$.

endfor
Final Model:

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

Figure 1: *AdaBoost Algorithm [9]*

negatives with the hope that it will allow the algorithm to *focus on both recall and precision equally.* This is achieved by updating weights of positives and negatives using different factors. If the model makes every prediction (C or NC) with an accuracy of greater than 50%; i.e., if $TP > FP$ and $TN > FN$, then the algorithm will decrease the weights of TP and TN examples and increase the weights of FP and FN.

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

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

After showing this formula, we can follow [9] to say that minimizing the classification error can be achieved by minimizing Z_t.

Next step is to show that the choice of α_t^p and α_t^n as given in the algorithm can indeed minimize Z_t.

Z_t is the sum of weights after each iteration t. Using notation $u_i = y_i h_t(x_i)$,

$$Z_t = Z_t^p + Z_t^n; Z_t^p = \sum_{i:h_t(x_i)\geq 0} D_t(i) exp(-\alpha_t^p u_i), \; Z_t^n = \sum_{i:h_t(x_i)<0} D_t(i) exp(-\alpha_t^n u_i)$$

Thus Z_t can be minimized by minimizing each of the terms, Z_t^p and Z_t^n. We will derive expressions for α_t^p by minimizing Z_t^p. Derivation for α_t^n is symmetric. If we were to use the same linear upper bound as given in [9],

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

then minimizing this bound by differentiating it w.r.t. α_t^p, will yield

$$\alpha_t^{p1} = \frac{1}{2} ln \left(\frac{1 + \sum_{i:h_t(x_i)\geq 0} D_t(i) y_i h_t(x_i)}{1 - \sum_{i:h_t(x_i)\geq 0} D_t(i) y_i h_t(x_i)} \right) = \frac{1}{2} ln \left(\frac{1 + TP_t - FP_t}{1 - TP_t + FP_t} \right) \tag{4}$$

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:

$$\alpha_t^{p2} = (1/2)\ ln(TP_t/FP_t). \tag{5}$$

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

$$Z_t^{p1} = TP_t \frac{\sqrt{\delta + 2\ FP_t}}{\sqrt{\delta + 2\ TP_t}} + FP_t \frac{\sqrt{\delta + 2\ TP_t}}{\sqrt{\delta + 2\ FP_t}};\ \text{where } \delta = (1 - TP_t - FP_t) \geq 0.$$

The following expression can be easily derived using above two formulae.

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

As can be seen from above, $Z_t^{p2} \leq Z_t^{p1}$. The algorithm in Figure 2 uses $\alpha_t^p = \alpha_t^{p2}$. ♠

3 Boosting Algorithms that Abstain

In this section, we describe algorithms that use base classifiers which may abstain from making any decision on some training examples. Possibility of such algorithms is pointed out in [9], and an actual such algorithm, SLIPPER, was proposed in [2]. We review this algorithm briefly and suggest a need to modify it for rare classes. The modified algorithm is also presented later.

3.1 SLIPPER Algorithm

SLIPPER algorithm is described in figure 3. The algorithm is primarily different from AdaBoost, in its way of concentrating on only the examples for which the base classifier makes some true or false decision. The original SLIPPER algorithm uses a very simple base classifier consisting of only one rule. We however allow it to use a model with many rules. The key feature of the algorithm is that it focuses on building rules for only one of the classes in all the iterations. Their paper suggests use of $y_i = +1$ examples, but the algorithm can be easily modified to build models for $y_i = -1$ examples. When a good model for the focused class cannot be built, SLIPPER reverts to using a default model that predicts everything to be of that class. From the recall-precision perspective, its ability to aim for precision comes from being able to build good model for C. Although this allows to aim for recall also, in the context of rare C, its primary ability to aim for recall comes due to its default model. When default model is chosen, the effect of weight update is that of stratifying the weights on two classes (this is explained in section 5). From our experience with rare classes in [7], 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. We suggest a mechanism to alleviate this problem in the next subsection.

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, α_t^p:

$$TP_t = \sum_{i:h_t(x_i)\geq 0, y_i>0} D_t(i)h_t(x_i), \quad FP_t = \sum_{i:h_t(x_i)\geq 0, y_i<0} D_t(i)h_t(x_i)$$

$$\alpha_t^p = 1/2 \; ln\,(TP_t/FP_t) \tag{6}$$

iii. Compute importance weight for negative predictions, α_t^n:

$$TN_t = \sum_{i:h_t(x_i)<0, y_i<0} D_t(i)h_t(x_i), \quad FN_t = \sum_{i:h_t(x_i)<0, y_i>0} D_t(i)h_t(x_i)$$

$$\alpha_t^n = 1/2 \; ln\,(TN_t/FN_t) \tag{7}$$

iv. Update weights: For positive predictions ($h_t(x_i) \geq 0$),

$$D_{t+1}(i) = D_t(i)exp(-\alpha_t^p y_i h_t(x_i))/Z_t, \tag{8}$$

For negative predictions ($h_t(x_i) < 0$),

$$D_{t+1}(i) = D_t(i)exp(-\alpha_t^n y_i h_t(x_i))/Z_t, \tag{9}$$

where Z_t is chosen such that $\sum D_{t+1}(i) = 1$.

endfor
Final Model:

$$H(x) = sign(g(x)), where\ g(x) = \left(\sum_{t:h_t(x)\geq 0} \alpha_t^p h_t(x) + \sum_{t:h_t(x)<0} \alpha_t^n h_t(x) \right) \tag{10}$$

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

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, $h_t : x_i \rightarrow \{+1, 0\}$, using D_t.

ii. Evaluate C's model and default model:

$$TP_t = \sum_{i:y_i>0, h_t(i)>0} D_t(i), \qquad FP_t = \sum_{i:y_i<0, h_t(i)>0} D_t(i) \tag{11}$$

$$p_t = \sum_{i:y_i>0} D_t(i), \qquad n_t = \sum_{i:y_i<0} D_t(i) \tag{12}$$

iii. Choose Model and Compute importance weight, α_t:
if($(1 - (TP_t - FP_t)^2) < (1 - (p_t - n_t)^2)$) then, Choose C's Model:

$$\alpha_t = 0.5 \; ln \, (TP_t / FP_t) \tag{13}$$

else Choose Default Model by setting $h_t(x) = +1, \; \forall x$:

$$\alpha_t = 0.5 \; ln \, (p_t / n_t) \tag{14}$$

endif

iv. Update Weights:

$$D_{t+1}(i) = D_t(i) exp(-\alpha_t y_i)/Z_t; h_t(x_i) > 0, \tag{15}$$

where Z_t is chosen such that $\sum D_{t+1}(i) = 1$.

endfor
Final Model:

$$H(x) = sign \left(\sum_{t:x \text{ satisfies } h_t} \alpha_t \right) \tag{16}$$

Figure 3: *SLIPPER Algorithm: The algorithm is presented assuming that the focus is on C ($y_i = +1$) examples. If the focus should be on NC examples, the algorithm can be simply modified by defining TP_t over examples where $y_i < 0, h_t(x_i) < 0$, FP_t over $y_i \geq 0, h_t(x_i) < 0$, TP over $y_i < 0$, and FP over $y_i \geq 0$. Other steps should be modified appropriately.*

3.2 Our Proposed Enhancement (RareBoost-2)

Our enhancement of SLIPPER, called RareBoost-2, is described in Figure 4. The primary difference is that we build models for both the classes in every iteration. In the described algorithm, we do not use default model. It can be used. However, from our experience, we have found that usually for rare classes, at every iteration, either of the C's or NC's model is better than the default model. The choice of α_t can be easily derived following the derivation given in SLIPPER's paper [2]. The key idea is again to choose α_t to minimize Z_t in every iteration, which will minimize the overall classification error. As a matter of fact, the choice between C's and NC's model is made based on whichever minimizes the corresponding Z_t value. We will describe derivation of α_t for the case of choosing NC's model. With this choice, only the weights of TN and FN examples get affected, because NC's model h_t^{-1} makes only a $h_t^{-1}(x_i) = -1$ decision or a $h_t^{-1}(x_i) = 0$ decision (which means abstaining). Thus, using equation 21,

$$\begin{aligned} Z_t &= \sum_{i:h_t(x_i)\geq 0} D_t(i) + \sum_{i:h_t(x_i)<0, y_i=+1} D_t(i)exp(\alpha_t) + \sum_{i:h_t(x_i)<0, y_i=-1} D_t(i)exp(-\alpha_t) \\ &= TP_t + FP_t + FN_t * exp(\alpha_t) + TN_t * exp(-\alpha_t) \end{aligned}$$

Differentiating Z_t w.r.t. α_t and equating it to 0, gives $\alpha_t = 0.5\ ln(TN_t/FN_t)$. The negative sign in equation 20 is there just to be able to write a single weight update formula in equation 21.

The condition that is checked in step 4 is essentially checking for Z_t value for each of the models; whichever yields smaller value is chosen.

4 Cost-Sensitive Boosting Algorithms

The algorithms described so far have one thing in common. They still use the same weight update factor to modify weights of true and false predictions of a given kind (positive or negative). RareBoost-1 and RareBoost-2 enhance AdaBoost and SLIPPER to use different factors across positive and negative predictions. However, there is a possibility of using different factors for each TP, FP, TN, and FN type of examples. Cost-sensitive algorithms effectively do this to some extent. We describe a representative algorithm, AdaCost [4], and two other algorithms given in [13].

AdaCost Algorithm [4]:

The original algorithm assumes a different misclassification cost for each example in the training data. Their algorithm assumes that the cost of a given training example remains fixed in all iterations. However, in our adaptation for rare classes, we assume a misclassification cost matrix that is pointed out in [13]. Although the cost matrix is fixed in our adaptation, the difference from original AdaCost is that each example will be assigned different cost in every iteration depending on whether it is predicted as a true or false positive or negative. 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 $cost(FN) = f$. f is an input parameter to the algorithm.

The AdaCost algorithm essentially modifies the AdaBoost's weight update equation 2 to

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

where $\alpha_t = 1/2\ ln((1+r_t)/(1-r_t))$ and $r_t = \sum_i D_t(i)exp(-y_i h_t(x_i)\beta_{sign(h_t(x_i)y_i)})$. 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 $\beta_{TP} = 0.5 - 0.5\ cost(TP)/f = 0.5$, $\beta_{TN} = 0.5 - 0.5\ cost(TN)/f = 0.5$, $\beta_{FP} = 0.5 + 0.5\ cost(FP)/f = 0.5(f+1)/f$, $\beta_{FN} = 0.5 + 0.5\ cost(FN)/f =$

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, $h_t^{+1} : x_i \rightarrow \{+1, 0\}$, using D_t.

ii. Learn weak model for $y_i = -1$ (class NC) examples, $h_t^{-1} : x_i \rightarrow \{-1, 0\}$, using D_t.

iii. Evaluate C's model and NC's model:

$$TP_t = \sum_{i:y_i>0, h_t^{+1}(i)>0} D_t(i), \qquad FP_t = \sum_{i:y_i<0, h_t^{+1}(i)>0} D_t(i) \tag{17}$$

$$TN_t = \sum_{i:y_i<0, h_t^{-1}(i)<0} D_t(i), \qquad FN_t = \sum_{i:y_i>0, h_t^{-1}(i)<0} D_t(i) \tag{18}$$

iv. Choose Model and Compute importance weight, α_t:
if($(1 - (TP_t - FP_t)^2) < (1 - (TN_t - FN_t)^2)$) then, Choose C's Model, by setting $h_t = h_t^{+1}$:

$$\alpha_t = 0.5 \; ln \, (TP_t / FP_t) \tag{19}$$

else Choose NC's Model by setting $h_t = h_t^{-1}$:

$$\alpha_t = -0.5 \; ln \, (TN_t / FN_t) \tag{20}$$

endif

v. Update Weights:

$$D_{t+1}(i) = D_t(i) exp(-\alpha_t y_i)/Z_t; h_t(x_i) \neq 0, \tag{21}$$

where Z_t is chosen such that $\sum D_{t+1}(i) = 1$.

endfor
Final Model:

$$H(x) = sign \left(\sum_{t:x \text{ satisfies } h_t} \alpha_t \right) \tag{22}$$

Figure 4: *RareBoost-2 Algorithm: The difference from SLIPPER algorithm is the choice between C's model and NC's model, and absence of Default Model*

1.0. f is being used as normalization factor. β_+ and β_- satisfy the required constraints, for $f \geq 1$. Using these values, the expanded weight update formulae look like

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

As these indicate, unlike previously described algorithms, true positives get modified with a different factor than false positives and true negatives get modified with a different factor than false negatives. The qualitative effect is that true predictions are suppressed by a smaller factor than by which false predictions are boosted. We quantize this effect in the next section, and discuss its repercussions on recall and precision.

Other Cost-sensitive Boosting Algorithms:

Two other variations of cost-sensitive algorithms are described in [13]. One variation called, CSB1, does not use any α_t factor (or $\alpha_t = 1$) and the other variation CSB2 uses same α_t as computed by AdaBoost. The weight update formulae for these two algorithms are

$$\begin{aligned} CSB1 &: D_{t+1}(i) = (D_t(i)C_{sign(h_t(x_i)y_i)}exp(-y_i h_t(x_i))/Z_t. \\ CSB2 &: D_{t+1}(i) = (D_t(i)C_{sign(h_t(x_i)y_i)}exp(-\alpha_t y_i h_t(x_i))/Z_t. \end{aligned}$$

The parameters C_+ and C_- are defined as $C_+ = 1$, and $C_- = cost(y_i, h_t(x_i)$, where $cost(i,j)$ is the cost of incorrectly predicting class j for an example whose actual class is i. Using the same cost matrix structure that we use for AdaCost, this is equivalent to $C_{TP} = 1$, $C_{TN} = 1$, $C_{FP} = 1$, and $C_{FN} = f$. The other feature of their algorithm is that they use Baye's optimality rule to combine the predictions of all models, so as to make a prediction that will incur lowest misclassification cost. In other words, in binary classification scenario, CSB1 and CSB2 modify the Final Model equation 3 of AdaBoost to

$$H(x) = argmin_k \left(\sum_{c:\{-1,+1\}} \left(\sum_{t:sign(h_t(x))=c}^{M} \alpha_t\ h_t(x)\ cost(c,k) \right) \right)$$

5 Comparing All Algorithms from One Perspective

In previous sections, we described three categories of algorithms, and proposed enhancements of two cost-insensitive algorithms. In this section, we put all these algorithms in one single perspective of how they update the weights on four types of examples that exist after each iteration: True Positives (TP), False Positives (FP), True Negatives (FN), and False Negatives (FN). For the purposes of this discussion, we make two assumptions: **(a)** positives refer to the rare target class (C) of interest, and negatives refer to the other class (NC), which can in reality be a combination of all the classes other than C; and **(b)** The model learned in each iteration gives a pure decision {+1,-1} in case of non-abstaining or cost-sensitive algorithms, and a decision of either {+1,0} or {-1,0} in case of abstaining algorithms. This second assumption essentially ignores the confidence-rating of the decisions and looks merely at the signs. Doing so simplifies the quantitative analysis without losing the validity of the qualitative analysis that we present.

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

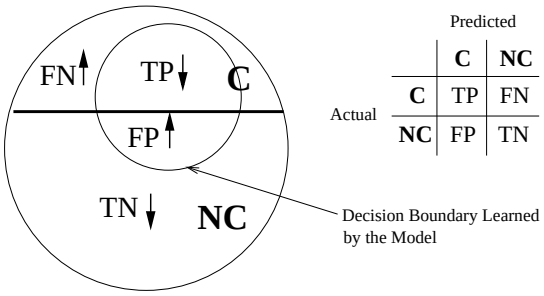


Figure 5: *How Boosting affects TP, FP, FN, and TN examples after each iteration. Upward arrow indicates aggregate weight increase and downward arrow indicates aggregate weight decrease.*

FN or FP and TN from iteration to iteration (its decided by the model learned in a given iteration). However, we can do an analysis of the effect of weight update of t^{th} iteration on the $(t+1)^{st}$ iteration.

The first comparative observation that we present is how each algorithm modifies the weights of TP, FP, TN, FN. Since each algorithm treats all examples of one kind equally (i.e. one TP is not distinguished from the other), we can look at the aggregate effect of weight update mechanisms of each algorithm. Using weight update equations that we stated for all the algorithms in previous section, and expanding them, we can derive the factors by which each example type gets suppressed or boosted. Table 1 summarizes these numbers for all algorithms. This table essentially puts all the algorithms in one perspective. We now use the table to discuss the effects of each algorithm on recall and precision of C.

We follow a simple rule: in order to achieve good recall, the algorithm has to build better models for distinguishing FN from TN, and in order to achieve good precision, the algorithm should build a better model to distinguish FP from TP. All algorithms try to concentrate on FP and FN examples by boosting their weights, and suppressing weights of TP and TN. This is pictorially illustrated in Figure 5. Looking at this figure, one can realize that, in the next iteration, a model that is geared towards learning C will try to capture more FN and less FP examples. This happens because FP's belong to NC class, and model for C will try to capture as little of those as possible to achieve high accuracy for C. So the model makes an effort 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 that learns NC in the next iteration, will try to learn better model for FP that captures as little of FN as possible, thus trying to convert more FP's into TN's. However, it might capture more TP's if weight on the TP's is reduced to low levels, thus losing recall. The moral of this discussion is that the factors by which the weights of the four types of examples are updated is crucial in how an algorithm can achieve balance between recall and precision.

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. This separate focus on two types of errors, may give RareBoost-1 better control over achieving higher levels for both recall and precision of C (as well as NC). 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

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	

Table 1: *Comparing the effect of weight updates in each of the algorithms. Notes: 1) This analysis assumes the range of h_t's prediction to be binary; i.e., without any confidence rating associated with any decision. 2) TP_t, TN_t, FP_t, FN_t denote the sum of weights on true positive, true negative, false positive, and false negative examples respectively, at the beginning of t^{th} iteration.*

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.

RareBoost-1 vs. RareBoost-2:

RareBoost-2 can be compared to AdaBoost similarly. In fact, RareBoost-2 is achieving an effect similar to RareBoost-1, however the difference is that its abstaining strategy allows the weak learner to focus on either of the two objectives (recall or precision) in the next iteration. For example, if RareBoost-2 chooses an NC model in an iteration (as against C model), then it will leave TP and FP untouched, and just stratify TN and FN. So, in the next iteration a model for NC may build better model to distinguish TN from FN, thereby aiming for recall. If C model is chosen, then the next iteration will aim for precision. This strategy of dynamically emphasizing one objective over the other can be thought as being similar to the search-based optimization algorithms that try to reach the goal in the orthogonal directions associated with the multi-attribute objective. RareBoost-1 on the other hand, tries to reach the goal along a composite direction. So, by having two flavors of algorithms RareBoost-1 and RareBoost-2, we can evaluate one strategy over the other, or simply use whichever is best for the given problem.

SLIPPER vs. RareBoost-2:

The difference in SLIPPER and RareBoost-2 is that SLIPPER chooses between the model of a pre-specified fixed class (C or NC) and a default model, whereas RareBoost-2 chooses a model between two classes C and NC and does not use default model. As we discussed in section 3, SLIPPER's primary capability to achieve recall comes from its choice of default model, especially for rare classes, because a traditional weak learner may not be able to build good model for such class [7]. As Table 1 shows, the effect of choosing default model is to stratify all positive (true and false) examples against all negative (true and false) examples, thereby stratifying one class against the other. This effectively puts SLIPPER back to square one. This strategy seems ad-hoc. 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.

AdaCost vs. Cost-Insensitive Algorithms:

Looking at Table 1, it can be seen that AdaCost updates weights in a completely different manner than any of the cost-insensitive algorithms. It comes close to a possible generic strategy of updating weights of all four types of examples differently. It primary effect, for values of $f > 1$, is to increase weights on FN examples more than any other type of examples. Also, it does not reduce the weights on true predictions as much as previous algorithms do. The net effect is that the weak learner focuses on capturing more of false negatives in next iteration, thereby increasing recall. The false positives also get more weight compared to true positives. 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. Note that unlike techniques that purely focus on recall by an up-front oversampling of rare classes, AdaCost has an ability to work on precision also.

AdaCost vs. CSB1 and CSB2:

As Table 1 shows, CSB1 and CSB2 update weights on TP, FP, and TN, similar to the cost-insensitive algorithms. However, they focus on FN, like AdaCost, and this focus can be emphasized by varying f. The difference from AdaCost is that the focus on recall is not accompanied by a focus

on precision, because weights of FP get emphasized less w.r.t. TP or TN. For the rare classes, this has an effect of achieving very good recall but at a loss of precision, because pure rare class signatures are usually difficult to find.

6 Results on Synthetic Datasets

In this section, we describe some models of synthetic datasets, that can be used to control the difficulty in achieving recall and precision by a classifier algorithm. Using these datasets, we compare all the algorithms for their recall and precision performance.

6.1 Base Classifier, Evaluation Strategy, and Algorithm Parameters

We used RIPPER [3] as our base classifier. In order to make a fair comparison, we used the evaluation metric of information gain in all the algorithms. Also, in all the algorithms, we let RIPPER grow as many rules as its stopping criterion would allow[4]. We fixed the number of optimizing loops of RIPPER to 0 instead of recommended 2 loops in RIPPER code. This ensures that we are using a weak algorithm at the heart of boosting rather than a very sophisticated algorithm. Except for SLIPPER and RareBoost-2, which choose only one model for either of the classes in each iteration, for all other algorithms, we let RIPPER grow two models (one for C and other for NC) and a default rule. Whichever rule applies with the best accuracy is chosen to make the prediction.

Here is how we evaluate the performance of each algorithm. We let each algorithm run for 100 boosting iterations. After every iteration, we monitor recall, precision, and F_1-measure on the test data[5]. The numbers that we report in the following results are the ones that correspond to highest value of F_1-measure. F_1-measure is defined as 2*(recall*precision)/(recall+precision), where recall and precision are defined with respect to target class C; recall = TP/(TP+FN) and precision = TP/(TP+FP).

We used a few different versions of each of the algorithms, except for AdaBoost and RareBoost-2, where only one version as described in Figures 1 and 4. For RareBoost-1, we used two versions of α_t, as given in equations 4 and 5, and chose the best performing α_t for a given dataset. For SLIPPER, in one version, we let it focus on C in all iterations, and in the other one, we let it focus on NC in all iterations. The best result of the two is reported. For CSB1 and AdaCost, we use three versions each with different f value: 1, 2, or 5, and report the best. CSB2 is equivalent to AdaBoost, when $f = 1$, so we just use two f values for it 2 and 5, and report the best. Last point to note is that, we used only the binary valued models (i.e. ignored confidence-rated predictions) for AdaBoost, RareBoost-1, CSB1, CSB2, and AdaCost. This is done so as to be consistent with the simplifying assumption we made for our qualitative analysis. For SLIPPER and RareBoost-2, the confidence rating is embedded in the α_t factor.

6.1.1 Results on Datasets with No correlations among attributes

Figure 6 describes the model used for generating such datasets. The model has three types of attributes as described in the figure. The records of classes C and NC are divided into multiple

[4]SLIPPER paper [2] suggests using a different evaluation metric that will aim to minimize Z_t while building the rule. We did experiments with it, but in very few cases did we find it giving any special performance boost for SLIPPER. In fact, in many cases, its performance was worst. Also SLIPPER paper aims to build a classifier with just one rule, which again we found to perform poorly in our experiments, as compared to multi-rule classifier.

[5]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.

subclasses. For each attribute, the distribution of classes is shown over the range of its values. Each attribute distinguishes one subclass of C and/or one subclass of NC. For example, a subclass C_a has a peak in its distribution on attribute $ACOM_a$ means that $ACOM_a$ can be used to distinguish that C_a by a rule that captures the range of values in $ACOM_a$ where C_a peaks. The subclasses that are not distinguishable by a given attribute are uniformly distributed all over the range of that attribute (such subclasses are collectively indicated as C_{rest} and NC_{rest} in the figure). The figure assumes C as the rare class, hence peaks of its subclasses are shown smaller.

In our experiments, we fixed number of ACOM-type attributes to 2, and number of ANC-type attributes to 3 and number of AC-type attributes to 2. We varied four parameters. The parameters are defined in Figure 6. These parameters let us control the recall and precision obtainable for a dataset. If C_b peaks are made wider (increasing WC_b), whenever C_b's peaks are captured, lot of false positives (NC examples) will be captured. So, the algorithm must have good ability to achieve precision. Effect of making C_a peaks wider is similar, however, C_a peak does not capture as many false positives as C_b peak of same width, because the $ACOM_a$ attribute also distinguishes NC_a. The reason to have the subclasses of type C_a is as follows. When C_a is captured, false positives of type NC_b are captured more than of those of type NC_a. Thus, controlling the nature of signature of NC_a and NC_b, we can achieve two distinct types of false positives. If an algorithm can learn only NC_a better, then its precision will suffer.

We describe results for five different sets of parameters, that make the four types of peaks wider or sharper. These parameters are given in the top part of Table 2. The results for these datasets are given in the bottom part. The proportion of the target class C with respect to NC was fixed at 5% in all these datasets.

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), thus capturing more false positives, the performance of the algorithms, except SLIPPER, remains competitive. This can be explained by the fact that most algorithms can still build good models for NC, because of its huge majority and sharp peaks. The reason for SLIPPER's performance deterioration can be attributed to its default model selection 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 NC subclass (as compared to NC_a type) captured when C's signatures are learned, and it becomes difficult to remove these false positives without removing true positives 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. Although AdaCost is achieving the best F_1-measure on the basis of boosting false negatives significantly (this is achieved for f=5), in the process it suffers from a very poor precision (lowest precision). Comparing AdaBoost with RareBoost-1 shows that our enhancement of AdaBoost is helping in achieving higher precision while maintaining the recall at the level of AdaBoost. Comparing SLIPPER with RareBoost-2 shows that our enhancement of SLIPPER is significantly better in both recall and precision. If fact, it can be claimed that although RareBoost-2 is achieving lower F_1-measure than AdaCost, its recall and precision numbers are better balanced as compared to all the other algorithms. 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 ability to significantly emphasize on recall (again this number is achieved for $f = 5$). Its precision, however, is again much worse as compared to that of RareBoost-1. In this case, RareBoost-1 can be said to achieve better balance. RareBoost-2 is again better than SLIPPER. The last dataset 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. However, RareBoost-1 and RareBoost-2 also win against their predecessors respectively, on the counts of both recall and precision.

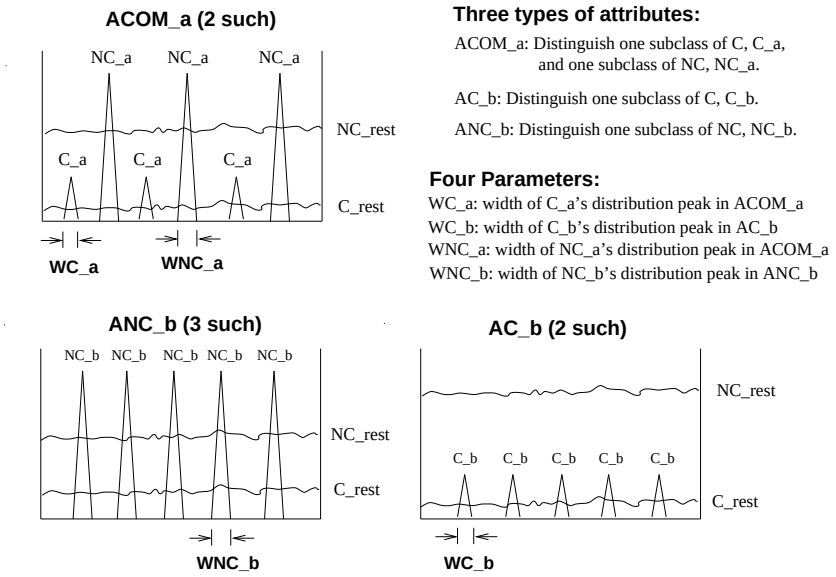


Figure 6: *Description of the model generating datasets with no correlations among attributes.*

Overall, these results indicate the following: **(a)** AdaCost can perform well for the rare classes of the kind generated with this model, at least on the set of parameters chosen, however it does not always obtain a good balance between recall and precision. **(b)** RareBoost-1 and RareBoost-2 can outperform their predecessors, and have ability to achieve better balance than AdaCost in some cases. **(c)** CSB1 and CSB2 variants of cost-sensitive boosting are not very good in their ability to perform as well as our enhancements of cost-insensitive algorithms, maybe because their choice of weight update mechanisms is ad-hoc without much theoretical justification.

6.1.2 Datasets with correlations among attributes

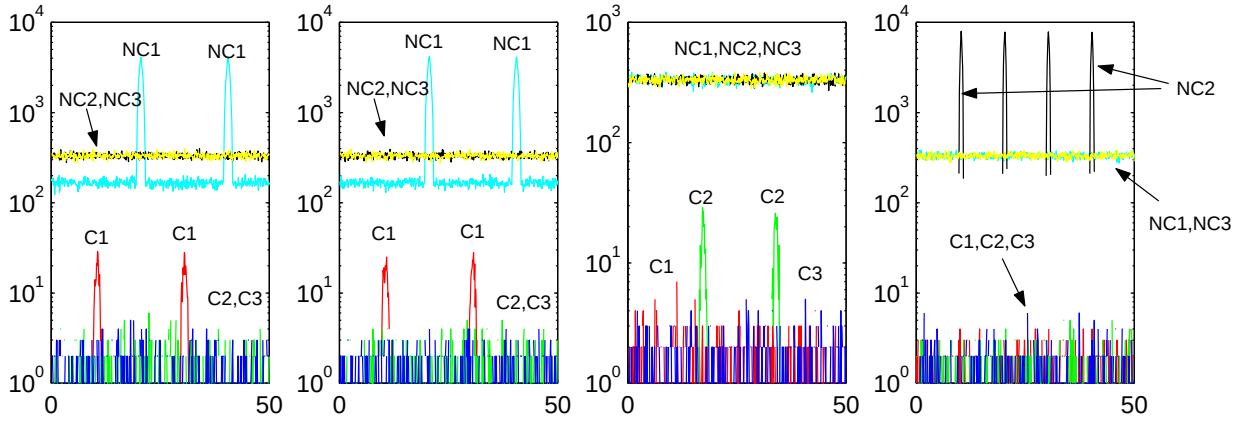
We now present results on synthetic datasets that have correlations among attributes. The data generating model was first used by us in [7], and is described in Figure 7. It has similar nature to the previous data model, except that now it has correlations among the signatures of $ACOM_a$ type of attributes. The signature peaks for C1 in first two attributes (leftmost and the one next to it) are correlated: in the sense that when first attribute is in the range of first (resp. second) C1 peak, the second attribute is also in the range of first (resp. second) C1 peak in its distribution. Similar correlation exists in the peaks of NC1 in these attributes. One more added feature is that now some of the attributes are categorical (similar signature peaks exist in categorical domain). We believe that this model is fairly general and complex to represent real-life situations.

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 in training (and test) data is varied. Looking at the column of any algorithm shows that as the target class proportion increases, it becomes easier to achieve better recall and better precision. RareBoost-1 outperforms AdaBoost in all the cases on the counts of both recall and precision. An important observation is that the performance gap increases as target class proportion decreases. RareBoost-2 does similarly compared to SLIPPER. 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 C-frac values, AdaCost achieves its best only for $f = 1$. And it is not

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	.8476	.8476	.8792	.8516	.8476	.8476	.8476
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	.8509	.8509	.7662	.8509	.8509	.8509	.8555
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	.4859	.5110	.4404	.5501	.5274	.5274	.6253
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	.6479	.7120	.6339	.6745	.5410	.4395	.7279
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	.5840	.6500	.5793	.6733	.5333	.6492	.7508

Table 2: *Top Table: Specific Datasets Generated with model of Figure 6. Peak widths are given as a percentage of the total range of the attribute. Bottom Table: Results on datasets with no attribute correlations. Notation: R:recall for C (in %), P: precision for C (in %), F: F_1-measure, ABst: AdaBoost, RB-1: RareBoost-1, SLIP: SLIPPER, RB-2: RareBoost-2, ACst: AdaCost.*



C3 and NC3, subclasses of target and non−target respectively,
are distinguished by categorical attributes:
C3: na=1, nspa=2, nwps=2
NC3: na=1, nspa=4, nwps=2

Figure 7: *Description of datasets that have correlations among attributes. This dataset is same as the one used in [7].*

C-frac		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	.5941	.6714	.6094	.6677	.5851	.5839	.6779
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	.7085	.7546	.6718	.7722	.6986	.5984	.7403
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	.7852	.8030	.7405	.8277	.7482	.7026	.7986
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	.8091	.8491	.8167	.8353	.8121	.8007	.8473

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

as good as RareBoost-2 for 13.1% dataset. In fact, with $f = 5$ its numbers for 5.7% dataset are R=81.33%, P=58.49%, and F=0.6804. For the 13.1% dataset, with $f = 2$, its numbers are similar to the reported numbers, and with $f = 5$, it achieves R=90.13%, P=68.08%, and F=0.7757. These numbers indicate that over-emphasizing 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

Boosting algorithms have shown promise in improving classification performance of a weak learner. Learning descriptive models for rare classes is emerging as a crucial problem. There does not exist any work that critically compares boosting strategies for mining rare classes. In this paper, we have taken up that task. We looked at three different representative categories of boosting algorithms. Our detailed analysis showed how each algorithm affects four types of examples, true and false positives and true and false negatives, after each iteration. We use this observation to qualitatively compare the effect of each algorithm on recall and precision of the rare class.

Another contribution is that we propose two enhancements to the popular algorithms AdaBoost and SLIPPER in the cost-insensitive category. We have designed special synthetic data generation models, whose parameters can be varied to control the recall and precision achievable by an algorithm. Using these models, we have supported our qualitative comparison between algorithms. We have demonstrated the situations in which our proposed enhancements outperform their predecessors. Our results indicate that they are required especially for the rare classes. Also, we have shown that in some situations, they can outperform even the most competitive cost-sensitive algorithms.

In summary, 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 for 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] Philip Chan and Salvatore 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] William W. Cohen. Fast effective rule induction. In *Proc. of Twelfth International Conference on Machine Learning*, Lake Tahoe, California, 1995.

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

[5] Yoav Freund and Robert Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[6] Adam J. Grove and Dale Schuurmans. Boosting in the limit: Maximizing the margin of learned ensembles. In *Proc. 15th National Conference on Artificial Intelligence (AAAI-98)*, Madison, WI, 1998.

[7] Mahesh V. Joshi, Ramesh C. Agarwal, and Vipin 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.

[8] Robert Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26(5):1651–1686, 1998.

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

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

[11] Robert E. Schapire, Yoram Singer, and Amit Singhal. Boosting and rocchio applied to text filtering. In *Proc. 21st International ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 1–10, Melbourne, Australia, 1998.

[12] Fabrizio Sebastiani, Alessandro Sperduti, and Nicola 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.

[13] Kai Ming 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.

[14] C. J. van Rijsbergen. *Information Retrieval.* Butterworths, London, 1979.