AdaCost: Misclassification Cost-sensitive Boosting

Wei Fan
Computer Science
Columbia University
New York, NY 10027
wfan@cs.columbia.edu

Salvatore J. Stolfo
Computer Science
Columbia University
New York, NY 10027
sal@cs.columbia.edu

Junxin Zhang
Computer Science
Columbia University
New York, NY 10027
jzhang@cs.columbia.edu

Philip K. Chan
Computer Science
Florida Institute of Technology
Melbourne, FL 32901
pkc@cs.fit.edu

Abstract

AdaCost, a variant of AdaBoost, is a misclassification cost-sensitive boosting method. It uses the cost of misclassifications to update the training distribution on successive boosting rounds. The purpose is to reduce the cumulative misclassification cost more than AdaBoost. We formally show that AdaCost reduces the upper bound of cumulative misclassification cost of the training set. Empirical evaluations have shown significant reduction in the cumulative misclassification cost over AdaBoost without consuming additional computing power.

1 Introduction

Recently, there has been considerable interest in cost-sensitive learning [15, 8, 7, 17, 6, 2]. Turney [15, 16] discusses learning tasks sensitive to the costs of misclassification among others. We are interested in reducing *misclassification cost*. It can be either constant for each type of misclassification or conditional on a specific example under different types of misclassification. In troubleshooting systems, for example, there is usually a fixed cost in producing one type of wrong diagnosis. In fraud detection systems, however, undetected frauds with high transaction amounts are obviously more costly. In this paper, we propose AdaCost, a variant of AdaBoost, that reduces both fixed and variable misclassification costs more significantly than AdaBoost.

Freund and Schapire's AdaBoost [4] learns a highly accurate voted ensemble of many "weak" hypotheses. Typically, each hypothesis outputs both a prediction and a confidence for this prediction. Each hypothesis is trained on the same data set yet with a different distribution. Different hypotheses are produced in different rounds of boosting. At each round, AdaBoost increases the weights of wrongly classified training instances and decreases those of correctly predicted instances. AdaBoost allows for an arbitrary initial distribution. For classification error, each example is given an equal weight. To reduce cumulative misclassification costs, costly examples can be given higher weights. AdaBoost reduces the weighted error for this initial distribution. Schapire, Singer and Singhal [11] gave different weights for false positives and false negatives to apply AdaBoost in text-filtering. Karakoulas and Shawe-Taylor [5] applied a similar approach. However, misclassification cost is not used in AdaBoost's weight updating rule. In AdaCost, the weight updating rule increases the weights of costly wrong classifications more aggressively, but decreases the weights of costly correct classifications more conservatively. This is accomplished by introducing a misclassification cost adjustment function into the weight updating formula. Under this updating rule, the weights for expensive examples are higher and the weights for inexpensive examples are comparatively lower. Each weak hypothesis correctly predicts more expensive examples for such a distribution. The final voted ensemble will also correctly predict more costly instances.

The focus of this paper is on both theoretical issues and empirical evaluations. In Section 2, we describe AdaCost and derive a reasonable upper bound on the misclassification cost. We also discuss the choices of the cost adjustment function β and the hypothesis weight α to reduce this upper bound. In Section 3, we evaluate AdaCost against AdaBoost on both real world credit card fraud detection and publicly available data sets. In Section 2.6, we discuss an alternative to AdaCost.

- Given: $\mathcal{S} = \{(x_1, c_1, y_1), \ldots, (x_m, c_m, y_m)\}$; $x_i \in \mathcal{X}, c_i \in \mathbb{R}^+, y_i \in \{-1, +1\}$.
- Initialize $D_1(i)$ (such as $D_1(i) = c_i / \sum_j^m c_j$).
- For $t = 1, \ldots, T$:
  1. Train weak learner using distribution D_t.
  2. Compute weak hypothesis $h_t : \mathcal{X} \to \mathbb{R}$.
  3. Choose $\alpha_t \in \mathbb{R}$ and $\beta(i) \in \mathbb{R}^+$.
  4. Update

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

     where $\beta(i) = \beta\Big(\text{sign}(y_i h_t(x_i)), c_i\Big)$ is a cost-adjustment function. Z_t is a normalization factor chosen so that D_{t+1} will be a distribution.
- Output the final hypothesis:

$$H(x) = \text{sign}(f(x)) \text{ where } f(x) = \left(\sum_{t=1}^{T} \alpha_t h_t(x) \right)$$

Figure 1: AdaCost

2 AdaCost Algorithm

2.1 Difference from AdaBoost

We follow the generalized analysis of AdaBoost by Schapire and Singer [9]. The algorithm is shown in Figure 1. Let $\mathcal{S} = ((x_1, c_1, y_1), \ldots, (x_m, c_m, y_m))$ be a sequence of training examples where each *instance* x_i belongs to a *domain* $\mathcal{X}$, each *cost factor* c_i belongs to the *non-negative real domain* $\mathbb{R}^+$, and each *label* y_i belongs to a finite *label space* $\mathcal{Y}$. In this paper, we focus on binary classification problems in which $\mathcal{Y} = \{-1, +1\}$. h is a weak hypothesis. It has the form $h : \mathcal{X} \to \mathbb{R}$. The sign of $h(x)$ is interpreted as the predicted label and the magnitude $|h(x)|$ is the "confidence" in this prediction. Let t be an index to show the round of boosting and $D_t(i)$ be the weight given to (x_i, c_i, y_i) at the t-th round. $0 \leq D_t(i) \leq 1$ and $\sum D_t(i) = 1$. α_t is a chosen parameter as a weight for weak hypothesis h_t at the t-th round. We assume $\alpha_t > 0$. $\beta(\text{sign}(y_i h_t(x_i)), c_i)$ is a cost adjustment function with two arguments: $\text{sign}(y_i h_t(x_i))$ to show if $h_t(x_i)$ is correct, and the cost factor c_i.

The difference between AdaCost and AdaBoost is the additional cost adjustment function $\beta(\text{sign}(y_i h_t(x_i)), c_i)$ in the weight updating rule (highlighted by a box in Figure 1). Where it is clear in context, we use either $\beta(i)$ or $\beta(c_i)$ as a shorthand for $\beta(\text{sign}(y_i h_t(x_i)), c_i)$. Furthermore, we use β_+ when $\text{sign}(y_i h_t(x_i)) = +1$ and β_- when $\text{sign}(y_i h_t(x_i)) = -1$. For an instance with a higher cost factor, $\beta(i)$ increases its weights "more" if the instance is misclassified, but decreases its weight "less" otherwise. Therefore, we require $\beta_-(c_i)$ to be non-decreasing with respect to c_i, $\beta_+(c_i)$ to be non-increasing, and both are non-negative. Karakoulas and Shawe-Taylor [5] have also introduced a similar parameter β into the weight updating formula: if $y = +1$ then $\beta = 1$, if $y = -1$, then $\beta = v$ $(v < 1)$. Their intuition is different from our approach. In their approach, positives are given a "fixed" higher weight than negatives. The weight updating rule will increase the weights of false negatives (+1 predicted as −1) more than false positives (−1 predicted as +1). However, it will also decrease the weights of true positives more than true negatives. Two related but different cost-sensitive boosting approaches for tree classifications are proposed by Ting and Zheng [14]. Their approaches apply to situations where the costs change very often. The solution is to repeatedly use the same induced model yet with different misclassification costs at classification. Our proposal applies to situations where misclassification costs are relatively stable. In the first approach, they modify the classification procedures of AdaBoost by calculating the expected misclassification cost for every class and choosing the predicted class with the lowest expected cost for a given instance. In the second approach, they completely change the weight updating rule. The rule replaces an instance's weight with the misclassification cost if it is misclassified; otherwise the current weight is retained. They didn't provide a training misclassification cost upper bound.

2.2 Training Misclassification Cost Upper Bound

Before studying how to reduce the training misclassification cost, we derive an upper bound on the cumulative training misclassification cost.

Lemma 1
Let $f'(x) = \sum_{t=1}^{T} \alpha_t h_t(x) \beta\Big(sign(y h_t(x)), c\Big)$ *and* $H'(x) = sign(f'(x))$. *If* $\forall c, \beta_-(c) \geq \beta_+(c)$, *the following is true:*

$$\forall x \in \mathcal{S} \left(H'(x) = y \implies H(x) = y \right)$$

Proof: By definition of $H'(x)$ and $f'(x)$:

$$H'(x) = y \iff$$

$$yf'(x) = y\sum_{t=1}^{T}\alpha_t h_t(x)\beta\Big(\text{sign}(yh_t(x)), c\Big) > 0 \quad (1)$$

The righthand side of (1) can be rewritten as a sum of the correct and wrong portions of predictions by the weak hypotheses:

$$y\sum_{t_+}\alpha_{t_+}h_{t_+}(x)\beta_+ + y\sum_{t_-}\alpha_{t_-}h_{t_-}(x)\beta_- > 0 \quad (2)$$

Since $\beta_- \geq 0, \beta_+ \geq 0$ and $\alpha_t > 0$ (the requirements by AdaCost), in Eq (2),

$$y\sum_{t_+}\alpha_{t_+}h_{t_+}(x)\beta_+ > 0 \text{ and } y\sum_{t_-}\alpha_{t_-}h_{t_-}(x)\beta_- \leq 0$$

The Lemma requires that $\beta_- \geq \beta_+ > 0$, the following must be true because we have decreased the votes of the wrong portion of predictions in Eq (2).

$$\begin{array}{lcl} y\sum_{t_+}\alpha_{t_+}h_{t_+}(x)\beta_+ & + & y\sum_{t_-}\alpha_{t_-}h_{t_-}(x)\beta_+ \\ & \geq & \\ y\sum_{t_+}\alpha_{t_+}h_{t_+}(x)\beta_+ & + & y\sum_{t_-}\alpha_{t_-}h_{t_-}(x)\beta_- \end{array} \quad (3)$$

Combining Eqs (2), (3) and definition of $f(x)$ (defined in Figure 1) gives the following:

$$\beta_+(yf(x)) = y\sum_{t_+}\alpha_{t_+}h_{t_+}(x)\beta_+ + y\sum_{t_-}\alpha_{t_-}h_{t_-}(x)\beta_+ > 0 \quad (4)$$

$\beta_+ > 0$ and (4) implies

$$yf(x) > 0 \quad (5)$$

By definition of $H(x)$ and $f(x)$, (5) shows that $H(x) = y$. ■

Lemma I shows that for every example correctly classified by $H'(x)$, it is also correctly classified by $H(x)$. The difference of $H(x)$ from $H'(x)$ is that $H(x)$ has no $\beta\Big(\text{sign}(yh_t(x)), c\Big)$ terms.

Theorem 1 *The following holds for the upper bound of the training cumulative misclassification cost.* $[\![\pi]\!]$ *returns 1 if predicate* $\pi = true$ *or 0 otherwise.*

$$\sum c_i[\![H(x_i) \neq y_i]\!] \leq d\prod_{t=1}^{T} Z_t,\ d = \sum c_j$$

Proof: From Lemma I, we know that

$$\sum c_i[\![H(x_i) \neq y_i]\!] \leq \sum c_i[\![H'(x_i) \neq y_i]\!] \quad (6)$$

By unraveling the update rule (extended and modified from the proof of Theorem I in Schapire and Singer [9]), we have that

$$\begin{aligned} D_{T+1}(i) &= \frac{D_1(i)\exp(-\sum_t \alpha_t y_i h_t(x_i)\beta(i))}{\prod_t Z_t} \\ &= \frac{D_1(i)\exp(-y_i f'(x_i))}{\prod_t Z_t} \end{aligned} \quad (7)$$

Moreover, if $H'(x_i) \neq y_i$, then $y_i f'(x_i) \leq 0$ implying that $\exp(-y_i f'(x_i)) \geq 1$. Thus,

$$[\![H'(x_i) \neq y_i]\!] \leq \exp(-y_i f'(x_i)) \quad (8)$$

Combining Eqs (6),(7), (8) and $D_1(i) = c_i/\sum_j^m c_j$ gives the stated bound on cumulative misclassification cost since

$$\begin{aligned} \sum c_i[\![H(x_i) \neq y_i]\!] &\leq \sum c_i \cdot \exp(-y_i f'(x_i)) \\ &= \sum_i (\prod_t Z_t)(\frac{c_i}{D_1(i)})D_{T+1}(i) \\ &= d\prod_{t=1}^{T} Z_t,\ d = \sum c_j \end{aligned}$$

■

The reason to require $\beta_- \geq \beta_+$ in Lemma I is clear. It removes the cost adjustment function β and the true label y terms from $H'(x)$ to generate the final hypothesis $H(x)$. This is important since neither the cost c nor the label y is available during testing. Additionally, Lemma 1 shows that $H(x)$ is more accurate than $H'(x)$. Moreover, for a particular example, $\beta_- \geq \beta_+$ guarantees that the weight updating rule will increase the weight more for wrong classifications than decreasing the weight for correct predictions. The requirements of non-decreasing $\beta_-(c_i)$ and non-increasing $\beta_+(c_i)$ force the weak learner to give more attention to more "costly" examples.

2.3 Choosing α_t

One consequence of Theorem 1 is that, to reduce training cost, we should seek to minimize Z_t at each round of boosting in the choice of α_t. We derive two choices for α_t following an estimation method by Freund and Schapire [4] and a numerical method by Schapire and Singer [9] and Karakoulas and Shave-Taylor [5]. We omit the round index t. For weak hypothesis h with range [-1,+1] and cost adjustment function $\beta(i)$ in the range [0,+1], the choice of α is

$$\alpha = \tfrac{1}{2}\ln\frac{1+r}{1-r} \quad \text{where } r = \sum_i D(i)u_i,\ u_i = y_i h(x_i)\beta(i) \quad (9)$$

Since $u_i \in [-1, +1]$, the following inequality holds.

$$Z = \sum D(i)e^{-\alpha u_i} \le \sum D(i)\left(\frac{1+u_i}{2}e^{-\alpha}+\frac{1-u_i}{2}e^{\alpha}\right)$$

By zeroing the first derivative of the right hand side, we have the choice of α in formula (9). For this choice of α, $Z \le \sqrt{1-r^2} \le 1$. The above proof is extended from the estimation method given by Schapire and Singer [9].

Corollary 1 *Assuming h_t has range [-1,+1] and $\beta(i)$ has range [0,+1]. α_t is chosen as in formula (9). The training cumulative cost of H is at most:*

$$d\prod_{t=1}^{T}\sqrt{1-r_t^2}$$
$$\textit{where } r_t = \sum_i D_t(i)y_i h_t(x_i)\beta(i) \textit{ and } d = \sum_j c_j$$

The upper bound on misclassification cost is actually reduced when $Z_t = \sqrt{1-r^2} < 1$. This estimation for Z is pessimistic and not tight. The exact range of u is $\{-\beta_-, \beta_+\}$. It is hard to find a general and tight analytical upper bound for Z if u is taken as $\{-\beta_-, \beta_+\}$ and it would still be an estimation.

A general numerical solution is given by:

$$Z'(\alpha) = \frac{dZ}{d\alpha} = -\sum_i D(i)u_i e^{-\alpha u_i} = 0 \qquad (10)$$

Since $Z''(\alpha) > 0$, $Z'(\alpha)$ can have at most one zero. A detailed analysis of this formula without the cost factor β is given by Schapire and Singer [9]. In practice, we can use the estimation method to find a candidate for α and use numerical methods to improve.

If $\alpha_t < 0$ during learning, we can reverse the sign of h_t. Thus, we use $h'_t = -h_t$ and $\alpha'_t = -\alpha_t$.

2.4 Example

We show the intuition behind AdaCost by an example from the breast cancer data set (Section 3). $((p_2, \ldots, p_{10}), y)$ is the schema and $y \in \{m, b\}$. The cost factor for instances with label m is 1 or otherwise 0.33. $h_1(x)$ is the same for both AdaBoost and AdaCost. One rule says "if $p_3 = 1$ then b" with confidence 0.972. So, $((4, 1, 1, 3, 1, 5, 2, 1, 1), m)$ is misclassified as b. D_1 of this item is .002849. AdaBoost generates $\alpha_{AdaBoost_1} = 1.48027$ and changes the example's distribution to .013. AdaCost calculates $\alpha_{AdaCost_1} = 1.481265$ and updates the distribution to .033. At round 2, the weak hypotheses produced by AdaBoost and AdaCost are different. $h_{AdaBoost_2}(x)$, has a rule "if $p_3 = 1 \wedge p_9 = 1$ then m" with 0.99 confidence that applies to the item and $\alpha_{AdaBoost_2} = 1.026$. Hence, the example is still misclassified as b. On the other hand, $h_{AdaCost_2}(x)$ has a default rule "if the instance is not classified as b, then it is m" with a confidence of 0.99. The weight is $\alpha_{AdaCost_2} = 2.11$. Therefore, the prediction by AdaCost at round 2 is m, since $1.481265 * 0.972 < 2.11 * 0.99$. The obvious reason is that AdaCost increases the weight of this instance more than AdaBoost by introducing the cost adjustment function β into the weight updating rule. The weight updating rule of AdaCost gives more weight to misclassifications with true label m since it is 3 times as "expensive" as b.

2.5 Applying Cost to other Boosting Algorithms

The addition of a cost adjustment function β into the weight updating formula doesn't violate the assumptions of AdaBoost. It can be introduced to the variations of the original AdaBoost. We show by two examples. For Schapire and Singer's AdaBoost.MH [10] (a multi-class multi-label ranking AdaBoost algorithm), the cost adjustment function can be defined as $\beta\Big(\mathrm{sign}\Big(Y_i\{l\}h_t(x_i, l)\Big), c_i(l)\Big)$. $c_i(l)$ is the cost factor that document x_i is not categorized as label l. For Schapire and Singer's AdaBoost.MR [10] (a multi-class multi-label preference ranking AdaBoost algorithm), we can design $\beta\Big(\mathrm{sign}\Big(h_t(x_i, l_0) - h_t(x_i, l_1)\Big), c_i(l_0, l_1)\Big)$ where $c_i(l_0, l_1)$ is the cost factor to mistakenly prefer l_0 over l_1. (The symbols presented by the original authors are used above without explanation.)

2.6 Alternative Method

An alternative weight updating formula is:

$$D_{t+1}(i) = \frac{\boxed{\beta} D_t(i)\exp\Big(-\alpha_t y_i h_t(x_i)\Big)}{Z_t} \qquad (11)$$

The difference here from Figure 1 is that β is outside of the exponent. β is a non-decreasing function of the cost factor c, such as $\beta(c) = c$. The following holds for the upper bound of training cumulative cost: $\sum c_i [\![H(x_i) \neq y_i]\!] \le d\Big(\prod_t^{T-1} Z_t\Big)\Big(\sum_i \frac{D_T(i)}{\beta^{T-1}}\exp(-\alpha_T y_i h_i(x_i))\Big)$, $d = \sum c_i$. It can be easily proved by unraveling the weight updating rule (12) and using the techniques in the proof of Theorem 1. In order to reduce misclassi-

Table 1: Data Set Summary

$\mathcal{S}$	**Data**	**Data Size**	**Testing Size**	**Positive%**
1	hypothyroid	3163	CV	4.77
2	boolean	32768	CV	13.34
3	dis	2800	972	4.63
4	crx	690	CV	44.5
5	breast cancer	699	CV	34.5
6	wpbc	198	CV	23.74
7	chase	40K*10	40K*10	≈ 20

fication cost, we will choose α_t in the first $T-1$ rounds to minimize Z_t and in the last round minimize $\sum_i \frac{D_T(i)}{\beta^{T-1}} \exp(-\alpha_T y_i h_i(x_i))$. Since $u_i = y_i h_i, u_i \in [-1, +1]$, we can use the estimation and numerical methods to choose both $\alpha_t (1 \leq t \leq T-1)$ and α_T.

3 Experiment

3.1 Set up

Data Set: We used seven data sets to evaluate the performance of AdaCost against AdaBoost in reducing cumulative misclassification cost. Table 1 lists their summary statistics. Five of the data sets were downloaded from the UCI Machine Learning Database. "boolean" is an artificial data set. It has 15 boolean variables. The instance is positive if any 4 or 5 variables are true, or otherwise it is negative. The last data set is a real world credit card fraud detection data set from Chase Bank. It contains .5M transaction records spanning a period of a whole year. It was provided for our research on fraud and intrusion detection [12]. Information on the schema can be found in [13].

Cost Factor: For a simplified cost model in the credit card business [13], there is an overhead $\$ovrhd$ to challenge a fraud. For any transaction with a transaction amount lower than the overhead, that is, $\$amt \leq \$ovrhd$, the authorization system will simply authorize it. The bank loses all $\$amt$ for all frauds of this kind (yet another cost of doing business). We assign them a cost factor $c = 0$, since these transcations may not contribute to learning a fraud model for a fixed $\$ovrhd$.

We can therefore save money by catching frauds with $\$amt > \$ovrhd$. We lose $\$amt$ if they are not detected. We still lose $\$ovrhd$ even if they are caught. It implies that by leaving a fraud undetected, we will lose $(\$amt - \$ovrhd)$ more. For legitimate transactions mistakenly labelled as frauds, we will lose $\$ovrhd$. Let fn, fp and tp represent the predicates of *false negative, false positive* and *true positive*. For example, $fp(i) = true$ if the i-th example is a false positive. The total cost to be minimized is formulated by:

$$\sum_{i:fn(i)} \$amt_i + \sum_{j:fp(j)} \$ovrhd + \sum_{k:tp(k)} \$ovrhd \qquad (12)$$

If we don't detect any frauds, we lose $\sum_{i:p(i)} \$amt$ (p returns *true* for fraud). We call this quantity *Maximal Loss*. If every fraud is detected, we still lose $\sum_{i:p(i)} \$ovrhd$. This is named *Least Loss*. Our goal is to minimize the total cost formula (12). Logically, we can assign a cost factor c of $\$amt - \$ovrhd$ to frauds and a factor c of $\$ovrhd$ to non-frauds. This reflects how the prediction errors will add to the total cost of a hypothesis. Since the overhead $\$ovrhd$ is not known to us, we chose to set $\$ovrhd \in \{60, 70, 80, 90\}$ to run four sets of experiments. We believe that these choices are realistic approximations to the real overhead. Detailed information on credit card cost model can be found in [13].

We do not have any "business-oriented" misclassification cost models for the remaining data sets. Instead, we varied the cost ratio R of positive vs. negative from 2 to 9.

We normalized each c_i to $[0, 1]$ for all data sets. The cost adjustment function β is chosen as: $\beta_-(c) = 0.5 \cdot c + 0.5$ and $\beta_+(c) = -0.5 \cdot c + 0.5$.

Training and Testing: In the credit card business, due to normal billing cycles, there is a two-month delay for data to be used to train a detection system applied to current transactions. We mirror this constraint by using one month data for training and data from two months later for testing. Our data set allowed us to form 10 such pairs of training and testing sets. For the other six data sets, we used 10-fold CV to average the results or used the specific testing data provided.

Weak Learner: We used Cohen's RIPPER [3] as the "weak" learner. We chose it because RIPPER provides an easy way to change the distribution of the training set. Since using the training set alone usually overly estimates the accuracy of a rule set, we used the Laplace estimate to generate the confidence ($|h(x)|$) for each rule [1]. RIPPER is cost-insensitive. We make it cost-sensitive, called cRIPPER, by supplying it with a distribution that is linear in the cost of each instance ($D_1(i) \propto c_i$).

Calculating $\alpha_\mathbf{t}$: Both AdaBoost and AdaCost were run to the 50^{th} round. We carefully engineered the bisection search algorithm with the first candidate α calculated by formula (9). The second point was searched with an exponential step increment. To avoid numerical errors introduced when adding a small number to a much bigger number, we used a vector v to compute $Z'(\alpha) = -\sum_i D(i)u_i e^{-\alpha u_i}$. For simplicity, we allow the index of v to be negative. The k-th element of the vector adds up ($w_i = -D(i)u_i e^{-\alpha u_i}$), if $10^{k-1} \leq |w_i| \leq 10^k$. We add every element in the vector starting from the smallest index (to avoid addition errors) to calculate Z'.

3.2 Results

We are interested in comparing cRIPPER (as a baseline), AdaCost and AdaBoost in several dimensions. First, for each data set and each cost model, we determine which algorithm has achieved the lowest cumulative misclassification cost and how many cases AdaCost is the clear winner. Second, we also seek to know, quantitatively, the difference in cumulative misclassification cost of AdaCost from AdaBoost and the baseline cRIPPER. It is interesting to measure the significance of these differences in terms of both reduction in misclassification loss and percentage of reduction. Additionally, we also want to see for all data sets and all cost models, how many of the times AdaCost has a lower cumulative cost than AdaBoost for all corresponding rounds of boosting. Finally, we are interested to know if AdaCost consumes more computing power. We grouped the results into two groups: the six data sets excluding Chase credit card and the Chase credit card alone.

Six Data Sets All the results are summarized in Table 2 and Figure 2. We use *percentage cumulative loss* (or $\%l$) to compare across data sets. $\%l$ is defined as $\frac{cumulative\ loss}{max\ loss}$, where *max loss* is the total loss when every instance is misclassified. Table 2 lists $\%l$ for cRIPPER (cRpr), $\%l$ at the end of 50^{th} round for AdaBoost (Bst) and AdaCost (Cst) for six data sets excluding Chase credit card. The data sets are numbered from 1 to 6. The information on each data set can be found in Table 1. The second column is the R or cost ratio chosen to go from 2 to 9. The last two columns list the difference of $\%l$ in both plain subtraction and percentage (parenthesized). $(C-B)(\%)$ includes values for $\%l_{AdaCost} - \%l_{AdaBoost}$ as well as $\frac{\%l_{AdaCost}-\%l_{AdaBoost}}{\%l_{AdaBoost}}$. $(C-P)(\%)$ lists similar values for AdaCost versus cRIPPER. We keep one decimal point significant digit for all results. When AdaCost is the best performer, it is highlighted in bold font.

In 42 (88% of 48) cases, AdaCost achieved the lowest misclassification loss. AdaBoost had the best performance in 5 (10% of 48) cases. AdaCost and AdaBoost tied at 1 run. The absolute reduction by AdaCost from AdaBoost (as shown in $(C-B)(\%)$ column) in these six data sets ranges from 0.1% (data set 1 or hypothyroid with ratio $R=7$) to 14.6% (data set 6 or wpbc with ratio $R=3$). The percentage increase goes from 2% (data set 4 or crx with ratio $R=6$) to 57% (data set 2 or boolean with ratio $R=3$). The improvements are clear. The difference of AdaCost from cRIPPER (as shown in column $(C-P)(\%)$) are even greater. The last row μ for every data set is the average percentage loss for all chosen cost ratios R.

We have plotted the individual error and error ratio points for these six data sets in Figure 2. Each point in the left figure is the ratio of AdaCost vs. AdaBoost, in other words, $\frac{\%l_{AdaCost}}{\%l_{AdaBoost}}$. We have drawn the points for all 8 ratios and all 50 boosting rounds. To avoid overlapping of these points, the boosting round t was given a random adjustment in the range of $[-0.45, +0.45]$. The majority of the points are below the "Ratio=1" line which means that in an overwhelming majority of cases, AdaCost has lower cost than AdaBoost. The figure on the right plots the same information in a different way where the value of each individual misclassification cost is shown. The majority of points are above the "$y=x$" line which also means that AdaCost clearly has lower cumulative cost.

Chase Credit Card The results are plotted in Figures 3 and 4. Figure 3 shows the average reduction of 10 months in percentage cumulative loss (defined as $\frac{cumulativeloss}{maximalloss-leastloss} * 100\%$) for AdaBoost and AdaCost for all 50 rounds and 4 overheads. We can clearly see that there is a consistent reduction of AdaCost over AdaBoost for all 400 ($= 50*2*4$) runs. The absolute amount of reduction is over 3%. We also observe that the speed of reduction by AdaCost is quicker than that of AdaBoost in all 4 figures. The speed is the quickest in the first few rounds. This means that in practice, we may not need to run AdaCost for many rounds.

Figure 4 plots the ratio of cumulative cost by AdaCost and AdaBoost. The figures are similar to those in Figure 2. We have plotted the results of all 10 pairs of training and test months over all rounds and overheads. Most of the points are below the "Ratio=1" line in the left drawing and above the "y=x" line in the right drawing, both implying that AdaCost has lower cumulative loss in an overwhelming number of cases.

Table 2: Percentage Cumulative Loss by cRIPPER, AdaBoost and AdaCost for Six Data Sets

$\mathcal{S}$	R	**cRpr**	**Bst**	**Cst**	**(C-B)(%)**	**(C-P)(%)**
	2	1.4	1.6	1.2	**-0.4(-25)**	**-0.2(-16)**
	3	1.8	2.0	1.6	**-0.3(-17)**	**-0.2(-10)**
	4	2.1	2.2	1.8	**-0.4(-16)**	**-0.3(-12)**
1	5	2.5	2.7	2.2	**-0.4(-16)**	**-0.3(-11)**
	6	3.2	3.0	2.5	**-0.6(-19)**	**-0.7(-23)**
	7	3.1	2.8	2.7	**-0.1(-3)**	**-0.4(-13)**
	8	3.0	3.1	2.7	**-0.4(-12)**	**-0.3(-10)**
	9	3.0	3.2	2.5	**-0.7(-23)**	**-0.5(-17)**
	μ	2.5	2.6	2.2	**-0.4 (-16.0)**	**-0.4(-14.2)**
	2	13.8	10.5	3.3	**-7.2(-69)**	**-10.6(-76)**
	3	14.2	11.6	5.0	**-6.6(-57)**	**-9.2(-65)**
	4	15.4	10.9	6.9	**-4.0(-37)**	**-8.5(-55)**
2	5	14.7	11.4	7.3	**-4.1(-36)**	**-7.4(-50)**
	6	13.9	9.3	8.1	**-1.3(-13)**	**-5.8(-42)**
	7	19.5	9.6	8.5	**-1.1(-11)**	**-11.0(-57)**
	8	18.0	9.6	8.3	**-1.3(-14)**	**-9.6(-54)**
	9	18.3	11.0	8.1	**-3.0(-27)**	**-10.2(-56)**
	μ	16.0	10.5	6.9	**-3.6 (-34.1)**	**-9.1(-56.7)**
	2	2.3	2.6	2.0	**-0.6(-24)**	**-0.3(-14)**
	3	4.1	3.5	3.1	**-0.4(-12)**	**-1.0(-25)**
	4	5.0	4.3	4.3	0.0(0)	**-0.7(-14)**
3	5	6.2	4.9	4.4	**-0.5(-10)**	**-1.8(-29)**
	6	6.5	7.0	5.5	**-1.5(-21)**	**-1.0(-15)**
	7	7.6	8.0	6.7	**-1.2(-15)**	**-0.9(-11)**
	8	6.7	7.6	6.1	**-1.5(-20)**	**-0.6(-9)**
	9	7.8	10.1	7.1	**-3.0(-30)**	**-0.7(-9)**
	μ	5.8	6.0	4.9	**-1.1 (-18.2)**	**-0.9(-14.9)**

$\mathcal{S}$	R	**cRpr**	**Bst**	**Cst**	**(C-B)(%)**	**(C-P)(%)**
	2	14.0	10.5	5.4	**-5.1(-48)**	**-8.6(-61)**
	3	11.7	12.4	12.1	**-0.3(-3)**	0.4(4)
	4	11.1	11.2	11.3	0.1(1)	0.2(2)
4	5	9.7	9.9	10.0	0.1(1)	0.3(3)
	6	9.8	7.4	7.3	**-0.1(-2)**	**-2.5(-25)**
	7	8.5	8.1	4.9	**-3.2(-39)**	**-3.6(-42)**
	8	8.1	10.6	8.7	**-2.0(-19)**	0.6(7)
	9	7.7	11.1	8.9	**-2.2(-20)**	1.2(15)
	μ	10.1	10.2	8.6	**-1.6 (-15.5)**	**-1.5(-14.8)**
	2	4.4	3.2	1.7	**-1.4(-45)**	**-2.7(-61)**
	3	3.7	3.4	1.8	**-1.6(-46)**	**-1.9(-51)**
	4	3.8	4.8	3.1	**-1.8(-37)**	**-0.7(-19)**
5	5	4.1	4.6	4.2	**-0.4(-9)**	0.1(2)
	6	3.5	3.6	2.2	**-1.4(-39)**	**-1.3(-38)**
	7	3.5	3.2	3.2	**-0.1(-2)**	**-0.3(-9)**
	8	3.3	3.5	2.2	**-1.4(-38)**	**-1.1(-34)**
	9	3.2	3.1	3.0	**-0.1(-3)**	**-0.2(-7)**
	μ	3.7	3.7	2.7	**-1.0 (-27.6)**	**-1.0(-27.7)**
	2	35.8	43.7	34.0	**-9.7(-22)**	**-1.8(-5)**
	3	38.9	35.1	20.5	**-14.6(-42)**	**-18.4(-47)**
	4	36.7	33.5	35.7	2.1(6)	**-1.0(-3)**
6	5	35.0	34.5	22.6	**-12.0(-35)**	**-12.4(-35)**
	6	31.2	18.7	11.1	**-7.6(-41)**	**-20.1(-64)**
	7	28.6	28.6	28.8	0.1(1)	0.2(1)
	8	24.6	24.8	25.1	0.4(2)	0.5(2)
	9	25.5	27.1	25.7	**-1.4(-5)**	0.2(1)
	μ	32.0	30.8	25.4	**-5.3 (-17.3)**	**-6.6(-20.6)**

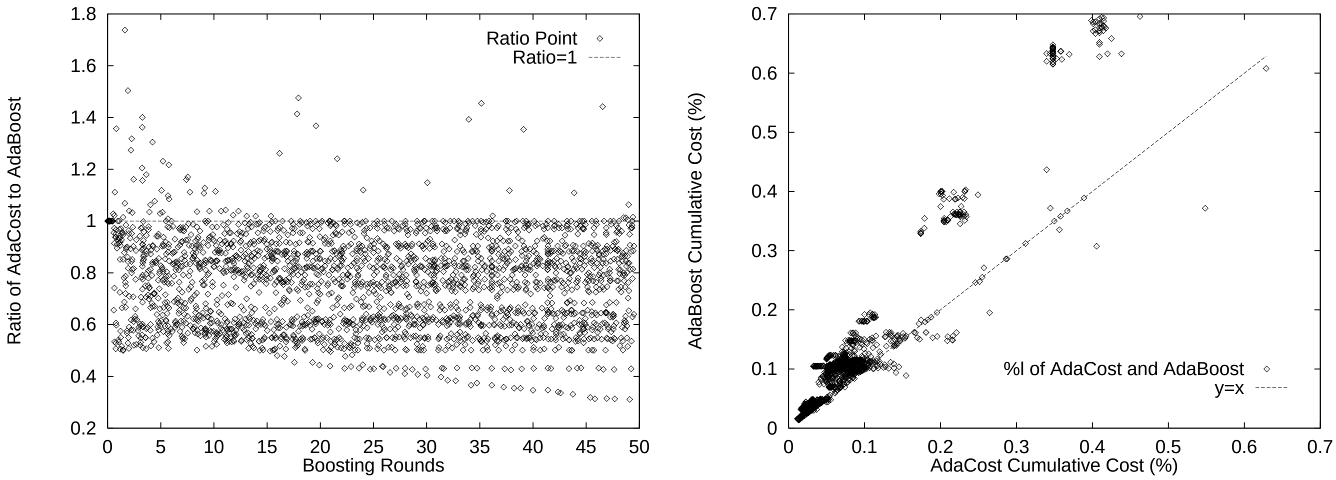

Figure 2: Cumulative Loss Ratio and Loss of AdaCost and AdaBoost for Six Data Sets

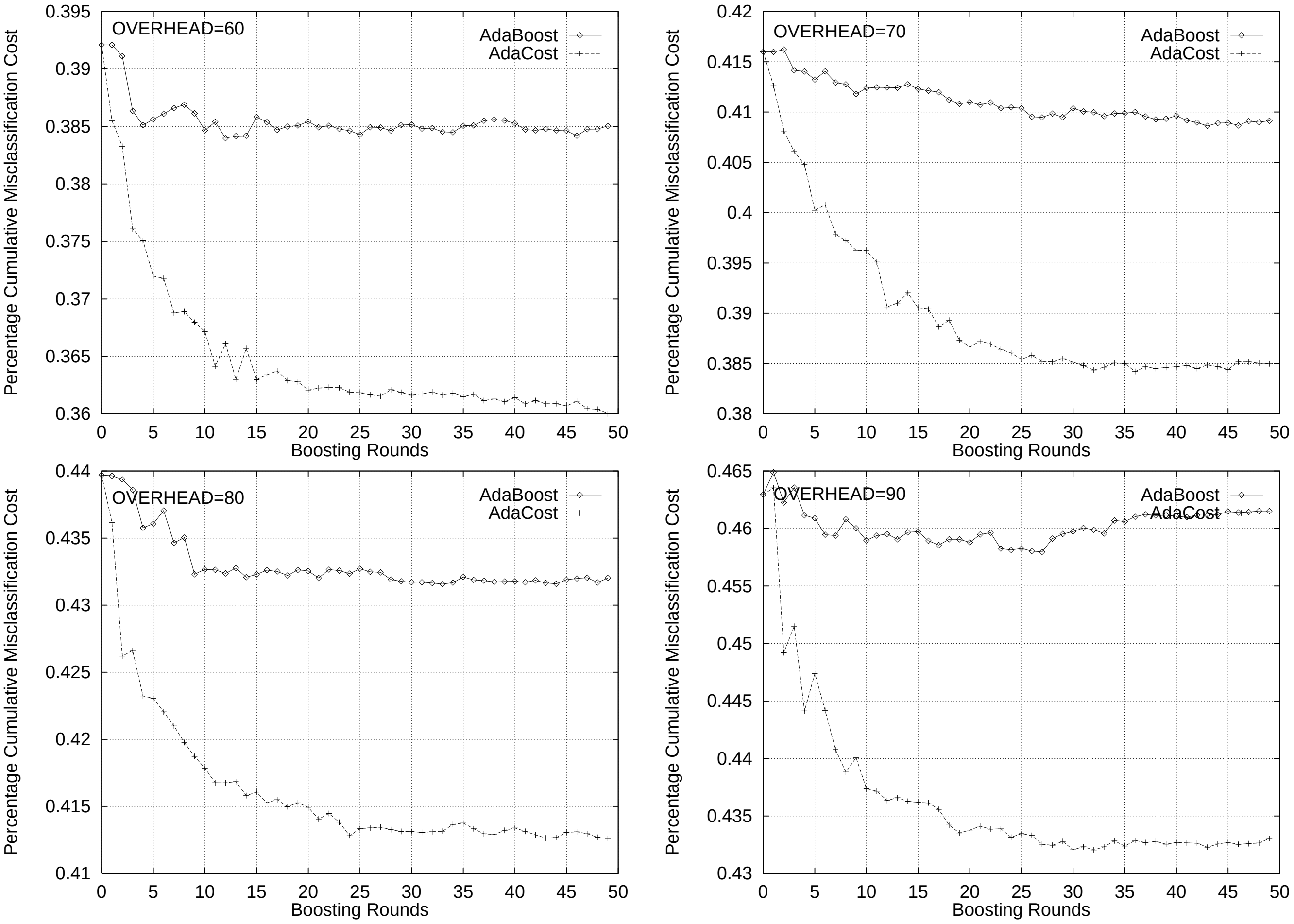

Figure 3: Cumulative Loss Ratio of AdaCost and AdaBoost for Chase Credit Card Data Set

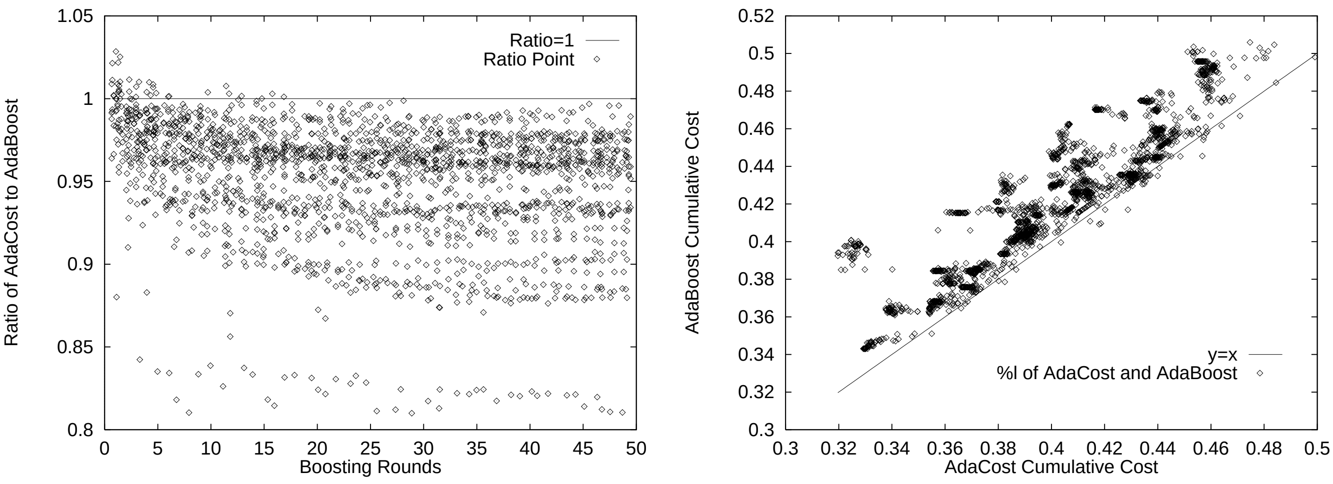

Figure 4: Cumulative Loss Ratio and Loss of AdaCost and AdaBoost on Chase Credit Card

4 Conclusion

This paper has raised an interesting question: can we improve AdaBoost based on a cost model? We have studied the problem of reducing misclassification cost using boosting methods and proposed the AdaCost algorithm. The intuition is that in addition to assigning high initial weights to costly examples, the weight updating rule should also take cost into account and increase the weights of costly misclassification more but decrease the weights of costly correct classification less. Based on Freund, Schapire and Singer's previous work, we have formally proved the upper bound of training cumulative misclassification cost and discussed the choice of α. We also attempted to apply AdaCost to other variants of AdaBoost.

We have empirically evaluated AdaCost and AdaBoost on seven data sets using both real world and artifical cost models and we have observed that AdaCost shows a consistent and significant reduction in misclassification cost over AdaBoost and it doesn't consume more computing power on average. One interesting experiment we have done, but not included in this paper, is to compare AdaBoost and AdaCost with a uniform initial distribution. In other words, we set $D(i) = \frac{1}{m}$. Chase credit card data was used in this study. The results show that AdaBoost reduces classification error significantly but doesn't reduce misclassification cost much. On the other hand, there is a significant reduction in misclassification cost by AdaCost. This experiment has, to some extent, provided additional evidence for the "benefits" of introducing cost into the weight updating rule.

Acknowledgments

This research is supported in part by grants from DARPA (F30602-96-1-0311) and NSF (IRI-96-32225 and CDA-96-25374). Our work has benefitted from in-depth discussions with Robert Schapire and Yoram Singer of AT&T Research Labs.

References

[1] K. Ali and M. Pazzani. (1996). Error Reduction through Learning Multiple Descriptions. *Machine Learning, 24(3)*, 173-202.

[2] P. Chan and S. Stolfo. (1998). Towards Scalable Learning with Non-Uniform Class and Cost Distributions: A Case Study in Credit Card Fraud Detection. In *Proc. Fourth Int'l Conf on Knowledge Discovery and Data Mining*, New York, New York. pp.164-168.

[3] W. Cohen. (1995). Fast Effective Rule Induction. In *Proc. Twelfth Int'l Conf on Machine Learning*, pp. 115-123, Morgan Kaufman.

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

[5] G. Karakoulas and J. Shawe-Taylor. (1998). Optimizing classifiers for imbalanced training sets. *NIPS* 1998

[6] N. Lavrac, D. Gamberger and P. Turney. (1996). Cost-sensitive feature reduction applied to a hybrid genetic algorithm. In *Proc of the Seventh Int'l Workshop on Algorithmic Learning Theory*, Sydney, Australia, pp 127-134.

[7] M. Pazzani, C. Merz, P. Murphy, K. Ali, T. Hume and C. Brunk. (1994). Reducing misclassification costs: Knowledge-intensive approaches to learning from noisy data In *Proc of the Eleventh Int'l Conf on Machine Learning*, pp:217-225, New Brunswick, New Jersey.

[8] F.J. Provost and T. Fawcett. (1997). Analysis and visualization of classifier performance: Comparison under imprecise class and cost distributions. In *Proc of the Third Int'l Conf. on Knowledge Discovery and Data Mining*.

[9] R. Schapire and Y. Singer. (1998). Improved boosting algorithms using confidence-rated predictions. In *Proceedings of the Eleventh Annual Conf on Computational Learning Theory*.

[10] R. Schapire and Y. Singer. (1998). BoosTexter: A system for multiclass multi-label text categorization. at `www.research.att.com/~schapire`.

[11] R. Schapire,Y. Singer and A. Singhal. (1998). Boosting and Rochhio applied to text filtering. In *SIGIR'98*.

[12] S. Stolfo, A. Prodromidis, S. Tselepsis, W. Lee, W. Fan and P. Chan. (1997). JAM: Java Agents for Meta-learning over Distributed Databases. In *Prod. Third Intl. Conf. Knowledge Discovery and Data Mining*.

[13] S. Stolfo, W. Fan, W. Lee, A. Prodromidis and P. Chan. (1997). Credit Card Fraud Detection Using Meta-learning: Issues and Initial Results. In *AAAI-97 Workshop on Fraud Detection and Risk Management*.

[14] K.M. Ting and Z. Zheng. (1998). Boosting Trees for Cost-Sensitive Classifications. In *Machine Learning: ECML-98: 10th European Conf on Machine Learning*, (pp. 190-195). Chemnitz, Germany:Springer.

[15] P. Turney. Cost-sensitive Learning Bibliographies. `ai.iit.nrc.ca/bibliographies/cost-sensitive.html`

[16] P. Turney. Types of Cost. at `ai.iit.nrc.ca/bibliographies/cost-types.html`

[17] P. Turney. (1995). Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm. In *Journal of Artificial Intelligence Research*, 2:369-409.