Enhancing the Performance of AdaBoost Algorithms by Introducing a Frequency Counting Factor for Weight Distribution Updating

Diego Alonso Fernández Merjildo and Lee Luan Ling

Department of Communications, DECOM
School of Electrical and Computer Engineering, FEEC
State University of Campinas, UNICAMP
Campinas, 13083-852, Brazil
{diegofer,lee}@decom.fee.unicamp.br

Abstract. This work presents a modified Boosting algorithm capable of avoiding training sample overfitting during training procedures. The proposed algorithm updates weight distributions according to amount of misclassified samples at each iteration training step. Experimental tests reveal that our approach has several advantages over many classical AdaBoost algorithms in terms of error generalization capacity, overfitting avoidance and superior classification performance.

Keywords: AdaBoost Algorithm, Weights Update, Frequency Factor, Misclassified Samples, Machine Learning.

1 Introduction

AdaBoost algorithm was probably the very first boosting algorithm widely used in real-time systems, due to its simplicity and adaptability. Freund and Schapire [1] introduced the adaptive Boosting algorithm (AdaBoost) based on a simple boosting algorithms developed by Valiant [2] and Schapire [3]. The Real AdaBoost algorithm is considered as the first variant of AdaBoost seeking to optimize the cost function of classifiers [4]. The Gentle AdaBoost uses Newton stepping to produce more stable and reliable ensembles [5]. The Modest AdaBoost provides lower generalization error but higher classification error rate during training stages when compared to the Gentle AdaBoost [6]. The FloatBoost removes less significant weak classifiers in order to reduce classification error rate [7]. The EmphasisBoost uses the so-called emphasis function of, in this case each sample is weighted according to a criterion defined by a parameter such that the training process focuses on critical samples [8].

Dietterich [9], Ratsch et al. [10] and Servedio [11] showed that AdaBoost algorithm has great tendencies to cause sample overfitting in the presence of high noise data. In order to avoid sample overfitting phenomena, Li et al. [12] proposed a modified method to update weights, focusing on the samples being

L. Alvarez et al. (Eds.): CIARP 2012, LNCS 7441, pp. 527–534, 2012.

misclassified by assigning large weights to samples erroneously classified. The proposed weight updating procedure may cause considerable distortion in weight distribution, cumulatively augmented by repeated iterations. In this sense, in this work we propose a new method to update sample weight distribution in order to avoid sample overfitting problems.

2 The Proposed AdaBoost Algorithm

2.1 The Conventional AdaBoost Algorithm

Adaboost algorithm combines several weak classifiers in such a way that the ensemble can improve their performance. The training ensemble consists of a set of samples $x_i \in X, i = 1 \ldots N$, with labels $y_i \in \{-1, 1\}$. At each round $t = 1, \ldots, T$, a new weak classifier is added implementing a function $h_t(x_i) : X \rightarrow [-1, 1]$. Through a gradient descent procedure, a classic AdaBoost algorithm attempts to minimize an exponential error function. $W_{t,i}$ denotes the weight that the t^{th} classifier function assign to x_i. Initially, a common weight value is adopted, $W_{1,i} = 1/N$. Then, the weight distribution is updated according to $W_{t+1,i} = W_{t,i} exp(-\alpha_t h_t(x_i) y_i)/Z_t$, where Z_t is a normalization factor assuring that $\sum_{i=1}^{N} W_{t,i} = 1$, and α_t is the importance assigned to the t^{th} weak classifier.

At each iteration, AdaBoost linearly combines weak classifiers as $F(x) = \sum_{j=1}^{T} \alpha_j h_j(x)$ in order to achieve a classification accuracy better than each weak classifier individually. As a result, the generalization error of the resulting linearly combined classifier becomes smaller and smaller as training iteration proceeds, possibly achieving zero classification error after several iterations.

2.2 The Improved AdaBoost Algorithm

In order to solve the overfitting problem detected during training processes, this work suggests a modified method to update the weight distribution, by introducing a new variable, denoted by δ_i, which is associated to each training sample. The proposed updating process is similar to that of the classical AdaBoost algorithm; thus, at each iteration step we add a weak classifier to the set of classifiers that are linearly combined to build a single strong classifier $F(x)$. In this way, the set of new variables, $\delta_i \in \Delta = \{0, \ldots, k\}$ is involved in the training and is responsible for preventing the increase of weights of misclassified samples.

Notice that in a classical Boosting procedure when a sample is misclassified, the associated weight has its value increased; on the other hand, when a sample is correctly classified, the associated weight is diminished. This weight updating strategy may cause sample overfitting due to excessive weight value accumulated in repeated iterations. In order to avoid this problem, in this work, we change the weight updating procedure by considering the frequency of occurrence of misclassified samples. In other words, the weights are only changed when the number of a sample being misclassified is above a predetermined threshold value, denoted by k frequency samples. Thus, the rise of the weight is not always

Algorithm 1. Proposed modified AdaBoost algorithm

- Given dataset $S = \{(x_i, y_i), ..., (x_n, y_n)\}$, where $y_i \in \{-1, +1\}$, with $\delta_i \in \Delta = \{0, ..., k\}$
- Initializing $\delta_i = 0$
- For $t = 1, ..., T$
  1. Normalizing the sample distribution: $W_{t,i}$
  2. Calling weak classification algorithm, and choosing the best weak classifier $h_t : S\{-1, +1\}$ with the lowest error ϵ_t
  3. Choosing $\alpha_t = \frac{1}{2} \ln \frac{1+\epsilon_t}{1-\epsilon_t}$
  4. Updating the sample weight distribution.

$$W_{t+1,i} = \begin{cases} W_t(i)exp(-\alpha_t) & h_t(x_i) = y_i \\ W_t(i)exp(\alpha_t) & h_t(x_i) \neq y_i \ \wedge \delta_i \geq k \\ W_t(i) & \text{otherwise} \end{cases} \quad (1)$$

  5. Updating the misclassified count (Frequency).

$$\delta_i = \begin{cases} \delta_i + 1 & h_t(x_i) \neq y_i \ \wedge \ \delta_i < k \\ 0 & \delta_i \geq k \\ \delta_i & \text{otherwise} \end{cases} . \quad (2)$$

- The final strong classifier is introduced as:

$$F(x) = sign(\sum_{t+1}^{T} \alpha_t h_t(x)) \quad (3)$$

performed continuously from iteration to iteration. The weight of a misclassified sample will be increased only if it was misclassified k times or more (see (1)).

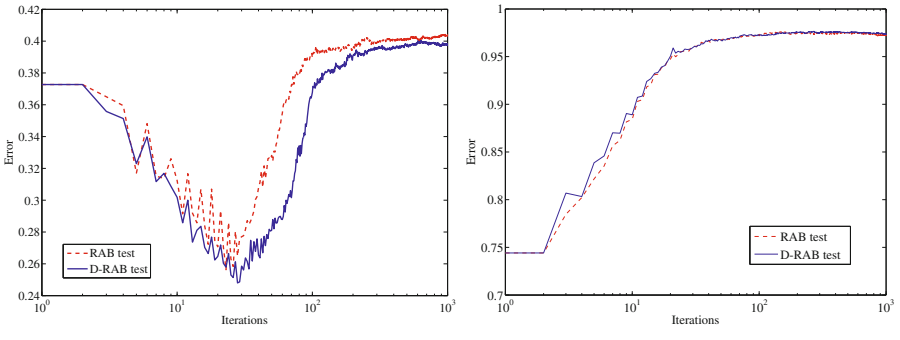
According to Algorithm 1, during the training phase, a dummy variable δ_i, belonging to the set $\{0, 1, 2, \ldots, k\}$, is used to indicating the number of times sample has already been misclassified. δ_i remains unchanged when the sample is classified correctly. δ_i is reset to zero when it is saturated, i.e. when $\delta_i = k$. The updating procedure of the variable δ_i is shown by (2) in Algorithm 1.

3 Experimental Investigation

The main objective of the experiments (computational trials) developed in this work consists in comparing the performance of the proposed method with those offered by other boosting approaches. Accordingly, most Boosting algorithms, reported in the literature, rely on a cost function optimization. We adopted the real data sets described in [13] and two artificially generated data sets in our experimental evaluation. Each dataset is splitted into a training set and a testing set under the 80:20 proportion. The maximum value for variable δ_i is limited to $k = 20$. We explicitly compare the performance of the proposed method with

that given by the Real AdaBoost Algorithm as illustrated by six classification error curves for six different data in Fig. 1 and 2. In addition, in Table 1 we compare the performance, in terms of average classification error rates, observed in both training and testing phases using thirteen different datasets [13].

According to [14] the Ringnorm dataset has a sphere surface separation. Such a geometric format prevents the Adaboost algorithms to guarantee good classification results. To contrast, our boosting method offered better performance for the testing subset with lower error rate almost in every iteration (see Figure 1a). The proposed method was designed to resist sample overfitting; in fact, our method outperforms the classical AdaBoost algorithm when data samples are noisy or separation surfaces are complex. As illustration, according to [14], the Twonorm dataset presents a planar separation surface what implies an easier classification task than the above case of the Ringnorm dataset (see Figures 1a and 1b). It is important to highlight that the comparison results are valid for all classical Adaboost algorithms (Real AdaBoost, GentleBoost, Modest AdaBoost, FloatBoost, EmphasisBoost, etc.), because the improved method described in this work can be introduced into these variants.



(a) *Ringnorm Test* (b) *Twonorm Test*

Fig. 1. Test error curves of classical approach (RAB - Real AdaBoost) and the approach of this work (D-RAB). The x-axis is on a logarithmic scale for easier comparison.

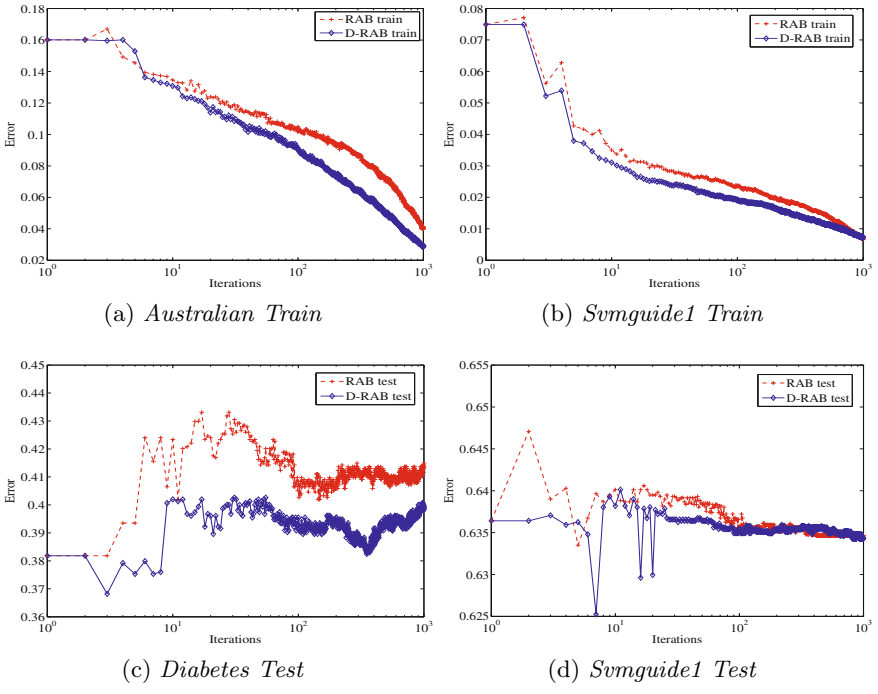
We also observed that there are significant differences between the proposed method and the Real AdaBoost in terms of error and convergence of the training process. As illustrated by Table 1, the proposed approach consistently performs better for the following datasets: *Australian, Bcancer, Diabetes, Fourclass, Liver, Splice, Svmguide1, W1a, Spam*. In terms of convergence speed, the proposed approach is higher than the classical approaches and also provides lower training errors. This is due to the fact the proposed updating scheme (using δ_i counter) has eliminated unnecessary and redundant weight updating steps. As a result, it is possible to use a reduced number of weak classifiers to build strong classifiers. Certainly this is a desirable property for real time applications such as real-time object detection. Figures 2a and 2b show that the method developed in this work may converge

faster than the classical approaches (in this case, when compared to the Real AdaBoost algorithm), in terms of the number of iterations (up to 1000 iterations).

Table 1. Results of Real Adaboost (RAB) and Real Adaboost using the δ_i variable (D-RAB) for training error and test error

Data set	Algorithm	Average Training error	Average Test error
australian	RAB	0.073797 ± 0.022533	0.73868 ± 0.016918
	D-RAB	**0.056123 ± 0.023557**	**0.73222 ± 0.032643**
bcancer	RAB	0.0020888 ± 0.0067642	**0.7033 ± 0.013678**
	D-RAB	**0.0017302 ± 0.0054692**	0.70771 ± 0.015291
diabetes	RAB	0.12941 ± 0.026878	0.41084 ± 0.0042467
	D-RAB	**0.11787 ± 0.029094**	**0.39325 ± 0.0041971**
fourclass	RAB	0.43295 ± 0.011658	-3.9745 ± 19.4456
	D-RAB	**0.37657 ± 0.039754**	**-4.4004 ± 30.8115**
heart	RAB	**0.0186255 ± 0.03319**	0.41922 ± 0.012025
	D-RAB	0.018817 ± 0.029935	**0.41492 ± 0.0099736**
ionosphere	RAB	**0.0018697 ± 0.011596**	**0.37 ± 0.016583**
	D-RAB	0.0020264 ± 0.011412	0.37452 ± 0.021583
liver	RAB	0.080236 ± 0.045446	**0.50874 ± 0.00487**
	D-RAB	**0.077945 ± 0.045223**	0.51065 ± 0.00466
sonar	RAB	**0.00187 ± 0.0156**	**0.51398 ± 0.00681**
	D-RAB	0.00205 ± 0.0159	0.51489 ± 0.00686
splice	RAB	0.017477 ± 0.02460	0.31484 ± 0.01243
	D-RAB	**0.017088 ±0.02410**	**0.30438 ± 0.01441**
svmguise1	RAB	0.01529 ± 0.00687	0.63523 ± 0.001161
	D-RAB	**0.01294 ± 0.00571**	**0.63519 ±0.000723**
svmguide3	RAB	**0.05238 ± 0.03806**	0.38258 ± 0.04875
	D-RAB	0.05560 ± 0.03602	**0.36418 ± 0.04123**
w1a	RAB	0.017522 ± 0.005182	**0.010509 ± 0.000903**
	D-RAB	**0.016046 ±0.005242**	0.011638 ± 0.001349
spam	RAB	0.038357 ± 0.0011447	-0.41031 ± 2.0464
	D-RAB	**0.034284 ± 0.0010706**	**-0.50353 ± 2.2454**

On the other hand, it was computed the test error in terms of the generalization ability. The behavior of the classical approaches may generate sample overfitting on several cases; that is, the performance on the training examples still increases while the performance on unseen data becomes worse, as presented by Dietterich [9], Ratsch et al. [10] and Servedio [11]. However, the experiments developed in this work (computational trials) showed that this approach may be more resistant to overfitting problems. Table 1 also shows the performance of the algorithms in the testing phase. Real AdaBoost algorithm is slightly better in five cases (*bcancer, ionosphere, liver, sonar, w1a*), while our method has better performance, in terms of better lower average test error, in eight distinct cases (*australian, diabetes, fourclass, heart, splice, svmguide1, smguide3, spam*). Now, in terms of generalization ability our approach is considerably better than those classical AdaBoost ones. This fact is illustrated by Fig. 2c when using the



(a) *Australian Train* (b) *Svmguide1 Train*

(c) *Diabetes Test* (d) *Svmguide1 Test*

Fig. 2. Train error curves and Test error curves of classical approach (RAB - Real AdaBoost) and the approach of this work (D-RAB). The x-axis is on a logarithmic scale for easier comparison.

diabetes database. Similar results were obtained for the *svmguide* dataset (see Figure 2d), where a lower average test error is obtained.

4 Discussion and Conclusions

4.1 Algorithm Structure and Capabilities

In this section, we provide some insights into the quality of the proposed method. For this, we use the margin theory proposed by Schapire et al. [15] who provided one the most popular explanations to the success of AdaBoost, i.e, it tends to improve the *margin* even after the error. The *margin* of an example with respect to a classifier is a measure of the confidence of the classification result. Thus, consider binary classification problems, where the examples are drawn independently according to an underlying distribution W over $X \times \{-1, +1\}$, and X is an instance space. Let to include H variable as a classifier space. A weak classifier $h \in H$ is a mapping from X to $-1, +1$. A linear convex combination is of the form $F(x) = \sum \alpha_i h_i(x)$, where $\sum \alpha_i = 1, \alpha_i \geq 0$. An error occurs on an example (x, y) if and only if:

$$yF(x) \leq 0. \tag{4}$$

The value of $yF(x)$ reflects the confidence of the prediction. Therefore, $yF(x)$ is called the *margin* for (x, y) with respect to F, where $yF(x)$ is the difference between the weights assigned to those weak classifiers that correctly classify (x, y) and the weights assigned to those that misclassify the example. We use $P_X(F(x, y))$ to denote the probability with respect to choosing an example (x, y) uniformly at random from the training set X, where, $P_X(yf(x) \leq 0)$ is the generalization error which we want to bound. Considering the margins over the whole set of training examples, we can regard $P_X(yF(x) \leq \beta)$ as a distribution over $\beta(-1 \leq \beta \leq 1)$, which is the fraction of training examples whose margin is at most β. This distribution is referred to as the margin distribution. On the other hand, during weights update process performed at each iteration of the training process, the misclassified samples in previous iterations are focused, i.e, those misclassified samples have a *low margin*. In each subsequent iteration, the classic AdaBoost algorithm try to increase the margins of those training samples; thus the distribution at each step can also rewritten as:

$$W_t(i) = \frac{exp(-yf(x_i))}{Z} \tag{5}$$

where, it is possible to assign high weights to samples misclassified in previous iterations. However, this operation may result in provoking misclassification of samples with *higher margin* at a given iteration; and, therefore, reducing their *margins*. Under such a circumstance, it is natural to ask whether it is a good practice to reduce the *margin* in the beginning of the training process. Apparently it is not a wise measure because it could also reduce the *margins* of samples that are not "truly" misclassified, since this could decrease generalization ability of the AdaBoost algorithm; and, therefore, causing higher errors in testing stages. Such a phenomenon is known as overfitting. In order to solve this overfitting problem, we introduce and use δ_i variable to monitoring the misclassification rate of those samples with *low margins*, avoiding excessive reduction of *margins* and simultaneously maintaining its generalization ability.

4.2 Conclusions

In this paper, we present and introduce a novel weight updating procedure to improve the performance of the AdaBoost algorithm. Experimental results validated the proposed method which has the following qualities:

- The proposed method enables faster convergence than the Classical AdaBoost, providing lower misclassification error rates. The fast convergence is the consequence of using a smaller number of weak classifiers to build the strong classifier. Such structure simplification has some desirable implication including real time applications such as object detection, where on line response is critical.
- The approach described in this work makes boosting algorithms less noise sensitive.

References

[1] Freund, Y., Schapire, R.E.: A Decision-Theoretic Generalization of On-line Learning and an Application to Boosting. In: Vitányi, P.M.B. (ed.) EuroCOLT 1995. LNCS, vol. 904, pp. 23–37. Springer, Heidelberg (1995)
[2] Valiant, L.G.: A theory of the learnable. Commun. ACM 27, 1134–1142 (1984)
[3] Schapire, R.E.: The strength of weak learnability. Mach. Learn. 5, 197–227 (1990)
[4] Schapire, R.E., Singer, Y.: Improved boosting algorithms using confidence-rated predictions. Mach. Learn. 37, 297–336 (1999)
[5] Friedman, J., Hastie, T., Tibshirani, R.: Additive Logistic Regression: a Statistical View of Boosting. The Annals of Statistics 38(2) (2000)
[6] Vezhnevets, A., Vezhnevets, V.: 'modest adaboost' - teaching adaboost to generalize better. In: GRAPHICON
[7] Li, S., Zhang, Z.: Floatboost learning and statistical face detection. IEEE Transactions on Pattern Analysis and Machine Intelligence 26(9), 1112–1123 (2004)
[8] Gómez-Verdejo, V., Ortega-Moral, M., Arenas-García, J., Figueiras-Vidal, A.R.: Boosting by weighting critical and erroneous samples. Neurocomput. 69, 679–685 (2006)
[9] Dietterich, T.G.: Ensemble Methods in Machine Learning. In: Kittler, J., Roli, F. (eds.) MCS 2000. LNCS, vol. 1857, pp. 1–15. Springer, Heidelberg (2000)
[10] Rätsch, G., Onoda, T., Müller, K.R.: Soft margins for adaboost. Mach. Learn. 42, 287–320 (2001)
[11] Servedio, R.A.: Smooth boosting and learning with malicious noise. J. Mach. Learn. Res. 4, 633–648 (2003)
[12] Li, G., Xu, Y., Wang, J.: An improved adaboost face detection algorithm based on optimizing skin color model. In: 2010 Sixth International Conference on Natural Computation (ICNC), vol. 4, pp. 2013–2015 (August 2010)
[13] Chang, C.C., Lin, C.J.: LIBSVM: A library for support vector machines. ACM Transactions on Intelligent Systems and Technology 2, 27:1–27:27 (2011), Software `http://www.csie.ntu.edu.tw/~cjlin/libsvm`
[14] Breiman, L.: Arcing classifiers. Annals of Statistics 26(3), 801–824 (1998)
[15] Schapire, R.E., Freund, Y., Bartlett, P., Lee, W.S.: Boosting the margin: a new explanation for the effectiveness of voting methods. Annals of Statistics 26(5), 1651–1686 (1998)